

# Timed Automata

Simon Wimmer

February 4, 2026

## Abstract

Timed automata are a widely used formalism for modeling real-time systems, which is employed in a class of successful model checkers such as UPPAAL [LPY97], HyTech [HHWt97] or Kronos [Yov97]. This work formalizes the theory for the subclass of diagonal-free timed automata, which is sufficient to model many interesting problems. We first define the basic concepts and semantics of diagonal-free timed automata. Based on this, we prove two types of decidability results for the language emptiness problem.

The first is the classic result of Alur and Dill [AD90, AD94], which uses a finite partitioning of the state space into so-called *regions*.

Our second result focuses on an approach based on *Difference Bound Matrices (DBMs)*, which is practically used by model checkers. We prove the correctness of the basic forward analysis operations on DBMs. One of these operations is the Floyd-Warshall algorithm for the all-pairs shortest paths problem. To obtain a finite search space, a widening operation has to be used for this kind of analysis. We use Patricia Bouyer's [Bou04] approach to prove that this widening operation is correct in the sense that DBM-based forward analysis in combination with the widening operation also decides language emptiness. The interesting property of this proof is that the first decidability result is reused to obtain the second one.

## Contents

<b>1</b>	<b>Miscellaneous</b>	<b>4</b>
1.1	Lists . . . . .	4
1.2	Streams . . . . .	11
1.3	Mixed Material . . . . .	19
<b>2</b>	<b>Graphs</b>	<b>39</b>
2.1	Basic Definitions and Theorems . . . . .	40
2.2	Graphs with a Start Node . . . . .	52
2.3	Subgraphs . . . . .	55
2.4	Bundles . . . . .	59

2.5	Directed Acyclic Graphs . . . . .	60
2.6	Finite Graphs . . . . .	61
2.7	Graph Invariants . . . . .	61
2.8	Simulations and Bisimulations . . . . .	64
2.9	CTL . . . . .	78
<b>3</b>	<b>Basic Definitions and Semantics</b>	<b>85</b>
3.1	Syntactic Definition . . . . .	85
3.2	Operational Semantics . . . . .	87
3.3	Contracting Runs . . . . .	89
3.4	Zone Semantics . . . . .	91
3.5	From Clock Constraints to DBMs . . . . .	94
3.6	Semantics Based on DBMs . . . . .	103
<b>4</b>	<b>Refinement to <math>\beta</math>-regions</b>	<b>110</b>
4.1	Definition . . . . .	110
4.2	Basic Properties . . . . .	113
4.3	Approximation with $\beta$ -regions . . . . .	124
4.4	Computing $\beta$ -Approximation . . . . .	130
4.5	Auxiliary $\beta$ -boundedness Theorems . . . . .	155
<b>5</b>	<b>The Classic Construction for Decidability</b>	<b>178</b>
5.1	Definition of Regions . . . . .	178
5.2	Basic Properties . . . . .	179
5.3	Set of Regions . . . . .	189
5.4	Compability With Clock Constraints . . . . .	225
5.5	Compability with Resets . . . . .	229
5.6	A Semantics Based on Regions . . . . .	247
5.7	Correct Approximation of Zones with $\alpha$ -regions . . . . .	253
5.8	Old Variant Using a Global Set of Regions . . . . .	254
5.9	A Zone Semantics Abstracting with $Closure_\alpha$ . . . . .	260
5.10	New Variant . . . . .	270
5.11	A Semantics Based on Localized Regions . . . . .	271
5.12	A New Zone Semantics Abstracting with $Closure_{\alpha,l}$ . . . . .	276
<b>6</b>	<b>Correctness of <math>\beta</math>-approximation from <math>\alpha</math>-regions</b>	<b>282</b>
6.1	Preparing Bouyer's Theorem . . . . .	283
6.2	Bouyer's Main Theorem . . . . .	302
6.3	Nice Corollaries of Bouyer's Theorem . . . . .	332
6.4	A New Zone Semantics Abstracting with $Approx_\beta$ . . . . .	334

<b>7</b>	<b>Simulation Graphs</b>	<b>341</b>
7.1	Simulation Graphs . . . . .	341
7.2	Poststability . . . . .	345
7.3	Prestability . . . . .	347
7.4	Double Simulation . . . . .	350
7.5	Finite Graphs . . . . .	354
7.6	Complete Simulation Graphs . . . . .	359
7.7	Finite Complete Double Simulations . . . . .	363
7.8	Encoding of Properties in Runs . . . . .	370
7.9	Instantiation of Simulation Locales . . . . .	409
<b>8</b>	<b>Forward Analysis with DBMs and Widening</b>	<b>431</b>
8.1	DBM-based Semantics with Normalization . . . . .	432
8.2	Additional Useful Properties of the Normalized Semantics . . . . .	447
8.3	Appendix: Standard Clock Numberings for Concrete Models . . . . .	448

# 1 Miscellaneous

## 1.1 Lists

```
theory More-List1
  imports
    Main
    Instantiate-Existentials
begin
```

### 1.1.1 First and Last Elements of Lists

```
lemma (in -) hd-butlast-last-id:
  hd xs # tl (butlast xs) @ [last xs] = xs if length xs > 1
  using that by (cases xs) auto
```

### 1.1.2 list-all

```
lemma (in -) list-all-map:
  assumes inv:  $\bigwedge x. P x \implies \exists y. f y = x$ 
  and all: list-all P as
  shows  $\exists as'. \text{map } f as' = as$ 
  using all
  apply (induction as)
  apply (auto dest!: inv)
  subgoal for as' a
  by (inst-existentials a # as') simp
done
```

### 1.1.3 list-all2

```
lemma list-all2-op-map-iff:
  list-all2 ( $\lambda a b. b = f a$ ) xs ys  $\longleftrightarrow$  map f xs = ys
  unfolding list-all2-iff
  proof (induction xs arbitrary: ys)
  case Nil
  then show ?case by auto
  next
  case (Cons a xs ys)
  then show ?case by (cases ys) auto
qed
```

```
lemma list-all2-last:
  R (last xs) (last ys) if list-all2 R xs ys xs  $\neq []$ 
  using that
```

**unfolding** *list-all2-iff*  
**proof** (*induction xs arbitrary: ys*)  
  **case** *Nil*  
  **then show** *?case* **by** *simp*  
**next**  
  **case** (*Cons a xs ys*)  
  **then show** *?case* **by** (*cases ys*) *auto*  
**qed**

**lemma** *list-all2-set1*:  
 $\forall x \in \text{set } xs. \exists xa \in \text{set } as. P \ x \ xa$  **if** *list-all2 P xs as*  
**using** *that*  
**proof** (*induction xs arbitrary: as*)  
  **case** *Nil*  
  **then show** *?case* **by** *auto*  
**next**  
  **case** (*Cons a xs as*)  
  **then show** *?case* **by** (*cases as*) *auto*  
**qed**

**lemma** *list-all2-swap*:  
 $\text{list-all2 } P \ xs \ ys \longleftrightarrow \text{list-all2 } (\lambda \ x \ y. P \ y \ x) \ ys \ xs$   
**unfolding** *list-all2-iff* **by** (*fastforce simp: in-set-zip*)**+**

**lemma** *list-all2-set2*:  
 $\forall x \in \text{set } as. \exists xa \in \text{set } xs. P \ xa \ x$  **if** *list-all2 P xs as*  
**using** *that* **by**  $-$  (*rule list-all2-set1, subst (asm) list-all2-swap*)

#### 1.1.4 Distinct lists

**lemma** *distinct-length-le*:  $\text{finite } s \implies \text{set } xs \subseteq s \implies \text{distinct } xs \implies \text{length } xs \leq \text{card } s$   
**by** (*metis card-mono distinct-card*)

#### 1.1.5 filter

**lemma** *filter-eq-appendD*:  
 $\exists \ xs' \ ys'. \text{filter } P \ xs' = xs \wedge \text{filter } P \ ys' = ys \wedge as = xs' @ ys'$  **if** *filter P as = xs @ ys*  
**using** *that*  
**proof** (*induction xs arbitrary: as*)  
  **case** *Nil*  
  **then show** *?case*  
  **by** (*inst-existentials [] :: 'a list as*) *auto*

**next**  
**case** (*Cons a xs*)  
**from** *filter-eq-ConsD*[*OF Cons.prem[simplified]*] **obtain** *us vs* **where**  
*as = us @ a # vs  $\forall u \in \text{set } us. \neg P u P a$  filter  $P vs = xs @ ys$*   
**by** *auto*  
**moreover from** *Cons.IH*[*OF  $\langle - = xs @ ys \rangle$* ] **obtain** *xs' ys* **where**  
*filter  $P xs' = xs vs = xs' @ ys$*   
**by** *auto*  
**ultimately show** *?case*  
**by** (*inst-existentials us @ [a] @ xs' ys*) *auto*  
**qed**

**lemma** *list-all2-elim-filter*:  
**assumes** *list-all2 P xs us  $x \in \text{set } xs$*   
**shows** *length (filter (P x) us)  $\geq 1$*   
**using** *assms* **by** (*induction xs arbitrary: us*) (*auto simp: list-all2-Cons1*)

**lemma** *list-all2-replicate-elim-filter*:  
**assumes** *list-all2 P (concat (replicate n xs)) ys  $x \in \text{set } xs$*   
**shows** *length (filter (P x) ys)  $\geq n$*   
**using** *assms*  
**by** (*induction n arbitrary: ys; fastforce dest: list-all2-elim-filter simp: list-all2-append1*)

### 1.1.6 Sublists

**lemma** *nths-split*:  
*nths xs (A  $\cup$  B) = nths xs A @ nths xs B* **if**  $\forall i \in A. \forall j \in B. i < j$   
**using** *that*  
**proof** (*induction xs arbitrary: A B*)  
**case** *Nil*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*Cons a xs*)  
**let** *?A = {j. Suc j  $\in$  A}* **and** *?B = {j. Suc j  $\in$  B}*  
**from** *Cons.prem* **have** *\**:  $\forall i \in ?A. \forall a \in ?B. i < a$   
**by** *auto*  
**have** [*simp*]:  $\{j. \text{Suc } j \in A \vee \text{Suc } j \in B\} = ?A \cup ?B$   
**by** *auto*  
**show** *?case*  
**unfolding** *nths-Cons*  
**proof** (*clarsimp, safe, goal-cases*)  
**case** *2*  
**with** *Cons.prem* **have** *A = {}*

**by auto**  
**with Cons.IH[OF \*] show ?case by auto**  
**qed (use Cons.premis Cons.IH[OF \*] in auto)**  
**qed**

**lemma nth-nth:**  
 $nths\ xs\ \{i\} = [xs\ !\ i]$  **if**  $i < length\ xs$   
**using that**  
**proof (induction xs arbitrary: i)**  
**case Nil**  
**then show ?case by simp**  
**next**  
**case (Cons a xs)**  
**then show ?case**  
**by (cases i) (auto simp: nth-Cons)**  
**qed**

**lemma nth-shift:**  
 $nths\ (xs\ @\ ys)\ S = nths\ ys\ \{x - length\ xs \mid x. x \in S\}$  **if**  
 $\forall i \in S. length\ xs \leq i$   
**using that**  
**proof (induction xs arbitrary: S)**  
**case Nil**  
**then show ?case by auto**  
**next**  
**case (Cons a xs)**  
**have [simp]:  $\{x - length\ xs \mid x. Suc\ x \in S\} = \{x - Suc\ (length\ xs) \mid x. x \in S\}$  if  $0 \notin S$**   
**using that apply safe**  
**apply force**  
**subgoal for  $x\ x'$**   
**by (cases  $x'$ ) auto**  
**done**  
**from Cons.premis show ?case**  
**by (simp, subst nth-Cons, subst Cons.IH; auto)**  
**qed**

**lemma nth-eq-ConsD:**  
**assumes  $nths\ xs\ I = x \# as$**   
**shows**  
 $\exists\ ys\ zs.$   
 $xs = ys\ @\ x \# zs \wedge length\ ys \in I \wedge (\forall i \in I. i \geq length\ ys)$   
 $\wedge nths\ zs\ (\{i - length\ ys - 1 \mid i. i \in I \wedge i > length\ ys\}) = as$   
**using assms**

```

proof (induction xs arbitrary: I x as)
  case Nil
  then show ?case by simp
next
  case (Cons a xs)
  from Cons.prem1 show ?case
  unfolding nth1-Cons
  apply (auto split: if-split-asm)
  subgoal
    by (inst-existentials [] :: 'a list xs; force intro: arg-cong2[of xs xs - -
nth1])
  subgoal
    apply (drule Cons.IH)
    apply safe
    subgoal for ys zs
      apply (inst-existentials a # ys zs)
      apply simp+
      apply standard
      subgoal for i
        by (cases i; auto)
      apply (rule arg-cong2[of zs zs - - nth1])
      apply simp
      apply safe
      subgoal for - i
        by (cases i; auto)
      by force
    done
  done
qed

```

**lemma** nth1-out-of-bounds:  
 $\text{nth1 } xs \ I = []$  **if**  $\forall i \in I. i \geq \text{length } xs$

```

proof -
  have
     $\forall N \ as.$ 
     $(\exists n. n \in N \wedge \neg \text{length } (as::'a \text{ list}) \leq n)$ 
     $\vee (\forall asa. \text{nth1 } (as @ asa) \ N = \text{nth1 } asa \ \{n - \text{length } as \mid n. n \in N\})$ 
  using nth1-shift by blast
  then have
     $\bigwedge as. \text{nth1 } as \ \{n - \text{length } xs \mid n. n \in I\} = \text{nth1 } (xs @ as) \ I$ 
     $\vee \text{nth1 } (xs @ []) \ I = []$ 
  using that by fastforce
  then have  $\text{nth1 } (xs @ []) \ I = []$ 

```

**by** (*metis* (*no-types*) *nths-nil*)  
**then show** *?thesis*  
**by** *simp*  
**qed**

**lemma** *nths-eq-appendD*:

**assumes**  $nths\ xs\ I = as\ @\ bs$

**shows**

$\exists\ ys\ zs.$

$xs = ys\ @\ zs \wedge nths\ ys\ I = as$

$\wedge nths\ zs\ \{i - length\ ys \mid i. i \in I \wedge i \geq length\ ys\} = bs$

**using** *assms*

**proof** (*induction as arbitrary: xs I*)

**case** *Nil*

**then show** *?case*

**by** (*inst-existentials [] :: 'a list nths bs*) *auto*

**next**

**case** (*Cons a ys xs*)

**from** *nths-eq-ConsD[of xs I a ys @ bs] Cons.prem*s

**obtain** *ys' zs'* **where**

$xs = ys' @ a \# zs'$

$length\ ys' \in I$

$\forall i \in I. i \geq length\ ys'$

$nths\ zs' \{i - length\ ys' - 1 \mid i. i \in I \wedge i > length\ ys'\} = ys @ bs$

**by** *auto*

**moreover from** *Cons.IH[OF <nths zs' - = ->]* **obtain** *ys'' zs''* **where**

$zs' = ys'' @ zs''$

$ys = nths\ ys'' \{i - length\ ys' - 1 \mid i. i \in I \wedge length\ ys' < i\}$

$bs = nths\ zs'' \{i - length\ ys'' \mid i. i \in \{i - length\ ys' - 1 \mid i. i \in I \wedge length\ ys' < i\} \wedge length\ ys'' \leq i\}$

**by** *auto*

**ultimately show** *?case*

**apply** (*inst-existentials ys' @ a # ys'' zs''*)

**apply** (*simp; fail*)

**subgoal**

**by** (*simp add: nths-out-of-bounds nths-append nths-Cons*)

(*rule arg-cong2[of ys'' ys'' - - nths]; force*)

**subgoal**

**by** *safe* (*rule arg-cong2[of zs'' zs'' - - nths]; force*)

**done**

**qed**

**lemma** *filter-nths-length*:

$length\ (filter\ P\ (nths\ xs\ I)) \leq length\ (filter\ P\ xs)$

```

proof (induction xs arbitrary: I)
  case Nil
  then show ?case
  by simp
next
  case Cons
  then show ?case

proof –
  fix a :: 'a and xsa :: 'a list and Ia :: nat set
  assume a1:  $\bigwedge I. \text{length} (\text{filter } P (\text{nths } xsa I)) \leq \text{length} (\text{filter } P xsa)$ 
  have f2:
     $\forall b \text{ bs } N. \text{if } 0 \in N \text{ then } \text{nths} ((b::'a) \# \text{bs}) N =$ 
       $[b] @ \text{nths } \text{bs} \{n. \text{Suc } n \in N\} \text{ else } \text{nths} (b \# \text{bs}) N = [] @ \text{nths } \text{bs}$ 
 $\{n. \text{Suc } n \in N\}$ 
    by (simp add: nths-Cons)
  have f3:
     $\text{nths} (a \# xsa) Ia = [] @ \text{nths } xsa \{n. \text{Suc } n \in Ia\}$ 
     $\longrightarrow \text{length} (\text{filter } P (\text{nths} (a \# xsa) Ia)) \leq \text{length} (\text{filter } P xsa)$ 
    using a1 by (metis append-Nil)
  have f4:  $\text{length} (\text{filter } P (\text{nths } xsa \{n. \text{Suc } n \in Ia\})) + 0 \leq \text{length} (\text{filter}$ 
 $P xsa) + 0$ 
    using a1 by simp
  have f5:
     $\text{Suc} (\text{length} (\text{filter } P (\text{nths } xsa \{n. \text{Suc } n \in Ia\})) + 0)$ 
     $= \text{length} (a \# \text{filter } P (\text{nths } xsa \{n. \text{Suc } n \in Ia\}))$ 
    by force
  have f6:  $\text{Suc} (\text{length} (\text{filter } P xsa) + 0) = \text{length} (a \# \text{filter } P xsa)$ 
    by simp
  { assume  $\neg \text{length} (\text{filter } P (\text{nths} (a \# xsa) Ia)) \leq \text{length} (\text{filter } P (a$ 
 $\# xsa))$ 
    { assume  $\text{nths} (a \# xsa) Ia \neq [a] @ \text{nths } xsa \{n. \text{Suc } n \in Ia\}$ 
      moreover
      { assume
           $\text{nths} (a \# xsa) Ia = [] @ \text{nths } xsa \{n. \text{Suc } n \in Ia\}$ 
           $\wedge \text{length} (\text{filter } P (a \# xsa)) \leq \text{length} (\text{filter } P xsa)$ 
          then have  $\text{length} (\text{filter } P (\text{nths} (a \# xsa) Ia)) \leq \text{length} (\text{filter } P$ 
 $(a \# xsa))$ 
            using a1 by (metis (no-types) append-Nil filter.simps(2) impos-
            sible-Cons) }
          ultimately have  $\text{length} (\text{filter } P (\text{nths} (a \# xsa) Ia)) \leq \text{length} (\text{filter}$ 
 $P (a \# xsa))$ 
            using f3 f2 by (meson dual-order.trans le-cases) }
          then have  $\text{length} (\text{filter } P (\text{nths} (a \# xsa) Ia)) \leq \text{length} (\text{filter } P (a$ 

```

```

# xsa))
  using f6 f5 f4 a1 by (metis Suc-le-mono append-Cons append-Nil
filter.simps(2)) }
  then show length (filter P (nth (a # xsa) Ia)) ≤ length (filter P (a #
xsa))
    by meson
  qed
qed
end

```

## 1.2 Streams

**theory** *Stream-More*

**imports**

*Transition-Systems-and-Automata.Sequence-LTL*

*Instantiate-Existentials*

*HOL-Library.Rewrite*

**begin**

**lemma** *list-all-stake-least*:

*list-all (Not ∘ P) (stake (LEAST n. P (xs !! n)) xs) (is ?G) if ∃ n. P (xs !! n)*

**proof** (*rule ccontr*)

let ?n = *LEAST n. P (xs !! n)*

assume  $\neg ?G$

then have  $\exists x \in \text{set } (\text{stake } ?n \text{ xs}). P x$  **unfolding** *list-all-iff* **by** *auto*

then obtain *n'* where  $n' < ?n$   $P (xs !! n')$  **using** *set-stake-snth* **by** *metis*

with *Least-le[ $\text{of } \lambda n. P (xs !! n) n'$ ]* **show** *False* **by** *auto*

**qed**

**lemma** *alw-stream-all2-mono*:

**assumes** *stream-all2 P xs ys alw Q xs  $\wedge$  xs ys. stream-all2 P xs ys  $\implies$  Q xs  $\implies$  R ys*

**shows** *alw R ys*

**using** *assms stream.rel-sel* **by** (*coinduction arbitrary: xs ys*) (*blast*)

**lemma** *alw-ev-HLD-cycle*:

**assumes** *stream-all2 ( $\in$ ) xs (cycle as) a  $\in$  set as*

**shows** *infs ( $\lambda x. x \in a$ ) xs*

**using** *assms(1)*

**proof** (*coinduct rule: infs-coinduct-shift*)

**case** (*infs xs*)

**have**  $1: as \neq []$  **using** *assms(2)* **by** *auto*

**have 2:**  
*list-all2* ( $\in$ ) (*stake* (*length as*) *xs*) (*stake* (*length as*) (*cycle as*))  
*stream-all2* ( $\in$ ) (*sdrop* (*length as*) *xs*) (*sdrop* (*length as*) (*cycle as*))  
**using** *infs stream-rel-shift stake-sdrop length-stake* **by** *metis+*  
**have 3:** *stake* (*length as*) (*cycle as*) = *as* **using 1** **by** *simp*  
**have 4:** *sdrop* (*length as*) (*cycle as*) = *cycle as* **using** *sdrop-cycle-eq 1* **by**  
*this*  
**have 5:** *set* (*stake* (*length as*) *xs*)  $\cap$  *a*  $\neq$   $\{\}$   
**using** *assms(2) 2(1) unfolding list.in-rel 3*  
**by** (*auto*) (*metis IntI empty-iff mem-Collect-eq set-zip-leftD split-conv*  
*subsetCE zip-map-fst-snd*)  
**show** *?case* **using 2 5 unfolding 4**  
**by** *force*  
**qed**

**lemma** *alw-ev-mono:*  
**assumes** *alw* (*ev*  $\varphi$ ) *xs* **and**  $\bigwedge$  *xs*.  $\varphi$  *xs*  $\implies$   $\psi$  *xs*  
**shows** *alw* (*ev*  $\psi$ ) *xs*  
**by** (*rule alw-mp[OF assms(1)]*) (*auto intro: ev-mono assms(2) simp:*  
*alw-iff-sdrop*)

**lemma** *alw-ev-lockstep:*  
**assumes**  
*alw* (*ev* (*holds P*)) *xs stream-all2 Q xs as*  
 $\bigwedge$  *x a*.  $P$  *x*  $\implies$   $Q$  *x a*  $\implies$   $R$  *a*  
**shows**  
*alw* (*ev* (*holds R*)) *as*  
**using** *assms(1,2)*  
**apply** (*coinduction arbitrary: xs as rule: alw.coinduct*)  
**apply** *auto*  
**subgoal**  
**by** (*metis alw.cases assms(3) ev-holds-sset stream-all2-sset1*)  
**subgoal**  
**by** (*meson alw.cases stream.rel-sel*)  
**done**

### 1.2.1 sfilter, wait, nxt

Useful?

**lemma** *nxt-holds-iff-snth:* (*nxt*  $\widetilde{\sim}$  *i*) (*holds P*) *xs*  $\longleftrightarrow$   $P$  (*xs* !! *i*)  
**by** (*induction i arbitrary: xs; simp add: holds.simps*)

Useful?

**lemma** *wait-LEAST:*

*wait (holds P) xs = (LEAST n. P (xs !! n))* **unfolding** *wait-def next-holds-iff-snth*  
 ..

**lemma** *sfilter-SCons-decomp*:

**assumes** *sfilter P xs = x ## zs ev (holds P) xs*

**shows**  $\exists ys' zs'. xs = ys' @- x ## zs' \wedge \text{list-all } (Not \circ P) ys' \wedge P x \wedge sfilter P zs' = zs$

**proof** –

**note** *[simp] = holds.simps*

**from** *ev-imp-shift[OF assms(2)]* **obtain** *as bs* **where** *xs = as @- bs* **holds** *P bs*

**by** *auto*

**then have** *P (shd bs)* **by** *auto*

**with**  $\langle xs = - \rangle$  **have**  $\exists n. P (xs !! n)$  **using** *assms(2)* *sdrop-wait* **by** *fastforce*

**from** *sdrop-while-sdrop-LEAST[OF this]* **have**  $*$ :

*sdrop-while (Not o P) xs = sdrop (LEAST n. P (xs !! n)) xs .*

**let**  $?xs = sdrop-while (Not \circ P) xs$  **let**  $?n = LEAST n. P (xs !! n)$

**from** *assms(1)* **have** *x = shd ?xs zs = sfilter P (stl ?xs)*

**by** *(subst (asm) sfilter.ctr; simp)+*

**have** *xs = stake ?n xs @- sdrop ?n xs* **by** *simp*

**moreover have** *P x* **using** *assms(1)* **unfolding** *sfilter-eq[OF assms(2)]*

..

**moreover from**  $\langle \exists n. P - \rangle$  **have** *list-all (Not o P) (stake ?n xs)* **by** *(rule list-all-stake-least)*

**ultimately show** *?thesis*

**using**  $\langle x = - \rangle \langle zs = - \rangle *[\text{symmetric}]$  **by** *(inst-existentials stake ?n xs stl ?xs) auto*

**qed**

**lemma** *sfilter-SCons-decomp'*:

**assumes** *sfilter P xs = x ## zs ev (holds P) xs*

**shows**

*list-all (Not o P) (stake (wait (holds P) xs) xs) (is ?G1)*

*P x*

$\exists zs'. xs = stake (wait (holds P) xs) xs @- x ## zs' \wedge sfilter P zs' = zs$  **(is ?G2)**

**proof** –

**note** *[simp] = holds.simps*

**from** *ev-imp-shift[OF assms(2)]* **obtain** *as bs* **where** *xs = as @- bs* **holds** *P bs*

**by** *auto*

**then have** *P (shd bs)* **by** *auto*

**with**  $\langle xs = - \rangle$  **have**  $\exists n. P (xs !! n)$  **using** *assms(2)* *sdrop-wait* **by**

*fastforce* **thm** *sdrop-wait*  
**from** *sdrop-while-sdrop-LEAST[OF this]* **have** \*:  
*sdrop-while* (Not o P) xs = *sdrop* (LEAST n. P (xs !! n)) xs .  
**let** ?xs = *sdrop-while* (Not o P) xs **let** ?n = *wait* (holds P) xs  
**from** *assms(1)* **have** x = *shd* ?xs zs = *sfilter* P (stl ?xs)  
**by** (subst (asm) *sfilter.ctr*; *simp*)+  
**have** xs = *stake* ?n xs @- *sdrop* ?n xs **by** *simp*  
**moreover** **show** P x **using** *assms(1)* **unfolding** *sfilter-eq[OF assms(2)]*  
**..**  
**moreover** **from**  $\langle \exists n. P \rightarrow \rangle$  **show** *list-all* (Not o P) (*stake* ?n xs)  
**by** (*auto intro: list-all-stake-least simp: wait-LEAST*)  
**ultimately** **show** ?G2  
**using**  $\langle x = \rightarrow \rangle \langle zs = \rightarrow \rangle$  \**[symmetric]* **by** (*inst-existentials stl ?xs*) (*auto simp: wait-LEAST*)  
**qed**

**lemma** *sfilter-shift-decomp*:  
**assumes** *sfilter* P xs = ys @- zs *alw* (ev (holds P)) xs  
**shows**  $\exists$  ys' zs'. xs = ys' @- zs'  $\wedge$  *filter* P ys' = ys  $\wedge$  *sfilter* P zs' = zs  
**using** *assms(1,2)*  
**proof** (*induction ys arbitrary: xs*)  
**case** Nil  
**then** **show** ?case **by** (*inst-existentials [] :: 'a list xs; simp*)  
**next**  
**case** (Cons y ys)  
**from** *alw-ev-imp-ev-alw[OF alw (ev -) xs]* **have** ev (holds P) xs  
**by** (*auto elim: ev-mono*)  
**with** *Cons.prem(1)* *sfilter-SCons-decomp[of P xs y ys @- zs]* **obtain** ys'  
zs' **where** *decomp*:  
xs = ys' @- y ## zs' *list-all* (Not o P) ys' P y *sfilter* P zs' = ys @- zs  
**by** *clarsimp*  
**then** **have** *sfilter* P zs' = ys @- zs **by** *auto*  
**from**  $\langle alw (ev -) xs \rangle \langle xs = \rightarrow \rangle$  **have** *alw* (ev (holds P)) zs'  
**by** (*metis ev.intros(2) ev-shift not-alw-iff stream.sel(2)*)  
**from** *Cons.IH[OF sfilter P zs' = \rightarrow this]* **obtain** zs1 zs2 **where**  
zs' = zs1 @- zs2 *filter* P zs1 = ys *sfilter* P zs2 = zs  
**by** *clarsimp*  
**with** *decomp* **show** ?case  
**by** (*inst-existentials ys' @ y ## zs1 zs2; simp add: list.pred-set*)  
**qed**

**lemma** *finite-sset-sfilter-decomp*:  
**assumes** *finite* (sset (*sfilter* P xs)) *alw* (ev (holds P)) xs  
**obtains** x ws ys zs **where** xs = ws @- x ## ys @- x ## zs P x

**proof** *atomize-elim*  
**let**  $?xs = sfilter\ P\ xs$   
**have**  $1: \neg\ sdistinct\ (sfilter\ P\ xs)$  **using** *sdistinct-infinite-sset\ assms(1)*  
**by** *auto*  
**from** *not-sdistinct-decomp[OF\ 1]* **obtain**  $ws\ ys\ x\ zs$  **where** *guessed1*:  
 $sfilter\ P\ xs = ws\ @-\ x\ ##\ ys\ @-\ x\ ##\ zs$ .  
**from** *sfilter-shift-decomp[OF\ this\ assms(2)]* **obtain**  $ys'\ zs'$  **where** *guessed2*:  
 $xs = ys'\ @-\ zs'$   
 $sfilter\ P\ zs' = x\ ##\ ys\ @-\ x\ ##\ zs$   
 $ws = filter\ P\ ys'$   
**by** *clarsimp*  
**then have**  $ev\ (holds\ P)\ zs'$  **using** *alw-shift\ assms(2)* **by** *blast*  
**from** *sfilter-SCons-decomp[OF\ guessed2(2)\ this]* **obtain**  $zs1\ zs2$  **where**  
*guessed3*:  
 $zs' = zs1\ @-\ x\ ##\ zs2$   
 $list-all\ (Not\ o\ P)\ zs1$   
 $P\ x$   
 $sfilter\ P\ zs2 = ys\ @-\ x\ ##\ zs$   
**by** *clarsimp*  
**have**  $alw\ (ev\ (holds\ P))\ zs2$   
**by** *(metis\ alw-ev-stl\ alw-shift\ assms(2)\ guessed2(1)\ guessed3(1)\ stream.sel(2))*  
**from** *sfilter-shift-decomp[OF\ guessed3(4)\ this]* **obtain**  $zs3\ zs4$  **where**  
*guessed4*:  
 $zs2 = zs3\ @-\ zs4$   
 $sfilter\ P\ zs4 = x\ ##\ zs$   
 $ys = filter\ P\ zs3$   
**by** *clarsimp*  
**have**  $ev\ (holds\ P)\ zs4$   
**using**  $\langle alw\ (ev\ (holds\ P))\ zs2 \rangle$  *alw-shift\ guessed4(1)* **by** *blast*  
**from** *sfilter-SCons-decomp[OF\ guessed4(2)\ this]* **obtain**  $zs5\ zs6$  **where**  
 $zs4 = zs5\ @-\ x\ ##\ zs6$   
 $list-all\ (Not\ o\ P)\ zs5$   
 $P\ x$   
 $zs = sfilter\ P\ zs6$   
**by** *clarsimp*  
**with** *guessed1\ guessed2\ guessed3\ guessed4* **show**  $\exists\ ws\ x\ ys\ zs.\ xs = ws\ @-\$   
 $x\ ##\ ys\ @-\ x\ ##\ zs \wedge\ P\ x$   
**by** *(inst-existentials\ ys'\ @\ zs1\ x\ zs3\ @\ zs5\ zs6; simp)*  
**qed**

Useful?

**lemma** *sfilter-shd-LEAST*:

$shd\ (sfilter\ P\ xs) = xs\ !!\ (LEAST\ n.\ P\ (xs\ !!\ n))$  **if**  $ev\ (holds\ P)\ xs$

**proof** –

**note**  $[simp] = holds.simps$   
**from**  $sdrop-wait[OF \langle ev - xs \rangle]$  **have**  $\exists n. P (xs !! n)$  **by** *auto*  
**from**  $sdrop-while-sdrop-LEAST[OF this]$  **show** *?thesis* **by** *simp*  
**qed**

**lemma** *alw-nxt-holds-cong*:

$(nxt \overset{\sim}{\sim} n) (holds (\lambda x. P x \wedge Q x)) xs = (nxt \overset{\sim}{\sim} n) (holds Q) xs$  **if** *alw*  
 $(holds P) xs$

**using** *that* **unfolding** *nxt-holds-iff-snth alw-iff-sdrop* **by**  $(simp\ add:\ holds.simps)$

**lemma** *alw-wait-holds-cong*:

$wait (holds (\lambda x. P x \wedge Q x)) xs = wait (holds Q) xs$  **if** *alw*  $(holds P) xs$

**unfolding** *wait-def alw-nxt-holds-cong[OF that]* **..**

**lemma** *alw-sfilter*:

$sfilter (\lambda x. P x \wedge Q x) xs = sfilter Q xs$  **if** *alw*  $(holds P) xs$  *alw*  $(ev (holds Q)) xs$

**using** *that*

**proof**  $(coinduction\ arbitrary:\ xs)$

**case** *prems: stream-eq*

**note**  $[simp] = holds.simps$

**from** *prems(3,4)* **have** *ev-one*:  $ev (holds (\lambda x. P x \wedge Q x)) xs$

**by**  $(subst\ ev-cong[of\ -\ -\ holds\ Q]) (assumption\ |\ auto)+$

**from** *prems* **have**  $a = shd (sfilter (\lambda x. P x \wedge Q x) xs)$   $b = shd (sfilter Q xs)$

**by**  $(metis\ stream.sel(1))+$

**with** *prems(3,4)* **have**

$a = xs !! (LEAST\ n. P (xs !! n) \wedge Q (xs !! n))$   $b = xs !! (LEAST\ n. Q (xs !! n))$

**using** *ev-one* **by**  $(auto\ 4\ 3\ dest:\ sfilter-shd-LEAST)$

**with** *alw-wait-holds-cong[unfolded wait-LEAST, OF \langle alw (holds P) xs \rangle]*

**have**  $a = b$  **by** *simp*

**from** *sfilter-SCons-decomp'[OF prems(1)[symmetric], OF ev-one]* **obtain** *u2* **where** *guessed-a*:

$list-all (Not \circ (\lambda x. P x \wedge Q x)) (stake (wait (holds (\lambda x. P x \wedge Q x)) xs) xs)$

$xs = stake (wait (holds (\lambda x. P x \wedge Q x)) xs) xs @- a \#\# u2$

$u = sfilter (\lambda x. P x \wedge Q x) u2$

**by** *clarsimp*

**have**  $ev (holds Q) xs$  **using** *prems(4)* **by** *blast*

**from** *sfilter-SCons-decomp'[OF prems(2)[symmetric], OF this]* **obtain** *v2*

**where**

$list-all (Not \circ Q) (stake (wait (holds Q) xs) xs)$

$xs = stake (wait (holds Q) xs) xs @- b \#\# v2$

```

    v = sfilter Q v2
  by clarsimp
with guessed-a ⟨a = b⟩ show ?case
  apply (intro conjI exI)
    apply assumption+
    apply (simp add: alw-wait-holds-cong[OF prems(3)], metis shift-left-inj
stream.inject)
  by (metis alw.cases alw-shift prems(3,4) stream.sel(2))+
qed

```

**lemma** *alw-ev-holds-mp*:

```

alw (holds P) xs  $\implies$  ev (holds Q) xs  $\implies$  ev (holds ( $\lambda x. P x \wedge Q x$ )) xs
by (subst ev-cong, assumption) (auto simp: holds.simps)

```

**lemma** *alw-ev-conjI*:

```

alw (ev (holds ( $\lambda x. P x \wedge Q x$ ))) xs if alw (holds P) xs alw (ev (holds
Q)) xs
using that(2,1) by - (erule alw-mp, coinduction arbitrary: xs, auto intro:
alw-ev-holds-mp)

```

### 1.2.2 Useful?

**lemma** *alw-holds-pred-stream-iff*:

```

alw (holds P) xs  $\longleftrightarrow$  pred-stream P xs
by (simp add: alw-iff-sdrop stream-pred-snth holds.simps)

```

**lemma** *alw-holds-sset*:

```

alw (holds P) xs = ( $\forall x \in \text{sset } xs. P x$ )
by (simp add: alw-holds-pred-stream-iff stream.pred-set)

```

**lemma** *pred-stream-sfilter*:

```

assumes alw-ev: alw (ev (holds P)) xs
shows pred-stream P (sfilter P xs)
using alw-ev
proof (coinduction arbitrary: xs)
  case (stream-pred xs)
  then have ev (holds P) xs by auto
  have sfilter P xs = shd (sfilter P xs) ## stl (sfilter P xs)
    by (cases sfilter P xs) auto
  from sfilter-SCons-decomp[OF this ⟨ev (holds P) xs⟩] obtain ys' zs'
where
  xs = ys' @- shd (sdrop-while (Not  $\circ$  P) xs) ## zs'
  list-all (Not  $\circ$  P) ys'
  P (shd (sdrop-while (Not  $\circ$  P) xs))

```

```

  sfilter P zs' =
    sfilter P (stl (sdrop-while (Not ∘ P) xs))
  by clarsimp
then show ?case
  apply (inst-existentials zs')
  apply (metis sfilter.simps(1) stream.sel(1) stream-pred(1))
  apply (metis sconseq sfilter.simps(2) stream-pred(1))
  apply (metis alw-ev-stl alw-shift stream.sel(2) stream-pred(2))
  done
qed

```

**lemma** *alw-ev-sfilter-mono*:

```

assumes alw-ev: alw (ev (holds P)) xs
  and mono:  $\bigwedge x. P x \implies Q x$ 
shows pred-stream Q (sfilter P xs)
using stream.pred-mono[of P Q] assms pred-stream-sfilter by blast

```

**lemma** *sset-sfilter*:

```

  sset (sfilter P xs)  $\subseteq$  sset xs if alw (ev (holds P)) xs
proof –
  have alw (holds ( $\lambda x. x \in$  sset xs)) xs by (simp add: alw-iff-sdrop holds.simps)
  with  $\langle$ alw (ev -)  $\rightarrow$  alw-sfilter[OF this  $\langle$ alw (ev -)  $\rightarrow$ , symmetric $\rangle$ 
  have pred-stream ( $\lambda x. x \in$  sset xs) (sfilter P xs)
  by (simp) (rule alw-ev-sfilter-mono; auto intro: alw-ev-conjI)
  then have  $\forall x \in$  sset (sfilter P xs).  $x \in$  sset xs unfolding stream.pred-set
by this
  then show ?thesis by blast
qed

```

**lemma** *stream-all2-weaken*:

```

  stream-all2 Q xs ys if stream-all2 P xs ys  $\bigwedge x y. P x y \implies Q x y$ 
using that by (coinduction arbitrary: xs ys) auto

```

**lemma** *stream-all2-SCons1*:

```

  stream-all2 P (x ## xs) ys = ( $\exists z zs. ys = z ## zs \wedge P x z \wedge$  stream-all2
  P xs zs)
  by (subst (3) stream.collapse[symmetric], simp del: stream.collapse, force)

```

**lemma** *stream-all2-SCons2*:

```

  stream-all2 P xs (y ## ys) = ( $\exists z zs. xs = z ## zs \wedge P z y \wedge$  stream-all2
  P zs ys)
  by (subst stream.collapse[symmetric], simp del: stream.collapse, force)

```

**lemma** *stream-all2-combine*:

*stream-all2*  $R$   $xs$   $zs$  **if**  
*stream-all2*  $P$   $xs$   $ys$  *stream-all2*  $Q$   $ys$   $zs$   $\wedge$   $x$   $y$   $z$ .  $P$   $x$   $y$   $\wedge$   $Q$   $y$   $z$   $\implies$   $R$   $x$   $z$   
**using** *that*(1,2)  
**by** (*coinduction arbitrary*:  $xs$   $ys$   $zs$ )  
(auto *intro*: *that*(3) *simp*: *stream-all2-SCons1 stream-all2-SCons2*)

**lemma** *stream-all2-shift1*:  
*stream-all2*  $P$  ( $xs1$  @-  $xs2$ )  $ys$  =  
( $\exists$   $ys1$   $ys2$ .  $ys$  =  $ys1$  @-  $ys2$   $\wedge$  *list-all2*  $P$   $xs1$   $ys1$   $\wedge$  *stream-all2*  $P$   $xs2$   $ys2$ )  
**apply** (*induction xs1 arbitrary*:  $ys$ )  
**apply** (*simp*; *fail*)  
**apply** (*simp add*: *stream-all2-SCons1 list-all2-Cons1*)  
**apply** *safe*  
**subgoal for**  $a$   $xs1$   $ys$   $z$   $zs$   $ys1$   $ys2$   
**by** (*inst-existentials*  $z$   $\#$   $ys1$   $ys2$ ; *simp*)  
**subgoal for**  $a$   $xs1$   $ys$   $ys1$   $ys2$   $z$   $zs$   
**by** (*inst-existentials*  $z$   $zs$  @-  $ys2$   $zs$   $ys2$ ; *simp*)  
**done**

**lemma** *stream-all2-shift2*:  
*stream-all2*  $P$   $ys$  ( $xs1$  @-  $xs2$ ) =  
( $\exists$   $ys1$   $ys2$ .  $ys$  =  $ys1$  @-  $ys2$   $\wedge$  *list-all2*  $P$   $ys1$   $xs1$   $\wedge$  *stream-all2*  $P$   $ys2$   $xs2$ )  
**by** (*meson list.rel-flip stream.rel-flip stream-all2-shift1*)

**lemma** *stream-all2-bisim*:  
**assumes** *stream-all2* ( $\in$ )  $xs$   $as$  *stream-all2* ( $\in$ )  $ys$   $as$  *sset*  $as$   $\subseteq$   $S$   
**shows** *stream-all2* ( $\lambda$   $x$   $y$ .  $\exists$   $a$ .  $x$   $\in$   $a$   $\wedge$   $y$   $\in$   $a$   $\wedge$   $a$   $\in$   $S$ )  $xs$   $ys$   
**using** *assms*  
**apply** (*coinduction arbitrary*:  $as$   $xs$   $ys$ )  
**subgoal for**  $a$   $u$   $b$   $v$   $as$   $xs$   $ys$   
**apply** (*rule conjI*)  
**apply** (*inst-existentials shd*  $as$ , *auto simp*: *stream-all2-SCons1*; *fail*)  
**apply** (*inst-existentials stl*  $as$ , *auto* 4 3 *simp*: *stream-all2-SCons1*; *fail*)  
**done**  
**done**

**end**

### 1.3 Mixed Material

**theory** *TA-Misc*  
**imports** *Main HOL.Real*

**begin**

### 1.3.1 Reals

**Properties of fractions lemma** *frac-add-le-preservation:*

**fixes**  $a\ d :: \text{real}$  **and**  $b :: \text{nat}$

**assumes**  $a < b\ d < 1 - \text{frac } a$

**shows**  $a + d < b$

**proof** –

**from** *assms* **have**  $a + d < a + 1 - \text{frac } a$  **by** *auto*

**also have**  $\dots = (a - \text{frac } a) + 1$  **by** *auto*

**also have**  $\dots = \text{floor } a + 1$  **unfolding** *frac-def* **by** *auto*

**also have**  $\dots \leq b$  **using**  $\langle a < b \rangle$

**by** (*metis floor-less-iff int-less-real-le of-int-1 of-int-add of-int-of-nat-eq*)

**finally show**  $a + d < b$  .

**qed**

**lemma** *lt-1-contr:*

$(a :: \text{int}) < b \implies b < a + 1 \implies \text{False}$  **by** *auto*

**lemma** *int-intv-frac-gt0:*

$(a :: \text{int}) < b \implies b < a + 1 \implies \text{frac } b > 0$  **by** *auto*

**lemma** *floor-frac-add-preservation:*

**fixes**  $a\ d :: \text{real}$

**assumes**  $0 < d\ d < 1 - \text{frac } a$

**shows**  $\text{floor } a = \text{floor } (a + d)$

**proof** –

**have**  $\text{frac } a \geq 0$  **by** *auto*

**with** *assms*(2) **have**  $d < 1$  **by** *linarith*

**from** *assms* **have**  $a + d < a + 1 - \text{frac } a$  **by** *auto*

**also have**  $\dots = (a - \text{frac } a) + 1$  **by** *auto*

**also have**  $\dots = (\text{floor } a) + 1$  **unfolding** *frac-def* **by** *auto*

**finally have**  $*$ :  $a + d < \text{floor } a + 1$  .

**have**  $\text{floor } (a + d) \geq \text{floor } a$  **using**  $\langle d > 0 \rangle$  **by** *linarith*

**moreover from**  $*$  **have**  $\text{floor } (a + d) < \text{floor } a + 1$  **by** *linarith*

**ultimately show**  $\text{floor } a = \text{floor } (a + d)$  **by** *auto*

**qed**

**lemma** *frac-distr:*

**fixes**  $a\ d :: \text{real}$

**assumes**  $0 < d\ d < 1 - \text{frac } a$

**shows**  $\text{frac } (a + d) > 0\ \text{frac } a + d = \text{frac } (a + d)$

**proof** –

**have**  $\text{frac } a \geq 0$  **by** *auto*  
**with** *assms(2)* **have**  $d < 1$  **by** *linarith*  
**from** *assms* **have**  $a + d < a + 1 - \text{frac } a$  **by** *auto*  
**also have**  $\dots = (a - \text{frac } a) + 1$  **by** *auto*  
**also have**  $\dots = (\text{floor } a) + 1$  **unfolding** *frac-def* **by** *auto*  
**finally have**  $*$ :  $a + d < \text{floor } a + 1$  .  
**have**  $**$ :  $\text{floor } a < a + d$  **using** *assms(1)* **by** *linarith*  
**have**  $\text{frac } (a + d) \neq 0$   
**proof** (*rule ccontr, auto, goal-cases*)  
  **case** 1  
    **then obtain**  $b :: \text{int}$  **where**  $b = a + d$  **by** (*metis Ints-cases*)  
    **with**  $*$   $**$  **have**  $b < \text{floor } a + 1$   $\text{floor } a < b$  **by** *auto*  
    **with** *lt-lt-1-ccontr* **show** *?case* **by** *blast*  
  **qed**  
**then show**  $\text{frac } (a + d) > 0$  **by** *auto*  
**from** *floor-frac-add-preservation assms* **have**  $\text{floor } a = \text{floor } (a + d)$  **by**  
*auto*  
**then show**  $\text{frac } a + d = \text{frac } (a + d)$  **unfolding** *frac-def* **by** *force*  
**qed**

**lemma** *frac-add-leD*:

**fixes**  $a d :: \text{real}$   
**assumes**  $0 < d$   $d < 1 - \text{frac } a$   $d < 1 - \text{frac } b$   $\text{frac } (a + d) \leq \text{frac } (b + d)$   
**shows**  $\text{frac } a \leq \text{frac } b$   
**proof** –  
**from** *floor-frac-add-preservation assms* **have**  
   $\text{floor } a = \text{floor } (a + d)$   $\text{floor } b = \text{floor } (b + d)$   
**by** *auto*  
**with** *assms(4)* **show**  $\text{frac } a \leq \text{frac } b$  **unfolding** *frac-def* **by** *auto*  
**qed**

**lemma** *floor-frac-add-preservation'*:

**fixes**  $a d :: \text{real}$   
**assumes**  $0 \leq d$   $d < 1 - \text{frac } a$   
**shows**  $\text{floor } a = \text{floor } (a + d)$   
**using** *assms floor-frac-add-preservation* **by** (*cases d = 0*) *auto*

**lemma** *frac-add-leIFF*:

**fixes**  $a d :: \text{real}$   
**assumes**  $0 \leq d$   $d < 1 - \text{frac } a$   $d < 1 - \text{frac } b$   
**shows**  $\text{frac } a \leq \text{frac } b \iff \text{frac } (a + d) \leq \text{frac } (b + d)$   
**proof** –  
**from** *floor-frac-add-preservation'* *assms* **have**

$\text{floor } a = \text{floor } (a + d) \text{ floor } b = \text{floor } (b + d)$   
 by *auto*  
 then show *?thesis unfolding frac-def by auto*  
 qed

**lemma** *nat-intv-frac-gt0*:  
 fixes  $c :: \text{nat}$  fixes  $x :: \text{real}$   
 assumes  $c < x$   $x < \text{real } (c + 1)$   
 shows  $\text{frac } x > 0$   
**proof** (*rule ccontr, auto, goal-cases*)  
 case 1  
 then obtain  $d :: \text{int}$  where  $x = d$  by (*metis Ints-cases*)  
 with *assms* have  $c < d$   $d < \text{int } c + 1$  by *auto*  
 with *int-intv-frac-gt0[OF this] 1 d* show *False* by *auto*  
 qed

**lemma** *nat-intv-frac-decomp*:  
 fixes  $c :: \text{nat}$  and  $d :: \text{real}$   
 assumes  $c < d$   $d < c + 1$   
 shows  $d = c + \text{frac } d$   
**proof** –  
 from *assms* have  $\text{int } c = \lfloor d \rfloor$  by *linarith*  
 thus *?thesis* by (*simp add: frac-def*)  
 qed

**lemma** *nat-intv-not-int*:  
 fixes  $c :: \text{nat}$   
 assumes  $\text{real } c < d$   $d < c + 1$   
 shows  $d \notin \mathbb{Z}$   
**proof** (*standard, goal-cases*)  
 case 1  
 then obtain  $k :: \text{int}$  where  $d = k$  using *Ints-cases* by *auto*  
 then have  $\text{frac } d = 0$  by *auto*  
 moreover from *nat-intv-frac-decomp[OF assms]* have  $*$ :  $d = c + \text{frac } d$   
 by *auto*  
 ultimately have  $d = c$  by *linarith*  
 with *assms* show *?case* by *auto*  
 qed

**lemma** *frac-nat-add-id*:  $\text{frac } ((n :: \text{nat}) + (r :: \text{real})) = \text{frac } r$  — Found by  
 sledgehammer  
**proof** –  
 have  $\bigwedge r. \text{frac } (r :: \text{real}) < 1$   
 by (*meson frac-lt-1*)

**then show** *?thesis*  
**by** (*simp add: floor-add frac-def*)  
**qed**

**lemma** *floor-nat-add-id*:  $0 \leq (r :: \text{real}) \implies r < 1 \implies \text{floor} (\text{real} (n::\text{nat}) + r) = n$  **by** *linarith*

**lemma** *int-intv-frac-gt-0'*:

$(a :: \text{real}) \in \mathbb{Z} \implies (b :: \text{real}) \in \mathbb{Z} \implies a \leq b \implies a \neq b \implies a \leq b - 1$

**proof** (*goal-cases*)

**case** 1

**then have**  $a < b$  **by** *auto*

**from** 1(1,2) **obtain**  $k\ l :: \text{int}$  **where**  $a = \text{real-of-int } k\ b = \text{real-of-int } l$

**by** (*metis Ints-cases*)

**with**  $\langle a < b \rangle$  **show** *?case* **by** *auto*

**qed**

**lemma** *int-lt-Suc-le*:

$(a :: \text{real}) \in \mathbb{Z} \implies (b :: \text{real}) \in \mathbb{Z} \implies a < b + 1 \implies a \leq b$

**proof** (*goal-cases*)

**case** 1

**from** 1(1,2) **obtain**  $k\ l :: \text{int}$  **where**  $a = \text{real-of-int } k\ b = \text{real-of-int } l$

**by** (*metis Ints-cases*)

**with**  $\langle a < b + 1 \rangle$  **show** *?case* **by** *auto*

**qed**

**lemma** *int-lt-neq-Suc-lt*:

$(a :: \text{real}) \in \mathbb{Z} \implies (b :: \text{real}) \in \mathbb{Z} \implies a < b \implies a + 1 \neq b \implies a + 1 < b$

**proof** (*goal-cases*)

**case** 1

**from** 1(1,2) **obtain**  $k\ l :: \text{int}$  **where**  $a = \text{real-of-int } k\ b = \text{real-of-int } l$

**by** (*metis Ints-cases*)

**with** 1 **show** *?case* **by** *auto*

**qed**

**lemma** *int-lt-neq-prev-lt*:

$(a :: \text{real}) \in \mathbb{Z} \implies (b :: \text{real}) \in \mathbb{Z} \implies a - 1 < b \implies a \neq b \implies a < b$

**proof** (*goal-cases*)

**case** 1

**from** 1(1,2) **obtain**  $k\ l :: \text{int}$  **where**  $a = \text{real-of-int } k\ b = \text{real-of-int } l$

**by** (*metis Ints-cases*)

**with** 1 **show** *?case* **by** *auto*

**qed**

```

lemma ints-le-add-frac1:
  fixes a b x :: real
  assumes  $0 < x$   $x < 1$   $a \in \mathbb{Z}$   $b \in \mathbb{Z}$   $a + x \leq b$ 
  shows  $a \leq b$ 
using assms by auto

lemma ints-le-add-frac2:
  fixes a b x :: real
  assumes  $0 \leq x$   $x < 1$   $a \in \mathbb{Z}$   $b \in \mathbb{Z}$   $b \leq a + x$ 
  shows  $b \leq a$ 
using assms
by (metis add.commute add-le-cancel-left add-mono-thms-linordered-semiring(1)
int-lt-Suc-le leD le-less-linear)

```

### 1.3.2 Ordering Fractions

```

lemma distinct-twice-contradiction:
   $xs ! i = x \implies xs ! j = x \implies i < j \implies j < \text{length } xs \implies \neg \text{distinct } xs$ 
proof (rule ccontr, simp, induction xs arbitrary: i j)
  case Nil thus ?case by auto
next
  case (Cons y xs)
  show ?case
  proof (cases i = 0)
    case True
    with Cons have  $y = x$  by auto
    moreover from True Cons have  $x \in \text{set } xs$  by auto
    ultimately show False using Cons(6) by auto
  next
  case False
  with Cons have
     $xs ! (i - 1) = x$   $xs ! (j - 1) = x$   $i - 1 < j - 1$   $j - 1 < \text{length } xs$ 
    distinct xs
    by auto
    from Cons.IH[OF this] show False .
  qed
qed

```

```

lemma distinct-nth-unique:
   $xs ! i = xs ! j \implies i < \text{length } xs \implies j < \text{length } xs \implies \text{distinct } xs \implies i = j$ 
apply (rule ccontr)
apply (cases i < j)

```

**apply** *auto*  
**apply** (*auto dest: distinct-twice-contradiction*)  
**using** *distinct-twice-contradiction* **by** *fastforce*

**lemma** (**in** *linorder*) *linorder-order-fun*:

**fixes**  $S :: 'a \text{ set}$

**assumes** *finite S*

**obtains**  $f :: 'a \Rightarrow \text{nat}$

**where**  $(\forall x \in S. \forall y \in S. f x \leq f y \iff x \leq y)$  **and**  $\text{range } f \subseteq \{0.. \text{card } S - 1\}$

**proof** –

**obtain**  $l$  **where** *l-def: l = sorted-list-of-set S* **by** *auto*

**with** *sorted-list-of-set(1)[OF assms]* **have**  $l: \text{set } l = S$  *sorted l distinct l*

**by** *auto*

**from**  $l(1,3)$   $\langle \text{finite } S \rangle$  **have**  $\text{len: length } l = \text{card } S$  **using** *distinct-card* **by** *force*

**let**  $?f = \lambda x. \text{if } x \notin S \text{ then } 0 \text{ else } \text{THE } i. i < \text{length } l \wedge l! i = x$

**{ fix**  $x y$  **assume**  $A: x \in S y \in S x < y$

**with**  $l(1)$  **obtain**  $i j$  **where**  $*: l! i = x l! j = y i < \text{length } l j < \text{length } l$

$l$

**by** (*meson in-set-conv-nth*)

**have**  $i < j$

**proof** (*rule ccontr, goal-cases*)

**case** 1

**with** *sorted-nth-mono[OF l(2)]*  $\langle i < \text{length } l \rangle$  **have**  $l! j \leq l! i$  **by** *auto*

**with**  $* A(3)$  **show** *False* **by** *auto*

**qed**

**moreover** **have**  $?f x = i$  **using**  $* l(3) A(1)$  **by** (*auto*) (*rule, auto intro: distinct-nth-unique*)

**moreover** **have**  $?f y = j$  **using**  $* l(3) A(2)$  **by** (*auto*) (*rule, auto intro: distinct-nth-unique*)

**ultimately** **have**  $?f x < ?f y$  **by** *auto*

**} moreover**

**{ fix**  $x y$  **assume**  $A: x \in S y \in S ?f x < ?f y$

**with**  $l(1)$  **obtain**  $i j$  **where**  $*: l! i = x l! j = y i < \text{length } l j < \text{length } l$

$l$

**by** (*meson in-set-conv-nth*)

**moreover** **have**  $?f x = i$  **using**  $* l(3) A(1)$  **by** (*auto*) (*rule, auto intro: distinct-nth-unique*)

**moreover** **have**  $?f y = j$  **using**  $* l(3) A(2)$  **by** (*auto*) (*rule, auto intro: distinct-nth-unique*)

**ultimately** **have**  $**: l! ?f x = x l! ?f y = y i < j$  **using**  $A(3)$  **by** *auto*

**have**  $x < y$

```

proof (rule ccontr, goal-cases)
  case 1
  then have  $y \leq x$  by simp
  moreover from sorted-nth-mono[OF l(2), of i j] **(3) * have  $x \leq y$ 
by auto
  ultimately show False using distinct-nth-unique[OF - *(3,4) l(3)]
*(1,2) **(3) by fastforce
  qed
}
ultimately have  $\forall x \in S. \forall y \in S. ?f x \leq ?f y \longleftrightarrow x \leq y$  by force
moreover have range ?f  $\subseteq \{0..card\ S - 1\}$ 
proof (auto, goal-cases)
  case (1 x)
  with l(1) obtain i where *:  $l ! i = x$   $i < length\ l$  by (meson
in-set-conv-nth)
  then have  $?f\ x = i$  using l(3) 1 by (auto) (rule, auto intro: dis-
tinct-nth-unique)
  with len show ?case using *(2) 1 by auto
  qed
ultimately show ?thesis ..
qed

```

```

locale enumerable =
  fixes T :: 'a set
  fixes less :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infix <<> 50)
  assumes finite: finite T
  assumes total:  $\forall x \in T. \forall y \in T. x \neq y \longrightarrow (x < y) \vee (y < x)$ 
  assumes trans:  $\forall x \in T. \forall y \in T. \forall z \in T. (x :: 'a) < y \longrightarrow y < z \longrightarrow$ 
 $x < z$ 
  assumes asymmetric:  $\forall x \in T. \forall y \in T. x < y \longrightarrow \neg (y < x)$ 
begin

```

```

lemma non-empty-set-has-least':
   $S \subseteq T \Longrightarrow S \neq \{\} \Longrightarrow \exists x \in S. \forall y \in S. x \neq y \longrightarrow \neg y < x$ 
proof (rule ccontr, induction card S arbitrary: S)
  case 0 then show ?case using finite by (auto simp: finite-subset)
next
  case (Suc n)
  then obtain x where x:  $x \in S$  by blast
  from finite Suc.premis(1) have finite: finite S by (auto simp: finite-subset)
  let ?S =  $S - \{x\}$ 
  show ?case
  proof (cases  $S = \{x\}$ )
    case True

```

```

with Suc.prems(3) show False by auto
next
case False
then have S:  $?S \neq \{\}$  using x by blast
show False
proof (cases  $\exists x \in ?S. \forall y \in ?S. x \neq y \longrightarrow \neg y < x$ )
  case False
    have  $n = \text{card } ?S$  using Suc.hyps finite by (simp add: x)
    from Suc.hyps(1)[OF this - S False] Suc.prems(1) show False by auto
  next
  case True
    then obtain x' where  $x': \forall y \in ?S. x' \neq y \longrightarrow \neg y < x'$   $x' \in ?S$   $x \neq x'$  by auto
    from total Suc.prems(1) x'(2) have  $\bigwedge y. y \in S \implies x' \neq y \implies \neg y < x' \implies x' < y$  by auto
    from total Suc.prems(1) x'(1,2) have  $*$ :  $\forall y \in ?S. x' \neq y \longrightarrow x' < y$  by auto
    from Suc.prems(3) x'(1,2) have  $**$ :  $x < x'$  by auto
    have  $\forall y \in ?S. x < y$ 
    proof
      fix y assume  $y: y \in S - \{x\}$ 
      show  $x < y$ 
      proof (cases  $y = x'$ )
        case True then show ?thesis using  $**$  by simp
      next
        case False
          with  $*$  y have  $x' < y$  by auto
          with trans Suc.prems(1)  $**$  y x'(2)  $x$   $**$  show ?thesis by auto
      qed
    qed
  with x Suc.prems(1,3) show False using asymmetric by blast
qed
qed
qed

```

**lemma** *non-empty-set-has-least''*:

$S \subseteq T \implies S \neq \{\} \implies \exists! x \in S. \forall y \in S. x \neq y \longrightarrow \neg y < x$

**proof** (*intro ex-ex1I, goal-cases*)

case 1

with *non-empty-set-has-least'*[*OF this*] show *?case* by *auto*

next

case (2 *x y*)

show *?case*

proof (*rule ccontr*)

**assume**  $x \neq y$   
**with**  $2$  *total* **have**  $x \prec y \vee y \prec x$  **by** *blast*  
**with**  $2(2-)$   $\langle x \neq y \rangle$  **show** *False* **by** *auto*  
**qed**  
**qed**

**abbreviation**  $\text{least } S \equiv \text{THE } t :: 'a. t \in S \wedge (\forall y \in S. t \neq y \longrightarrow \neg y \prec t)$

**lemma** *non-empty-set-has-least*:

$S \subseteq T \implies S \neq \{\} \implies \text{least } S \in S \wedge (\forall y \in S. \text{least } S \neq y \longrightarrow \neg y \prec \text{least } S)$

**proof** *goal-cases*

**case**  $1$

**note**  $A = \text{this}$

**show** *?thesis*

**proof** (*rule theI', goal-cases*)

**case**  $1$

**from** *non-empty-set-has-least''[OF A]* **show** *?case .*

**qed**

**qed**

**fun**  $f :: 'a \text{ set} \Rightarrow \text{nat} \Rightarrow 'a \text{ list}$

**where**

$f \ S \ 0 = [] \mid$

$f \ S \ (\text{Suc } n) = \text{least } S \ \# \ f \ (S - \{\text{least } S\}) \ n$

**inductive** *sorted* ::  $'a \text{ list} \Rightarrow \text{bool}$  **where**

*Nil* [*iff*]: *sorted*  $[]$

$\mid$  *Cons*:  $\forall y \in \text{set } xs. x \prec y \implies \text{sorted } xs \implies \text{sorted } (x \ \# \ xs)$

**lemma** *f-set*:

$S \subseteq T \implies n = \text{card } S \implies \text{set } (f \ S \ n) = S$

**proof** (*induction n arbitrary: S*)

**case**  $0$  **then show** *?case* **using** *finite* **by** (*auto simp: finite-subset*)

**next**

**case**  $(\text{Suc } n)$

**then have** *fin: finite S* **using** *finite* **by** (*auto simp: finite-subset*)

**with** *Suc.prem*s **have**  $S \neq \{\}$  **by** *auto*

**from** *non-empty-set-has-least[OF Suc.prem*s(1) *this]* **have** *least: least S*  
 $\in S$  **by** *blast*

**let**  $?S = S - \{\text{least } S\}$

**from** *fin* *least* *Suc.prem*s **have**  $?S \subseteq T \ n = \text{card } ?S$  **by** *auto*

**from** *Suc.IH[OF this]* **have**  $\text{set } (f \ ?S \ n) = ?S$  .

**with least show ?case by auto**  
**qed**

**lemma f-distinct:**

$S \subseteq T \implies n = \text{card } S \implies \text{distinct } (f \ S \ n)$

**proof** (induction n arbitrary: S)

**case 0 then show ?case using finite by (auto simp: finite-subset)**

**next**

**case (Suc n)**

**then have fin: finite S using finite by (auto simp: finite-subset)**

**with Suc.prem have  $S \neq \{\}$  by auto**

**from non-empty-set-has-least[OF Suc.prem(1) this] have least: least S**  
 $\in S$  **by blast**

**let ?S = S - {least S}**

**from fin least Suc.prem have  $?S \subseteq T$   $n = \text{card } ?S$  by auto**

**from Suc.IH[OF this] f-set[OF this] have distinct (f ?S n) set (f ?S n)**  
 $= ?S$  .

**then show ?case by simp**

**qed**

**lemma f-sorted:**

$S \subseteq T \implies n = \text{card } S \implies \text{sorted } (f \ S \ n)$

**proof** (induction n arbitrary: S)

**case 0 then show ?case by auto**

**next**

**case (Suc n)**

**then have fin: finite S using finite by (auto simp: finite-subset)**

**with Suc.prem have  $S \neq \{\}$  by auto**

**from non-empty-set-has-least[OF Suc.prem(1) this] have least:**

$\text{least } S \in S \ (\forall y \in S. \text{least } S \neq y \implies \neg y \prec \text{least } S)$

**by blast+**

**let ?S = S - {least S}**

**{ fix x assume x: x ∈ ?S**

**with least have  $\neg x \prec \text{least } S$  by auto**

**with total x Suc.prem(1) least(1) have least S < x by blast**

**} note le = this**

**from fin least Suc.prem have  $?S \subseteq T$   $n = \text{card } ?S$  by auto**

**from f-set[OF this] Suc.IH[OF this] have \*: set (f ?S n) = ?S sorted (f**  
 $?S \ n)$  .

**with le have  $\forall x \in \text{set } (f \ ?S \ n). \text{least } S \prec x$  by auto**

**with \*(2) show ?case by (auto intro: Cons)**

**qed**

**lemma sorted-nth-mono:**

```

  sorted xs  $\implies$  i < j  $\implies$  j < length xs  $\implies$  xs!i < xs!j
proof (induction xs arbitrary: i j)
  case Nil thus ?case by auto
next
  case (Cons x xs)
  show ?case
  proof (cases i = 0)
    case True
    with Cons.prem1 show ?thesis by (auto elim: sorted.cases)
  next
  case False
  from Cons.prem1 have sorted xs by (auto elim: sorted.cases)
  from Cons.IH[OF this] Cons.prem2 show ?thesis by auto
qed
qed

lemma order-fun:
  fixes S :: 'a set
  assumes S  $\subseteq$  T
  obtains f :: 'a  $\implies$  nat where  $\forall x \in S. \forall y \in S. f x < f y \iff x < y$ 
  and range f  $\subseteq$  {0.. $\text{card } S - 1$ }
proof -
  obtain l where l-def: l = f S (card S) by auto
  with f-set f-distinct f-sorted assms have l: set l = S sorted l distinct l by
  auto
  then have len: length l = card S using distinct-card by force
  let ?f =  $\lambda x. \text{if } x \notin S \text{ then } 0 \text{ else } \text{THE } i. i < \text{length } l \wedge l! i = x$ 
  { fix x y :: 'a assume A: x  $\in$  S y  $\in$  S x < y
  with l(1) obtain i j where *: l! i = x l! j = y i < length l j < length
  l
  by (meson in-set-conv-nth)
  have i  $\neq$  j
  proof (rule ccontr, goal-cases)
  case 1
  with A * have x < x by auto
  with asymmetric A assms show False by auto
  qed
  have i < j
  proof (rule ccontr, goal-cases)
  case 1
  with <i  $\neq$  j> sorted-nth-mono[OF l(2)] <i < length l> have l! j < l!
  i by auto
  with * A(3) A assms asymmetric show False by auto
  qed

```

**moreover have**  $?f x = i$  **using**  $* l(3)$   $A(1)$  **by**  $(auto)$  (*rule, auto intro: distinct-nth-unique*)  
**moreover have**  $?f y = j$  **using**  $* l(3)$   $A(2)$  **by**  $(auto)$  (*rule, auto intro: distinct-nth-unique*)  
**ultimately have**  $?f x < ?f y$  **by**  $auto$   
**}** **moreover**  
**{** **fix**  $x y$  **assume**  $A: x \in S y \in S$   $?f x < ?f y$   
**with**  $l(1)$  **obtain**  $i j$  **where**  $*: l ! i = x l ! j = y i < length l j < length l$   
**l**  
**by** (*meson in-set-conv-nth*)  
**moreover have**  $?f x = i$  **using**  $* l(3)$   $A(1)$  **by**  $(auto)$  (*rule, auto intro: distinct-nth-unique*)  
**moreover have**  $?f y = j$  **using**  $* l(3)$   $A(2)$  **by**  $(auto)$  (*rule, auto intro: distinct-nth-unique*)  
**ultimately have**  $**: l ! ?f x = x l ! ?f y = y i < j$  **using**  $A(3)$  **by**  $auto$   
**from** *sorted-nth-mono*[ $OF l(2)$ , *of i j*]  $**(3)$  **have**  $x < y$  **by**  $auto$   
**}**  
**ultimately have**  $\forall x \in S. \forall y \in S. ?f x < ?f y \longleftrightarrow x < y$  **by**  $force$   
**moreover have**  $range ?f \subseteq \{0..card S - 1\}$   
**proof** (*auto, goal-cases*)  
**case**  $(1 x)$   
**with**  $l(1)$  **obtain**  $i$  **where**  $*: l ! i = x i < length l$  **by** (*meson in-set-conv-nth*)  
**then have**  $?f x = i$  **using**  $l(3)$   $1$  **by**  $(auto)$  (*rule, auto intro: distinct-nth-unique*)  
**with**  $len$  **show**  $?case$  **using**  $*(2)$   $1$  **by**  $auto$   
**qed**  
**ultimately show**  $?thesis ..$   
**qed**  
**end**

**lemma** *finite-total-preorder-enumeration:*

**fixes**  $X :: 'a set$   
**fixes**  $r :: 'a rel$   
**assumes**  $fin: finite X$   
**assumes**  $tot: total-on X r$   
**assumes**  $refl: refl-on X r$   
**assumes**  $trans: trans r$   
**obtains**  $f :: 'a \Rightarrow nat$  **where**  $\forall x \in X. \forall y \in X. f x \leq f y \longleftrightarrow (x, y) \in r$   
**proof** –  
**let**  $?A = \lambda x. \{y \in X. (y, x) \in r \wedge (x, y) \in r\}$   
**have**  $ex: \forall x \in X. x \in ?A x$  **using**  $refl$  **unfolding**  $refl-on-def$  **by**  $auto$

```

let ?R = λ S. SOME y. y ∈ S
let ?T = {?A x | x. x ∈ X}
{ fix A assume A: A ∈ ?T
  then obtain x where x: x ∈ X ?A x = A by auto
  then have x ∈ A using refl unfolding refl-on-def by auto
  then have ?R A ∈ A by (auto intro: someI)
  with x(2) have (?R A, x) ∈ r (x, ?R A) ∈ r by auto
  with trans have (?R A, ?R A) ∈ r unfolding trans-def by blast
} note refl-lifted = this
{ fix A assume A: A ∈ ?T
  then obtain x where x: x ∈ X ?A x = A by auto
  then have x ∈ A using refl unfolding refl-on-def by auto
  then have ?R A ∈ A by (auto intro: someI)
} note R-in = this
{ fix A y z assume A: A ∈ ?T and y: y ∈ A and z: z ∈ A
  from A obtain x where x: x ∈ X ?A x = A by auto
  then have x ∈ A using refl unfolding refl-on-def by auto
  with x y have (x, y) ∈ r (y, x) ∈ r by auto
  moreover from x z have (x,z) ∈ r (z,x) ∈ r by auto
  ultimately have (y, z) ∈ r (z, y) ∈ r using trans unfolding trans-def
by blast+
} note A-dest' = this
{ fix A y assume A ∈ ?T and y ∈ A
  with A-dest'[OF - - R-in] have (?R A, y) ∈ r (y, ?R A) ∈ r by blast+
} note A-dest = this
{ fix A y z assume A: A ∈ ?T and y: y ∈ A and z: z ∈ X and r: (y,
z) ∈ r (z, y) ∈ r
  from A obtain x where x: x ∈ X ?A x = A by auto
  then have x ∈ A using refl unfolding refl-on-def by auto
  with x y have (x,y) ∈ r (y, x) ∈ r by auto
  with r have (x,z) ∈ r (z,x) ∈ r using trans unfolding trans-def by
blast+
  with x z have z ∈ A by auto
} note A-intro' = this
{ fix A y assume A: A ∈ ?T and y: y ∈ X and r: (?R A, y) ∈ r (y,
?R A) ∈ r
  with A-intro' R-in have y ∈ A by blast
} note A-intro = this
{ fix A B C
  assume r1: (?R A, ?R B) ∈ r and r2: (?R B, ?R C) ∈ r
  with trans have (?R A, ?R C) ∈ r unfolding trans-def by blast
} note trans-lifted[intro] = this
{ fix A B a b
  assume A: A ∈ ?T and B: B ∈ ?T

```

```

and  $a: a \in A$  and  $b: b \in B$ 
and  $r: (a, b) \in r (b, a) \in r$ 
with  $R$ -in have  $?R A \in A ?R B \in B$  by blast+
have  $A = B$ 
proof auto
  fix  $x$  assume  $x: x \in A$ 
  with  $A$  have  $x \in X$  by auto
  from  $A$ -intro'[ $OF B b$  this]  $A$ -dest'[ $OF A x a$ ]  $r$  trans[unfolded trans-def]
show  $x \in B$  by blast
next
  fix  $x$  assume  $x: x \in B$ 
  with  $B$  have  $x \in X$  by auto
  from  $A$ -intro'[ $OF A a$  this]  $A$ -dest'[ $OF B x b$ ]  $r$  trans[unfolded trans-def]
show  $x \in A$  by blast
qed
} note  $eq$ -lifted'' = this
{ fix  $A B C$ 
  assume  $A: A \in ?T$  and  $B: B \in ?T$  and  $r: (?R A, ?R B) \in r (?R B,$ 
   $?R A) \in r$ 
  with  $eq$ -lifted''  $R$ -in have  $A = B$  by blast
} note  $eq$ -lifted' = this
{ fix  $A B C$ 
  assume  $A: A \in ?T$  and  $B: B \in ?T$  and  $eq: ?R A = ?R B$ 
  from  $R$ -in[ $OF A$ ]  $A$  have  $?R A \in X$  by auto
  with refl have  $(?R A, ?R A) \in r$  unfolding refl-on-def by auto
  with  $eq$ -lifted'[ $OF A B$ ]  $eq$  have  $A = B$  by auto
} note  $eq$ -lifted = this
{ fix  $A B$ 
  assume  $A: A \in ?T$  and  $B: B \in ?T$  and  $neg: A \neq B$ 
  from  $neg$   $eq$ -lifted[ $OF A B$ ] have  $?R A \neq ?R B$  by metis
  moreover from  $A B$   $R$ -in have  $?R A \in X ?R B \in X$  by auto
  ultimately have  $(?R A, ?R B) \in r \vee (?R B, ?R A) \in r$  using tot
unfolding total-on-def by auto
} note  $total$ -lifted = this
{ fix  $x y$  assume  $x: x \in X$  and  $y: y \in X$ 
  from  $x y$  have  $?A x \in ?T ?A y \in ?T$  by auto
  from  $R$ -in[ $OF this(1)$ ]  $R$ -in[ $OF this(2)$ ] have  $?R (?A x) \in ?A x ?R$ 
 $(?A y) \in ?A y$  by auto
  then have  $(x, ?R (?A x)) \in r (?R (?A y), y) \in r (?R (?A x), x) \in r$ 
 $(y, ?R (?A y)) \in r$  by auto
  with trans[unfolded trans-def] have  $(x, y) \in r \longleftrightarrow (?R (?A x), ?R (?A$ 
 $y)) \in r$  by meson
} note repr = this
interpret interp: enumerateable  $\{?A x \mid x. x \in X\} \lambda A B. A \neq B \wedge (?R$ 

```

```

A, ?R B) ∈ r
proof (standard, goal-cases)
  case 1
  from fin show ?case by auto
next
  case 2
  with total-lifted show ?case by auto
next
  case 3
  then show ?case unfolding transp-def
proof (standard, standard, standard, standard, standard, goal-cases)
  case (1 A B C)
  note A = this
  with trans-lifted have (?R A, ?R C) ∈ r by blast
  moreover have A ≠ C
  proof (rule ccontr, goal-cases)
  case 1
  with A have (?R A, ?R B) ∈ r (?R B, ?R A) ∈ r by auto
  with eq-lifted[OF A(1,2)] A show False by auto
  qed
  ultimately show ?case by auto
qed
next
  case 4
  { fix A B assume A: A ∈ ?T and B: B ∈ ?T and neq: A ≠ B (?R A,
  ?R B) ∈ r
  with eq-lifted[OF A B] neq have ¬ (?R B, ?R A) ∈ r by auto
  }
  then show ?case by auto
qed
from interp.order-fun[OF subset-refl] obtain f :: 'a set ⇒ nat where
  f: ∀ x ∈ ?T. ∀ y ∈ ?T. f x < f y ⟷ x ≠ y ∧ (?R x, ?R y) ∈ r
  f ⊆ {0..card ?T - 1}
by auto
let ?f = λ x. if x ∈ X then f (?A x) else 0
  { fix x y assume x: x ∈ X and y: y ∈ X
  have ?f x ≤ ?f y ⟷ (x, y) ∈ r
  proof (cases x = y)
  case True
  with refl x show ?thesis unfolding refl-on-def by auto
  }
next
  case False
  note F = this
  from ex x y have *: ?A x ∈ ?T ?A y ∈ ?T x ∈ ?A x y ∈ ?A y by

```

```

auto
  show ?thesis
  proof (cases  $(x, y) \in r \wedge (y, x) \in r$ )
    case True
      from eq-lifted'[OF *] True x y have  $?f x = ?f y$  by auto
      with True show ?thesis by auto
    next
      case False
        with A-dest'[OF *(1,3), of y] *(4) have **:  $?A x \neq ?A y$  by auto
        from total-lifted[OF *(1,2) this] have  $(?R (?A x), ?R (?A y)) \in r$ 
 $\vee (?R (?A y), ?R (?A x)) \in r$  .
        then have neq:  $?f x \neq ?f y$ 
        proof (standard, goal-cases)
          case 1
            with f *(1,2) ** have  $f (?A x) < f (?A y)$  by auto
            with * show ?case by auto
          next
            case 2
              with f *(1,2) ** have  $f (?A y) < f (?A x)$  by auto
              with * show ?case by auto
          qed
        then have ?thesis =  $(?f x < ?f y \longleftrightarrow (x, y) \in r)$  by linarith
        moreover from f ** * have  $(?f x < ?f y \longleftrightarrow (?R (?A x), ?R (?A$ 
 $y)) \in r)$  by auto
        moreover from repr * have  $\dots \longleftrightarrow (x, y) \in r$  by auto
        ultimately show ?thesis by auto
      qed
    qed
  }
  then have  $\forall x \in X. \forall y \in X. ?f x \leq ?f y \longleftrightarrow (x, y) \in r$  by blast
  then show ?thesis ..
qed

```

### 1.3.3 Finiteness

```

lemma pairwise-finiteI:
  assumes finite {b.  $\exists a. P a b$ } (is finite ?B)
  assumes finite {a.  $\exists b. P a b$ }
  shows finite {(a,b).  $P a b$ } (is finite ?C)
proof -
  from assms(1) have finite ?B .
  let ?f =  $\lambda b. \{(a,b) \mid a. P a b\}$ 
  { fix b
    have  $?f b \subseteq \{(a,b) \mid a. \exists b. P a b\}$  by blast
  }

```

**moreover have** *finite* ... **using** *assms(2)* **by** *auto*  
**ultimately have** *finite* (?f b) **by** (*blast intro: finite-subset*)  
**}**  
**with** *assms(1)* **have** *finite* ( $\bigcup$  (?f ' ?B)) **by** *auto*  
**moreover have** ?C  $\subseteq$   $\bigcup$  (?f ' ?B) **by** *auto*  
**ultimately show** ?thesis **by** (*blast intro: finite-subset*)  
**qed**

**lemma** *finite-ex-and1*:  
**assumes** *finite* {b.  $\exists a. P a b$ } (**is** *finite* ?A)  
**shows** *finite* {b.  $\exists a. P a b \wedge Q a b$ } (**is** *finite* ?B)  
**proof** –  
**have** ?B  $\subseteq$  ?A **by** *auto*  
**with** *assms* **show** ?thesis **by** (*blast intro: finite-subset*)  
**qed**

**lemma** *finite-ex-and2*:  
**assumes** *finite* {b.  $\exists a. Q a b$ } (**is** *finite* ?A)  
**shows** *finite* {b.  $\exists a. P a b \wedge Q a b$ } (**is** *finite* ?B)  
**proof** –  
**have** ?B  $\subseteq$  ?A **by** *auto*  
**with** *assms* **show** ?thesis **by** (*blast intro: finite-subset*)  
**qed**

### 1.3.4 Numbering the elements of finite sets

**lemma** *upt-last-append*:  $a \leq b \implies [a..<b] @ [b] = [a ..< Suc b]$  **by** (*induction b*) *auto*

**lemma** *map-of-zip-dom-to-range*:  
 $a \in \text{set } A \implies \text{length } B = \text{length } A \implies \text{the } (\text{map-of } (\text{zip } A B) a) \in \text{set } B$   
**by** (*metis map-of-SomeD map-of-zip-is-None option.collapse set-zip-rightD*)

**lemma** *zip-range-id*:  
 $\text{length } A = \text{length } B \implies \text{snd } \text{' set } (\text{zip } A B) = \text{set } B$   
**by** (*metis map-snd-zip set-map*)

**lemma** *map-of-zip-in-range*:  
 $\text{distinct } A \implies \text{length } B = \text{length } A \implies b \in \text{set } B \implies \exists a \in \text{set } A. \text{the } (\text{map-of } (\text{zip } A B) a) = b$

**proof** *goal-cases*

**case** 1

**from** *ran-distinct*[*of zip A B*] 1(1,2) **have**  
 $\text{ran } (\text{map-of } (\text{zip } A B)) = \text{set } B$

by (auto simp: zip-range-id)  
 with 1(3) obtain a where map-of (zip A B) a = Some b unfolding  
 ran-def by auto  
 with map-of-zip-is-Some[OF 1(2)[symmetric]] have the (map-of (zip A  
 B) a) = b a ∈ set A by auto  
 then show ?case by blast  
 qed

**lemma** *distinct-zip-inj*:

$distinct\ ys \implies (a, b) \in set\ (zip\ xs\ ys) \implies (c, b) \in set\ (zip\ xs\ ys) \implies a = c$

**proof** (induction ys arbitrary: xs)

  case Nil then show ?case by auto

next

  case (Cons y ys)

  from this(3) have  $xs \neq []$  by auto

  then obtain z zs where  $xs = z \# zs$  by (cases xs) auto

  show ?case

**proof** (cases (a, b) ∈ set (zip zs ys))

    case True

    note T = this

    then have b: b ∈ set ys by (meson in-set-zipE)

    show ?thesis

**proof** (cases (c, b) ∈ set (zip zs ys))

    case True

    with T Cons show ?thesis by auto

  next

    case False

    with Cons.prem1 xs b show ?thesis by auto

  qed

next

  case False

  with Cons.prem1 xs have b: a = z b = y by auto

  show ?thesis

**proof** (cases (c, b) ∈ set (zip zs ys))

    case True

    then have b ∈ set ys by (meson in-set-zipE)

    with b <distinct (y#ys)> show ?thesis by auto

  next

    case False

    with Cons.prem1 xs b show ?thesis by auto

  qed

qed

qed

**lemma** *map-of-zip-distinct-inj*:  
*distinct B*  $\implies$  *length A = length B*  $\implies$  *inj-on (the o map-of (zip A B))*  
*(set A)*  
**unfolding** *inj-on-def* **proof** (*clarify, goal-cases*)  
  **case** (*1 x y*)  
  **with** *map-of-zip-is-Some[OF 1(2)]* **obtain** *a* **where**  
    *map-of (zip A B) x = Some a* *map-of (zip A B) y = Some a*  
  **by** *auto*  
  **then** **have**  $(x, a) \in \text{set } (\text{zip } A \ B)$   $(y, a) \in \text{set } (\text{zip } A \ B)$  **using** *map-of-SomeD*  
**by** *metis+*  
  **from** *distinct-zip-inj[OF - this]* **1** **show** *?case* **by** *auto*  
**qed**

**lemma** *nat-not-ge-1D*:  $\neg \text{Suc } 0 \leq x \implies x = 0$  **by** *auto*

**lemma** *standard-numbering*:  
  **assumes** *finite A*  
  **obtains**  $v :: 'a \Rightarrow \text{nat}$  **and**  $n$  **where** *bij-betw v A {1..n}*  
  **and**  $\forall c \in A. v \ c > 0$   
  **and**  $\forall c. c \notin A \longrightarrow v \ c > n$   
**proof** –  
  **from** *assms* **obtain**  $L$  **where**  $L: \text{distinct } L$   $\text{set } L = A$  **by** (*meson finite-distinct-list*)  
  **let**  $?N = \text{length } L + 1$   
  **let**  $?P = \text{zip } L \ [1..<?N]$   
  **let**  $?v = \lambda x. \text{let } v = \text{map-of } ?P \ x \text{ in if } v = \text{None} \text{ then } ?N \text{ else the } v$   
  **from** *length-upt* **have**  $\text{len}: \text{length } [1..<?N] = \text{length } L$  **by** *auto* (*cases L, auto*)  
  **then** **have**  $\text{lsimp}: \text{length } [\text{Suc } 0 \ ..<\text{Suc } (\text{length } L)] = \text{length } L$  **by** *simp*  
  **note**  $*$   $= \text{map-of-zip-dom-to-range}[OF - \text{len}]$   
  **have** *bij-betw ?v A {1..length L}* **unfolding** *bij-betw-def*  
  **proof**  
  **show**  $?v \ ` \ A = \{1..\text{length } L\}$  **apply** *auto*  
  **apply** (*auto simp: L*)[]  
  **apply** (*auto simp only: upt-last-append*)[] **using**  $*$  **apply** *force*  
  **using**  $*$  **apply** (*simp only: upt-last-append*) **apply** *force*  
  **apply** (*simp only: upt-last-append*) **using**  $L(2)$  **apply** (*auto dest: nat-not-ge-1D*)  
  **apply** (*subgoal-tac x \in \text{set } [1..< \text{length } L + 1]*)  
  **apply** (*force dest!: map-of-zip-in-range[OF L(1) len]*)  
  **apply** *auto*  
  **done**  
**next**

```

from  $L$  map-of-zip-distinct-inj[OF distinct-upt, of L 1 length L + 1] len
have inj-on (the o map-of ?P)  $A$  by auto
moreover have inj-on (the o map-of ?P)  $A = inj-on ?v A$ 
using len L(2) by  $-$  (rule inj-on-cong, auto)
ultimately show inj-on ?v A by blast
qed
moreover have  $\forall c \in A. ?v c > 0$ 
proof
  fix  $c$ 
  show  $?v c > 0$ 
  proof (cases c ∈ set L)
    case False
    then show ?thesis by auto
  next
    case True
    with dom-map-of-zip[OF len[symmetric]] obtain  $x$  where
      Some x = map-of ?P c x ∈ set [1..<length L + 1]
    by (metis * domIff option.collapse)
    then have  $?v c \in set [1..<length L + 1]$  using  $* True len$  by auto
    then show ?thesis by auto
  qed
qed
moreover have  $\forall c. c \notin A \longrightarrow ?v c > length L$  using  $L$  by auto
ultimately show ?thesis ..
qed

```

### 1.3.5 Products

**lemma** *prod-set-fst-id*:

```

 $x = y$  if  $\forall a \in x. fst a = b \forall a \in y. fst a = b$  snd ' x = snd ' y
using that by (auto 4 6 simp: fst-def snd-def image-def split: prod.splits)

```

**end**

## 2 Graphs

**theory** *Graphs*

**imports**

*More-List1 Stream-More*

*HOL-Library.Rewrite*

**begin**

## 2.1 Basic Definitions and Theorems

**locale** *Graph-Defs* =

**fixes**  $E :: 'a \Rightarrow 'a \Rightarrow bool$

**begin**

**inductive** *steps* **where**

*Single*:  $steps\ [x] \ |$

*Cons*:  $steps\ (x \# y \# xs) \ \mathbf{if}\ E\ x\ y\ steps\ (y \# xs)$

**lemmas** [*intro*] = *steps.intros*

**lemma** *steps-append*:

$steps\ (xs\ @\ tl\ ys) \ \mathbf{if}\ steps\ xs\ steps\ ys\ last\ xs = hd\ ys$

**using** *that* **by** *induction* (*auto* 4 4 *elim*: *steps.cases*)

**lemma** *steps-append'*:

$steps\ xs \ \mathbf{if}\ steps\ as\ steps\ bs\ last\ as = hd\ bs\ as\ @\ tl\ bs = xs$

**using** *steps-append* **that** **by** *blast*

**coinductive** *run* **where**

$run\ (x \#\# y \#\# xs) \ \mathbf{if}\ E\ x\ y\ run\ (y \#\# xs)$

**lemmas** [*intro*] = *run.intros*

**lemma** *steps-appendD1*:

$steps\ xs \ \mathbf{if}\ steps\ (xs\ @\ ys) \ xs \neq []$

**using** *that* **proof** (*induction* *xs*)

**case** *Nil*

**then show** *?case* **by** *auto*

**next**

**case** (*Cons* *a* *xs*)

**then show** *?case*

**by** – (*cases* *xs*; *auto* *elim*: *steps.cases*)

**qed**

**lemma** *steps-appendD2*:

$steps\ ys \ \mathbf{if}\ steps\ (xs\ @\ ys) \ ys \neq []$

**using** *that* **by** (*induction* *xs*) (*auto* *elim*: *steps.cases*)

**lemma** *steps-appendD3*:

$steps\ (xs\ @\ [x]) \wedge E\ x\ y \ \mathbf{if}\ steps\ (xs\ @\ [x, y])$

**using** *that* **proof** (*induction* *xs*)

**case** *Nil*

```

then show ?case by (auto elim!: steps.cases)
next
  case prems: (Cons a xs)
  then show ?case by (cases xs) (auto elim: steps.cases)
qed

```

```

lemma steps-ConsD:
  steps xs if steps (x # xs) xs ≠ []
  using that by (auto elim: steps.cases)

```

```

lemmas stepsD = steps-ConsD steps-appendD1 steps-appendD2

```

```

lemma steps-alt-induct[consumes 1, case-names Single Snoc]:
  assumes
    steps x (∧x. P [x])
    ∧y x xs. E y x ⇒ steps (xs @ [y]) ⇒ P (xs @ [y]) ⇒ P (xs @ [y,x])
  shows P x
  using assms(1)
  proof (induction rule: rev-induct)
    case Nil
    then show ?case by (auto elim: steps.cases)
  next
    case prems: (snoc x xs)
    then show ?case by (cases xs rule: rev-cases) (auto intro: assms(2,3)
dest!: steps-appendD3)
  qed

```

```

lemma steps-appendI:
  steps (xs @ [x, y]) if steps (xs @ [x]) E x y
  using that
  proof (induction xs)
    case Nil
    then show ?case by auto
  next
    case (Cons a xs)
    then show ?case by (cases xs; auto elim: steps.cases)
  qed

```

```

lemma steps-append-single:
  assumes
    steps xs E (last xs) x xs ≠ []
  shows steps (xs @ [x])
  using assms(3,1,2) by (induction xs rule: list-nonempty-induct) (auto 4
4 elim: steps.cases)

```

```

lemma extend-run:
  assumes
    steps xs E (last xs) x run (x ## ys) xs ≠ []
  shows run (xs @- x ## ys)
  using assms(4,1-3) by (induction xs rule: list-nonempty-induct) (auto
4 3 elim: steps.cases)

```

```

lemma run-cycle:
  assumes steps xs E (last xs) (hd xs) xs ≠ []
  shows run (cycle xs)
  using assms proof (coinduction arbitrary: xs)
  case run
  then show ?case
    apply (rewrite at <cycle xs> stream.collapse[symmetric])
    apply (rewrite at <stl (cycle xs)> stream.collapse[symmetric])
    apply clarsimp
    apply (erule steps.cases)
    subgoal for x
      apply (rule conjI)
      apply (simp; fail)
      apply (rule disjI1)
      apply (inst-existentials xs)
      apply (simp, metis cycle-Cons[of x [], simplified])
      by auto
    subgoal for x y xs'
      apply (rule conjI)
      apply (simp; fail)
      apply (rule disjI1)
      apply (inst-existentials y # xs' @ [x])
      using steps-append-single[of y # xs' x]
      apply (auto elim: steps.cases split: if-split-asm simp: cycle-Cons)
    done
  done
qed

```

```

lemma run-stl:
  run (stl xs) if run xs
  using that by (auto elim: run.cases)

```

```

lemma run-sdrop:
  run (sdrop n xs) if run xs
  using that by (induction n arbitrary: xs) (auto intro: run-stl)

```

**lemma** *run-reachable'*:

**assumes**  $\text{run } (x \#\# \text{xs}) \ E^{**} \ x_0 \ x$   
**shows**  $\text{pred-stream } (\lambda x. \ E^{**} \ x_0 \ x) \ \text{xs}$   
**using** *assms* **by** (*coinduction arbitrary: x xs*) (*auto 4 3 elim: run.cases*)

**lemma** *run-reachable*:

**assumes**  $\text{run } (x_0 \#\# \text{xs})$   
**shows**  $\text{pred-stream } (\lambda x. \ E^{**} \ x_0 \ x) \ \text{xs}$   
**by** (*rule run-reachable'[OF assms]*) *blast*

**lemma** *run-decomp*:

**assumes**  $\text{run } (\text{xs} \ @- \ \text{ys}) \ \text{xs} \neq []$   
**shows**  $\text{steps } \text{xs} \wedge \text{run } \text{ys} \wedge \ E \ (\text{last } \text{xs}) \ (\text{shd } \text{ys})$   
**using** *assms(2,1)* **proof** (*induction xs rule: list-nonempty-induct*)  
**case** (*single x*)  
**then show** *?case* **by** (*auto elim: run.cases*)  
**next**  
**case** (*cons x xs*)  
**then show** *?case* **by** (*cases xs; auto 4 4 elim: run.cases*)  
**qed**

**lemma** *steps-decomp*:

**assumes**  $\text{steps } (\text{xs} \ @ \ \text{ys}) \ \text{xs} \neq [] \ \text{ys} \neq []$   
**shows**  $\text{steps } \text{xs} \wedge \text{steps } \text{ys} \wedge \ E \ (\text{last } \text{xs}) \ (\text{hd } \text{ys})$   
**using** *assms(2,1,3)* **proof** (*induction xs rule: list-nonempty-induct*)  
**case** (*single x*)  
**then show** *?case* **by** (*auto elim: steps.cases*)  
**next**  
**case** (*cons x xs*)  
**then show** *?case* **by** (*cases xs; auto 4 4 elim: steps.cases*)  
**qed**

**lemma** *steps-rotate*:

**assumes**  $\text{steps } (x \# \text{xs} \ @ \ y \# \text{ys} \ @ \ [x])$   
**shows**  $\text{steps } (y \# \text{ys} \ @ \ x \# \text{xs} \ @ \ [y])$   
**proof** –  
**from** *steps-decomp[of x # xs y # ys @ [x]] assms* **have**  
 $\text{steps } (x \# \text{xs}) \ \text{steps } (y \# \text{ys} \ @ \ [x]) \ \ E \ (\text{last } (x \# \text{xs})) \ y$   
**by** *auto*  
**then have**  $\text{steps } ((x \# \text{xs}) \ @ \ [y])$  **by** (*blast intro: steps-append-single*)  
**from** *steps-append[OF ‹steps (y # ys @ [x])› this]* **show** *?thesis* **by** *auto*  
**qed**

**lemma** *run-shift-coinduct[case-names run-shift, consumes 1]*:

```

assumes  $R\ w$ 
  and  $\bigwedge w. R\ w \implies \exists u\ v\ x\ y. w = u\ @- x\ ##\ y\ ##\ v \wedge \text{steps}\ (u\ @\ [x]) \wedge E\ x\ y \wedge R\ (y\ ##\ v)$ 
  shows  $\text{run}\ w$ 
  using  $\text{assms}(2)[OF\ \langle R\ w \rangle]$  proof (coinduction arbitrary: w)
  case ( $\text{run}\ w$ )
  then obtain  $u\ v\ x\ y$  where  $w = u\ @- x\ ##\ y\ ##\ v\ \text{steps}\ (u\ @\ [x])\ E\ x\ y\ R\ (y\ ##\ v)$ 
    by auto
  then show ?case
    apply  $-$ 
    apply ( $\text{drule}\ \text{assms}(2)$ )
    apply ( $\text{cases}\ u$ )
    apply force
  subgoal for  $z\ zs$ 
    apply ( $\text{cases}\ zs$ )
  subgoal
    apply simp
    apply safe
    apply ( $\text{force}\ \text{elim:}\ \text{steps.cases}$ )
    subgoal for  $u'\ v'\ x'\ y'$ 
      by ( $\text{inst-existentials}\ x\ \# u'$ ) ( $\text{cases}\ u'$ ; auto)
    done
  subgoal for  $a\ as$ 
    apply simp
    apply safe
    apply ( $\text{force}\ \text{elim:}\ \text{steps.cases}$ )
    subgoal for  $u'\ v'\ x'\ y'$ 
      apply ( $\text{inst-existentials}\ a\ \# as\ @\ x\ \# u'$ )
      using  $\text{steps-append}[of\ a\ \# as\ @\ [x,\ y]\ u'\ @\ [x']]$ 
      apply simp
      apply ( $\text{drule}\ \text{steps-appendI}[of\ a\ \# as\ x,\ \text{rotated}]$ )
      by ( $\text{cases}\ u'$ ;  $\text{force}\ \text{elim:}\ \text{steps.cases}$ )+
    done
  done
  done
  done
qed

```

**lemma** *run-flat-coinduct[case-names run-shift, consumes 1]:*

```

assumes  $R\ xss$ 
  and
     $\bigwedge xs\ ys\ xss. R\ (xs\ ##\ ys\ ##\ xss) \implies xs \neq [] \wedge \text{steps}\ xs \wedge E\ (\text{last}\ xs)\ (\text{hd}\ ys) \wedge R\ (ys\ ##\ xss)$ 

```

```

shows run (flat xss)
proof –
obtain xs ys xss' where xss = xs ## ys ## xss' by (metis stream.collapse)
with assms(2)[OF assms(1)[unfolded this]] show ?thesis
proof (coinduction arbitrary: xs ys xss' xss rule: run-shift-coinduct)
  case (run-shift xs ys xss' xss)
  from run-shift show ?case
  apply (cases xss')
  apply clarify
  apply (drule assms(2))
  apply (inst-existentials butlast xs tl ys @– flat xss' last xs hd ys)
  apply (cases ys)
  apply (simp; fail)
  subgoal premises prems for x1 x2 z zs
  proof (cases xs = [])
  case True
  with prems show ?thesis
  by auto
  next
  case False
  then have xs = butlast xs @ [last xs] by auto
  then have butlast xs @– last xs ## tail = xs @– tail for tail
  by (metis shift.simps(1,2) shift-append)
  with prems show ?thesis by simp
  qed
  apply (simp; fail)
  apply assumption
  subgoal for ws wss
  by (inst-existentials ys ws wss) (cases ys, auto)
  done
  qed
qed

```

```

lemma steps-non-empty[simp]:
  ¬ steps []
  by (auto elim: steps.cases)

```

```

lemma steps-non-empty'[simp]:
  xs ≠ [] if steps xs
  using that by auto

```

```

lemma steps-replicate:
  steps (hd xs # concat (replicate n (tl xs))) if last xs = hd xs steps xs n >

```

```

0
  using that
proof (induction n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  show ?case
  proof (cases n)
    case 0
    with Suc.prem show ?thesis by (cases xs; auto)
  next
    case prems: (Suc nat)
    from Suc.prem have [simp]: hd xs # tl xs @ ys = xs @ ys for ys
      by (cases xs; auto)
    from Suc.prem have **: tl xs @ ys = tl (xs @ ys) for ys
      by (cases xs; auto)
    from prems Suc show ?thesis
      by (fastforce intro: steps-append')
  qed
qed

```

**notation**  $E$  ( $\langle \cdot \rightarrow \cdot \rangle$  [100, 100] 40)

**abbreviation**  $reaches$  ( $\langle \cdot \rightarrow^* \cdot \rangle$  [100, 100] 40) **where**  $reaches\ x\ y \equiv E^{**}\ x\ y$

**abbreviation**  $reaches1$  ( $\langle \cdot \rightarrow^+ \cdot \rangle$  [100, 100] 40) **where**  $reaches1\ x\ y \equiv E^{++}\ x\ y$

**lemma**  $steps-reaches$ :  
 $hd\ xs \rightarrow^* last\ xs$  **if**  $steps\ xs$   
**using** *that* **by** (*induction xs*) *auto*

**lemma**  $steps-reaches'$ :  
 $x \rightarrow^* y$  **if**  $steps\ xs\ hd\ xs = x\ last\ xs = y$   
**using** *that*  $steps-reaches$  **by** *auto*

**lemma**  $reaches-steps$ :  
 $\exists\ xs.\ hd\ xs = x \wedge last\ xs = y \wedge steps\ xs$  **if**  $x \rightarrow^* y$   
**using** *that*  
**apply** (*induction*)  
**apply** *force*  
**apply** *clarsimp*

**subgoal for  $z\ xs$**   
**by** (*inst-existentials*  $xs\ @\ [z]$ , (*cases*  $xs$ ; *simp*), *auto intro: steps-append-single*)  
**done**

**lemma reaches-steps-iff:**  
 $x \rightarrow^* y \iff (\exists\ xs.\ hd\ xs = x \wedge last\ xs = y \wedge steps\ xs)$   
**using** *steps-reaches reaches-steps* **by** *fast*

**lemma steps-reaches1:**  
 $x \rightarrow^+ y$  **if**  $steps\ (x\ \#\ xs\ @\ [y])$   
**by** (*metis list.sel(1,3) rtranclp-into-tranclp2 snoc-eq-iff-butlast steps.cases steps-reaches that*)

**lemma stepsI:**  
 $steps\ (x\ \#\ xs)$  **if**  $x \rightarrow hd\ xs\ steps\ xs$   
**using** *that* **by** (*cases*  $xs$ ) *auto*

**lemma reaches1-steps:**  
 $\exists\ xs.\ steps\ (x\ \#\ xs\ @\ [y])$  **if**  $x \rightarrow^+ y$   
**proof** –  
**from** *that* **obtain**  $z$  **where**  $x \rightarrow z\ z \rightarrow^* y$   
**by** *atomize-elim (simp add: tranclpD)*  
**from** *reaches-steps[OF this(2)]* **obtain**  $xs$  **where**  $*: hd\ xs = z\ last\ xs = y\ steps\ xs$   
**by** *auto*  
**then obtain**  $xs'$  **where**  $[simp]: xs = xs' @ [y]$   
**by** *atomize-elim (auto 4 3 intro: append-butlast-last-id[symmetric])*  
**with**  $\langle x \rightarrow z \rangle *$  **show** *?thesis*  
**by** (*auto intro: stepsI*)  
**qed**

**lemma reaches1-steps-iff:**  
 $x \rightarrow^+ y \iff (\exists\ xs.\ steps\ (x\ \#\ xs\ @\ [y]))$   
**using** *steps-reaches1 reaches1-steps* **by** *fast*

**lemma reaches-steps-iff2:**  
 $x \rightarrow^* y \iff (x = y \vee (\exists\ vs.\ steps\ (x\ \#\ vs\ @\ [y])))$   
**by** (*simp add: Nitpick.rtranclp-unfold reaches1-steps-iff*)

**lemma reaches1-reaches-iff1:**  
 $x \rightarrow^+ y \iff (\exists\ z.\ x \rightarrow z \wedge z \rightarrow^* y)$   
**by** (*auto dest: tranclpD*)

**lemma reaches1-reaches-iff2:**

$x \rightarrow^+ y \iff (\exists z. x \rightarrow^* z \wedge z \rightarrow y)$   
**apply** *safe*  
**apply** (*metis Nitpick.rtranclp-unfold tranclp.cases*)  
**by** *auto*

**lemma**  
 $x \rightarrow^+ z$  **if**  $x \rightarrow^* y$   $y \rightarrow^+ z$   
**using** *that* **by** *auto*

**lemma**  
 $x \rightarrow^+ z$  **if**  $x \rightarrow^+ y$   $y \rightarrow^* z$   
**using** *that* **by** *auto*

**lemma** *steps-append2*:  
 $steps (xs @ x \# ys)$  **if**  $steps (xs @ [x])$   $steps (x \# ys)$   
**using** *that* **by** (*auto dest: steps-append*)

**lemma** *reaches1-steps-append*:  
**assumes**  $a \rightarrow^+ b$   $steps xs$   $hd xs = b$   
**shows**  $\exists ys. steps (a \# ys @ xs)$   
**using** *assms* **by** (*fastforce intro: steps-append' dest: reaches1-steps*)

**lemma** *steps-last-step*:  
 $\exists a. a \rightarrow last xs$  **if**  $steps xs$   $length xs > 1$   
**using** *that* **by** *induction auto*

**lemma** *steps-remove-cycleE*:  
**assumes**  $steps (a \# xs @ [b])$   
**obtains**  $ys$  **where**  $steps (a \# ys @ [b])$   $distinct ys$   $a \notin set ys$   $b \notin set ys$   
 $set ys \subseteq set xs$   
**using** *assms*

**proof** (*induction length xs arbitrary: xs rule: less-induct*)  
**case** *less*  
**note**  $prems = less.prems(2)$  **and**  $intro = less.prems(1)$  **and**  $IH = less.hyps$   
**consider**  
 $distinct xs$   $a \notin set xs$   $b \notin set xs \mid a \in set xs \mid b \in set xs \mid \neg distinct xs$   
**by** *auto*  
**then consider** (*goal*) *?case*  
 $\mid (a) as bs$  **where**  $xs = as @ a \# bs \mid (b) as bs$  **where**  $xs = as @ b \# bs$   
 $\mid (between) x as bs cs$  **where**  $xs = as @ x \# bs @ x \# cs$   
**using** *prems* **by** (*cases; fastforce dest: not-distinct-decomp simp: split-list*  
*intro: intro*)  
**then show** *?case*  
**proof** *cases*

```

case a
with prems show ?thesis
  by - (rule IH[where xs = bs], auto 4 3 intro: intro dest: stepsD)
next
case b
with prems have steps (a # as @ b # [] @ (bs @ [b]))
  by simp
then have steps (a # as @ [b])
  by (metis Cons-eq-appendI Graph-Defs.steps-appendD1 append-eq-appendI
neq-Nil-conv)
  with b show ?thesis
    by - (rule IH[where xs = as], auto 4 3 dest: stepsD intro: intro)
next
case between
with prems have steps (a # as @ x # cs @ [b])
  by simp (metis
    stepsI append-Cons list.distinct(1) list.sel(1) list.sel(3) steps-append
steps-decomp)
  with between show ?thesis
    by - (rule IH[where xs = as @ x # cs], auto 4 3 intro: intro dest:
stepsD)
  qed
qed

```

```

lemma reaches1-stepsE:
  assumes a →+ b
  obtains xs where steps (a # xs @ [b]) distinct xs a ∉ set xs b ∉ set xs
proof -
  from assms obtain xs where steps (a # xs @ [b])
  by (auto dest: reaches1-steps)
  then show ?thesis
  by - (erule steps-remove-cycleE, rule that)
qed

```

```

lemma reaches-stepsE:
  assumes a →* b
  obtains a = b | xs where steps (a # xs @ [b]) distinct xs a ∉ set xs b ∉
set xs
proof -
  from assms consider a = b | xs where a →+ b
  by (meson rtranclpD)
  then show ?thesis
  by cases ((erule reaches1-stepsE)?; rule that; assumption)+
qed

```

**definition** *sink* **where**

*sink*  $a \equiv \nexists b. a \rightarrow b$

**lemma** *sink-or-cycle*:

**assumes** *finite*  $\{b. \text{reaches } a \ b\}$

**obtains**  $b$  **where** *reaches*  $a \ b$  *sink*  $b \mid b$  **where** *reaches*  $a \ b$  *reaches1*  $b \ b$

**proof** –

**let**  $?S = \{b. \text{reaches1 } a \ b\}$

**have**  $?S \subseteq \{b. \text{reaches } a \ b\}$

**by** *auto*

**then have** *finite*  $?S$

**using** *assms* **by** (*rule finite-subset*)

**then show** *?thesis*

**using** *that*

**proof** (*induction*  $?S$  *arbitrary: a rule: finite-psubset-induct*)

**case** *psubset*

**consider** (*empty*) *Collect* (*reaches1*  $a$ ) =  $\{\} \mid b$  **where** *reaches1*  $a \ b$

**by** *auto*

**then show** *?case*

**proof** *cases*

**case** *empty*

**then have** *sink*  $a$

**unfolding** *sink-def* **by** *auto*

**with** *psubset.prem*s **show** *?thesis*

**by** *auto*

**next**

**case**  $2$

**show** *?thesis*

**proof** (*cases* *reaches*  $b \ a$ )

**case** *True*

**with**  $\langle \text{reaches1 } a \ b \rangle$  **have** *reaches1*  $a \ a$

**by** *auto*

**with** *psubset.prem*s **show** *?thesis*

**by** *auto*

**next**

**case** *False*

**show** *?thesis*

**proof** (*cases* *reaches1*  $b \ b$ )

**case** *True*

**with**  $\langle \text{reaches1 } a \ b \rangle$  *psubset.prem*s **show** *?thesis*

**by** (*auto intro: tranclp-into-rtranclp*)

**next**

**case** *False*

```

with  $\langle \neg \text{reaches } b \ a \rangle \langle \text{reaches1 } a \ b \rangle$  have  $\text{Collect } (\text{reaches1 } b) \subset$ 
 $\text{Collect } (\text{reaches1 } a)$ 
by  $(\text{intro } \text{psubsetI})$  auto
then show  $?thesis$ 
using  $\langle \text{reaches1 } a \ b \rangle$   $\text{psubset.prem}$ 
by  $-$   $(\text{erule } \text{psubset.hyps}; \text{meson } \text{tranclp-into-rtranclp } \text{tran-}$ 
 $\text{clp-rtranclp-tranclp})$ 
qed
qed
qed
qed
qed

```

A directed graph where every node has at least one ingoing edge, contains a directed cycle.

**lemma** *directed-graph-indegree-ge-1-cycle'*:

```

assumes  $\text{finite } S \ S \neq \{\}$   $\forall y \in S. \exists x \in S. E \ x \ y$ 
shows  $\exists x \in S. \exists y. E \ x \ y \wedge E^{**} \ y \ x$ 
using  $\text{assms}$ 
proof  $(\text{induction arbitrary: } E \ \text{rule: } \text{finite-ne-induct})$ 
case  $(\text{singleton } x)$ 
then show  $?case$  by  $\text{auto}$ 

```

**next**

```

case  $(\text{insert } x \ S \ E)$ 
from  $\text{insert.prem}$  obtain  $y$  where  $y \in \text{insert } x \ S \ E \ y \ x$ 
by  $\text{auto}$ 
show  $?case$ 
proof  $(\text{cases } y = x)$ 
case  $\text{True}$ 
with  $\langle E \ y \ x \rangle$  show  $?thesis$  by  $\text{auto}$ 
next
case  $\text{False}$ 
with  $\langle y \in - \rangle$  have  $y \in S$  by  $\text{auto}$ 
define  $E'$  where  $E' \ a \ b \equiv E \ a \ b \vee (a = y \wedge E \ x \ b)$  for  $a \ b$ 
have  $E'-E: \exists c. E \ a \ c \wedge E^{**} \ c \ b$  if  $E' \ a \ b$  for  $a \ b$ 
using  $\text{that } \langle E \ y \ x \rangle$  unfolding  $E'-\text{def}$  by  $\text{auto}$ 
have  $[\text{intro}]: E^{**} \ a \ b$  if  $E' \ a \ b$  for  $a \ b$ 
using  $\text{that } \langle E \ y \ x \rangle$  unfolding  $E'-\text{def}$  by  $\text{auto}$ 
have  $[\text{intro}]: E^{**} \ a \ b$  if  $E'^{**} \ a \ b$  for  $a \ b$ 
using  $\text{that}$  by  $(\text{induction}; \text{blast } \text{intro: } \text{rtranclp-trans})$ 
have  $\forall y \in S. \exists x \in S. E' \ x \ y$ 
proof  $(\text{rule } \text{ball})$ 
fix  $b$  assume  $b \in S$ 
with  $\text{insert.prem}$  obtain  $a$  where  $a \in \text{insert } x \ S \ E \ a \ b$ 

```

```

    by auto
  show  $\exists a \in S. E' a b$ 
  proof (cases  $a = x$ )
    case True
      with  $\langle E a b \rangle$  have  $E' y b$  unfolding  $E'$ -def by simp
      with  $\langle y \in S \rangle$  show ?thesis ..
    next
      case False
      with  $\langle a \in S \rangle \langle E a b \rangle$  show ?thesis unfolding  $E'$ -def by auto
  qed
  from insert.IH[OF this] obtain  $x y$  where  $x \in S E' x y E'^{*} y x$  by
safe
  then show ?thesis by (blast intro: rtranclp-trans dest:  $E'-E$ )
  qed
  qed

```

**lemma** *directed-graph-indegree-ge-1-cycle:*  
 assumes  $finite\ S\ S \neq \{\}$   $\forall y \in S. \exists x \in S. E\ x\ y$   
 shows  $\exists x \in S. \exists y. x \rightarrow^+ x$   
 using *directed-graph-indegree-ge-1-cycle'*[OF *assms*] *reaches1-reaches-iff1*  
 by *blast*

Vertices of a graph

**definition**  $vertices = \{x. \exists y. E\ x\ y \vee E\ y\ x\}$

**lemma** *reaches1-verts:*  
 assumes  $x \rightarrow^+ y$   
 shows  $x \in vertices$  and  $y \in vertices$   
 using *assms reaches1-reaches-iff2 reaches1-reaches-iff1 vertices-def* by  
*blast+*

**lemmas** *graphI =*  
*steps.intros*  
*steps-append-single*  
*steps-reaches'*  
*stepsI*

**end**

## 2.2 Graphs with a Start Node

**locale** *Graph-Start-Defs = Graph-Defs +*

```

fixes  $s_0 :: 'a$ 
begin

definition reachable where
  reachable =  $E^{**} s_0$ 

lemma start-reachable[intro!, simp]:
  reachable  $s_0$ 
  unfolding reachable-def by auto

lemma reachable-step:
  reachable  $b$  if reachable  $a$   $E$   $a$   $b$ 
  using that unfolding reachable-def by auto

lemma reachable-reaches:
  reachable  $b$  if reachable  $a$   $a \rightarrow^* b$ 
  using that(2,1) by induction (auto intro: reachable-step)

lemma reachable-steps-append:
  assumes reachable  $a$  steps  $xs$   $hd\ xs = a$  last  $xs = b$ 
  shows reachable  $b$ 
  using assms by (auto intro: graphI reachable-reaches)

lemmas steps-reachable = reachable-steps-append[of  $s_0$ , simplified]

lemma reachable-steps-elem:
  reachable  $y$  if reachable  $x$  steps  $xs$   $y \in set\ xs$   $hd\ xs = x$ 
proof –
  from  $\langle y \in set\ xs \rangle$  obtain as bs where [simp]:  $xs = as @ y \# bs$ 
  by (auto simp: in-set-conv-decomp)
  show ?thesis
  proof (cases  $as = []$ )
    case True
      with that show ?thesis
      by simp
    next
      case False

    from  $\langle steps\ xs \rangle$  have steps ( $as @ [y]$ )
      by (auto intro: stepsD)
    with  $\langle as \neq [] \rangle$   $\langle hd\ xs = x \rangle$   $\langle reachable\ x \rangle$  show ?thesis
      by (auto 4 3 intro: reachable-reaches graphI)
  qed
qed

```

**lemma** *reachable-steps*:  
 $\exists xs. \text{steps } xs \wedge \text{hd } xs = s_0 \wedge \text{last } xs = x$  **if** *reachable*  $x$   
**using** *that unfolding reachable-def*  
**proof** *induction*  
  **case** *base*  
  **then show** *?case* **by** (*inst-existentials*  $[s_0]$ ; *force*)  
**next**  
  **case** (*step*  $y z$ )  
  **from** *step.IH* **obtain**  $xs$  **where**  $\text{steps } xs \ s_0 = \text{hd } xs \ y = \text{last } xs$  **by** *clarsimp*  
  **with** *step.hyps* **show** *?case*  
  **apply** (*inst-existentials*  $xs$  @  $[z]$ )  
  **apply** (*force intro: graphI*)  
  **by** (*cases*  $xs$ ; *auto*)  
**qed**

**lemma** *reachable-cycle-iff*:  
 $\text{reachable } x \wedge x \rightarrow^+ x \longleftrightarrow (\exists ws \ xs. \text{steps } (s_0 \# ws @ [x] @ xs @ [x]))$   
**proof** (*safe, goal-cases*)  
  **case** ( $2 \ ws$ )  
  **then show** *?case*  
  **by** (*auto intro: steps-reachable stepsD*)  
**next**  
  **case** ( $3 \ ws \ xs$ )  
  **then show** *?case*  
  **by** (*auto intro: stepsD steps-reaches1*)  
**next**  
  **case** *prems: 1*  
  **from**  $\langle \text{reachable } x \rangle$  *prems(2)* **have**  $s_0 \rightarrow^+ x$   
  **unfolding** *reachable-def* **by** *auto*  
  **with**  $\langle x \rightarrow^+ x \rangle$  **show** *?case*  
  **by** (*fastforce intro: steps-append' dest: reaches1-steps*)  
**qed**

**lemma** *reachable-induct*[*consumes 1, case-names start step, induct pred: reachable*]:  
  **assumes** *reachable*  $x$   
  **and**  $P \ s_0$   
  **and**  $\bigwedge a \ b. \text{reachable } a \Longrightarrow P \ a \Longrightarrow a \rightarrow b \Longrightarrow P \ b$   
  **shows**  $P \ x$   
  **using** *assms(1) unfolding reachable-def*  
  **by** *induction (auto intro: assms(2-)[unfolded reachable-def])*

**lemmas** *graphI-aggressive* =

*tranclp-into-rtranclp*  
*rtranclp.rtrancl-into-rtrancl*  
*tranclp.trancl-into-trancl*  
*rtranclp-into-tranclp2*

**lemmas** *graphI-aggressive1* =  
*graphI-aggressive*  
*steps-append'*

**lemmas** *graphI-aggressive2* =  
*graphI-aggressive*  
*stepsD*  
*steps-reaches1*  
*steps-reachable*

**lemmas** *graphD* =  
*reaches1-steps*

**lemmas** *graphD-aggressive* =  
*tranclpD*

**lemmas** *graph-startI* =  
*reachable-reaches*  
*start-reachable*

**end**

## 2.3 Subgraphs

### 2.3.1 Edge-induced Subgraphs

**locale** *Subgraph-Defs* = *G: Graph-Defs* +  
*fixes E' :: 'a ⇒ 'a ⇒ bool*  
**begin**

**sublocale** *G': Graph-Defs E'* .

**end**

**locale** *Subgraph-Start-Defs* = *G: Graph-Start-Defs* +  
*fixes E' :: 'a ⇒ 'a ⇒ bool*  
**begin**

**sublocale** *G': Graph-Start-Defs E' s<sub>0</sub>* .

end

**locale** *Subgraph* = *Subgraph-Defs* +  
  **assumes** *subgraph[intro]*:  $E' a b \implies E a b$   
**begin**

**lemma** *non-subgraph-cycle-decomp*:

$\exists c d. G.reaches a c \wedge E c d \wedge \neg E' c d \wedge G.reaches d b$  **if**  
 $G.reaches1 a b \neg G'.reaches1 a b$  **for**  $a b$

**using** *that*

**proof** *induction*

**case** (*base y*)

**then show** *?case*

**by** *auto*

**next**

**case** (*step y z*)

**show** *?case*

**proof** (*cases E' y z*)

**case** *True*

**with step have**  $\neg G'.reaches1 a y$

**by** (*auto intro: tranclp.trancl-into-trancl*)

**with step obtain**  $c d$  **where**

$G.reaches a c E c d \neg E' c d G.reaches d y$

**by** *auto*

**with**  $\langle E' y z \rangle$  **show** *?thesis*

**by** (*blast intro: rtranclp.rtrancl-into-rtrancl*)

**next**

**case** *False*

**with step show** *?thesis*

**by** (*intro exI conjI*) *auto*

**qed**

**qed**

**lemma** *reaches*:

$G.reaches a b$  **if**  $G'.reaches a b$

**using that by induction** (*auto intro: rtranclp.intros(2)*)

**lemma** *reaches1*:

$G.reaches1 a b$  **if**  $G'.reaches1 a b$

**using that by induction** (*auto intro: tranclp.intros(2)*)

end

**locale** *Subgraph-Start* = *Subgraph-Start-Defs* + *Subgraph*  
**begin**

**lemma** *reachable-subgraph*[*intro*]: *G.reachable b* **if**  $\langle G.reachable a \rangle \langle G'.reaches a b \rangle$  **for** *a b*  
**using that by** (*auto intro: G.graph-startI mono-rtranclp*[*rule-format*, of *E'*])

**lemma** *reachable*:  
*G.reachable x* **if** *G'.reachable x*  
**using that by** (*fastforce simp: G.reachable-def G'.reachable-def*)

**end**

### 2.3.2 Node-induced Subgraphs

**locale** *Subgraph-Node-Defs* = *Graph-Defs* +  
**fixes** *V* :: '*a*  $\Rightarrow$  *bool*  
**begin**

**definition** *E'* **where**  $E' x y \equiv E x y \wedge V x \wedge V y$

**sublocale** *Subgraph E E'* **by** *standard* (*auto simp: E'-def*)

**lemma** *subgraph'*:  
*E' x y* **if** *E x y V x V y*  
**using that unfolding** *E'-def* **by** *auto*

**lemma** *E'-V1*:  
*V x* **if** *E' x y*  
**using that unfolding** *E'-def* **by** *auto*

**lemma** *E'-V2*:  
*V y* **if** *E' x y*  
**using that unfolding** *E'-def* **by** *auto*

**lemma** *G'-reaches-V*:  
*V y* **if** *G'.reaches x y V x*  
**using that by** (*cases*) (*auto intro: E'-V2*)

**lemma** *G'-steps-V-all*:  
*list-all V xs* **if** *G'.steps xs V (hd xs)*  
**using that by** *induction* (*auto intro: E'-V2*)

**lemma** *G'-steps-V-last*:

*V (last xs) if G'.steps xs V (hd xs)*

**using** *that by induction (auto dest: E'-V2)*

**lemmas** *subgraphI = E'-V1 E'-V2 G'-reaches-V*

**lemmas** *subgraphD = E'-V1 E'-V2 G'-reaches-V*

**end**

**locale** *Subgraph-Node-Defs-Notation = Subgraph-Node-Defs*

**begin**

**no-notation** *E (⟨- → -⟩ [100, 100] 40)*

**notation** *E' (⟨- → -⟩ [100, 100] 40)*

**no-notation** *reaches (⟨- →\* -⟩ [100, 100] 40)*

**notation** *G'.reaches (⟨- →\* -⟩ [100, 100] 40)*

**no-notation** *reaches1 (⟨- →+ -⟩ [100, 100] 40)*

**notation** *G'.reaches1 (⟨- →+ -⟩ [100, 100] 40)*

**end**

### 2.3.3 The Reachable Subgraph

**context** *Graph-Start-Defs*

**begin**

**interpretation** *Subgraph-Node-Defs-Notation E reachable .*

**sublocale** *reachable-subgraph: Subgraph-Node-Defs E reachable .*

**lemma** *reachable-supgraph*:

*x → y if E x y reachable x*

**using** *that unfolding E'-def by (auto intro: graph-startI)*

**lemma** *reachable-reaches-equiv*: *reaches x y ⟷ x →\* y if reachable x for x y*

**apply** *standard*

**subgoal** **premises** *prems*

**using** *prems ⟨reachable x⟩*

**by** *induction (auto dest: reachable-supgraph intro: graph-startI graphI-aggressive)*

**subgoal** **premises** *prems*

**using** *prems ⟨reachable x⟩*

by induction (auto dest: subgraph)  
done

**lemma** *reachable-reaches1-equiv*:  $\text{reaches1 } x \ y \longleftrightarrow x \rightarrow^+ y$  **if** *reachable*  $x$   
**for**  $x \ y$   
 apply *standard*  
 subgoal **premises**  $\text{prems}$   
 using  $\text{prems}$   $\langle \text{reachable } x \rangle$   
 by induction (auto dest: *reachable-supgraph* intro: *graph-startI* *graphI-aggressive*)  
 subgoal **premises**  $\text{prems}$   
 using  $\text{prems}$   $\langle \text{reachable } x \rangle$   
 by induction (auto dest: *subgraph*)  
 done

**lemma** *reachable-steps-equiv*:  
 $\text{steps } (x \ \# \ xs) \longleftrightarrow G'.\text{steps } (x \ \# \ xs)$  **if** *reachable*  $x$   
 apply *standard*  
 subgoal **premises**  $\text{prems}$   
 using  $\text{prems}$   $\langle \text{reachable } x \rangle$   
 by (induction  $x \ \# \ xs$  arbitrary:  $x \ xs$ ) (auto dest: *reachable-supgraph*  
 intro: *graph-startI*)  
 subgoal **premises**  $\text{prems}$   
 using  $\text{prems}$  by induction auto  
 done

end

## 2.4 Bundles

**bundle** *graph-automation*  
**begin**

**lemmas** [*intro*] = *Graph-Defs.graphI* *Graph-Start-Defs.graph-startI*  
**lemmas** [*dest*] = *Graph-Start-Defs.graphD*

end

**bundle** *reaches-steps-iff* =  
*Graph-Defs.reaches1-steps-iff* [*iff*]  
*Graph-Defs.reaches-steps-iff* [*iff*]

**bundle** *graph-automation-aggressive*  
**begin**

```

unbundle graph-automation

lemmas [intro] = Graph-Start-Defs.graphI-aggressive
lemmas [dest] = Graph-Start-Defs.graphD-aggressive

```

```

end

```

```

bundle subgraph-automation
begin

```

```

unbundle graph-automation

```

```

lemmas [intro] = Subgraph-Node-Defs.subgraphI
lemmas [dest] = Subgraph-Node-Defs.subgraphD

```

```

end

```

## 2.5 Directed Acyclic Graphs

```

locale DAG = Graph-Defs +
  assumes acyclic:  $\neg E^{++} x x$ 
begin

```

```

lemma topological-numbering:

```

```

  fixes S assumes finite S

```

```

  shows  $\exists f :: - \Rightarrow \text{nat. inj-on } f S \wedge (\forall x \in S. \forall y \in S. E x y \longrightarrow f x < f y)$ 

```

```

  using assms

```

```

proof (induction rule: finite-psubset-induct)

```

```

  case (psubset A)

```

```

  show ?case

```

```

  proof (cases A = {})

```

```

    case True

```

```

    then show ?thesis

```

```

      by simp

```

```

  next

```

```

    case False

```

```

    then obtain x where  $x \in A \forall y \in A. \neg E y x$ 

```

```

      using directed-graph-indegree-ge-1-cycle[OF <finite A>] acyclic by auto

```

```

    let ?A =  $A - \{x\}$ 

```

```

    from  $\langle x \in A \rangle$  have ?A  $\subset A$ 

```

```

      by auto

```

```

    from psubset.IH(1)[OF this] obtain f ::  $- \Rightarrow \text{nat}$  where f:

```

```

      inj-on f ?A  $\forall x \in ?A. \forall y \in ?A. x \rightarrow y \longrightarrow f x < f y$ 

```

```

      by blast

```

```

let ?f = λy. if x ≠ y then f y + 1 else 0
from ⟨x ∈ A⟩ have A = insert x ?A
  by auto
from ⟨inj-on f ?A⟩ have inj-on ?f A
  by (auto simp: inj-on-def)
moreover from f(2) x(2) have ∀ x∈A. ∀ y∈A. x → y → ?f x < ?f y
  by auto
ultimately show ?thesis
  by blast
qed
qed

end

```

## 2.6 Finite Graphs

```

locale Finite-Graph = Graph-Defs +
  assumes finite-graph: finite vertices

```

```

locale Finite-DAG = Finite-Graph + DAG
begin

```

```

lemma finite-reachable:
  finite {y. x →* y} (is finite ?S)
proof –
  have ?S ⊆ insert x vertices
    by (metis insertCI mem-Collect-eq reaches1-verts(2) rtranclpD subsetI)
  also from finite-graph have finite ... ..
  finally show ?thesis .
qed

end

```

## 2.7 Graph Invariants

```

locale Graph-Invariant = Graph-Defs +
  fixes P :: 'a ⇒ bool
  assumes invariant: P a ⇒ a → b ⇒ P b
begin

```

```

lemma invariant-steps:
  list-all P as if steps (a # as) P a
  using that by (induction a # as arbitrary: as a) (auto intro: invariant)

```

**lemma** *invariant-reaches*:

$P\ b$  **if**  $a \rightarrow^* b$   $P\ a$

**using** *that* **by** (*induction*; *blast intro: invariant*)

**lemma** *invariant-run*:

**assumes** *run*:  $\text{run } (x \#\# xs)$  **and**  $P: P\ x$

**shows** *pred-stream*  $P\ (x \#\# xs)$

**using** *run P* **by** (*coinduction arbitrary: x xs*) (*auto 4 3 elim: invariant run.cases*)

Every graph invariant induces a subgraph.

**sublocale** *Subgraph-Node-Defs* **where**  $E = E$  **and**  $V = P$  .

**lemma** *subgraph'*:

**assumes**  $x \rightarrow y$   $P\ x$

**shows**  $E'\ x\ y$

**using** *assms* **by** (*intro subgraph'*) (*auto intro: invariant*)

**lemma** *invariant-steps-iff*:

$G'.\text{steps } (v \# vs) \longleftrightarrow \text{steps } (v \# vs)$  **if**  $P\ v$

**apply** (*rule iffI*)

**subgoal**

**using**  $G'.\text{steps-alt-induct steps-appendI}$  **by** *blast*

**subgoal** **premises** *prems*

**using** *prems*  $\langle P\ v \rangle$  **by** (*induction v # vs arbitrary: v vs*) (*auto intro: subgraph' invariant*)

**done**

**lemma** *invariant-reaches-iff*:

$G'.\text{reaches } u\ v \longleftrightarrow \text{reaches } u\ v$  **if**  $P\ u$

**using** *that* **by** (*simp add: reaches-steps-iff2 G'.reaches-steps-iff2 invariant-steps-iff*)

**lemma** *invariant-reaches1-iff*:

$G'.\text{reaches1 } u\ v \longleftrightarrow \text{reaches1 } u\ v$  **if**  $P\ u$

**using** *that* **by** (*simp add: reaches1-steps-iff G'.reaches1-steps-iff invariant-steps-iff*)

**end**

**locale** *Graph-Invariants* = *Graph-Defs* +

**fixes**  $P\ Q :: 'a \Rightarrow \text{bool}$

**assumes** *invariant*:  $P\ a \Longrightarrow a \rightarrow b \Longrightarrow Q\ b$  **and**  $Q\text{-P}$ :  $Q\ a \Longrightarrow P\ a$

**begin**

**sublocale** *Pre: Graph-Invariant E P*  
 by *standard (blast intro: invariant Q-P)*

**sublocale** *Post: Graph-Invariant E Q*  
 by *standard (blast intro: invariant Q-P)*

**lemma** *invariant-steps:*  
*list-all Q as if steps (a # as) P a*  
 using *that by (induction a # as arbitrary: as a) (auto intro: invariant Q-P)*

**lemma** *invariant-run:*  
 assumes *run: run (x ## xs) and P: P x*  
 shows *pred-stream Q xs*  
 using *run P by (coinduction arbitrary: x xs) (auto 4 4 elim: invariant run.cases intro: Q-P)*

**lemma** *invariant-reaches1:*  
*Q b if a →<sup>+</sup> b P a*  
 using *that by (induction; blast intro: invariant Q-P)*

**end**

**locale** *Graph-Invariant-Start = Graph-Start-Defs + Graph-Invariant +*  
 assumes *P-s<sub>0</sub>: P s<sub>0</sub>*  
**begin**

**lemma** *invariant-steps:*  
*list-all P as if steps (s<sub>0</sub> # as)*  
 using *that P-s<sub>0</sub> by (rule invariant-steps)*

**lemma** *invariant-reaches:*  
*P b if s<sub>0</sub> →\* b*  
 using *invariant-reaches[OF that P-s<sub>0</sub>]* .

**lemmas** *invariant-run = invariant-run[OF - P-s<sub>0</sub>]*

**end**

**locale** *Graph-Invariant-Strong = Graph-Defs +*  
 fixes *P :: 'a ⇒ bool*  
 assumes *invariant: a → b ⇒ P b*  
**begin**

**sublocale** *inv*: *Graph-Invariant by standard (rule invariant)*

**lemma** *P-invariant-steps*:

*list-all P as if steps (a # as)*

**using that by** (*induction a # as arbitrary: as a*) (*auto intro: invariant*)

**lemma** *steps-last-invariant*:

*P (last xs) if steps (x # xs) xs ≠ []*

**using steps-last-step**[*of x # xs*] **that by** (*auto intro: invariant*)

**lemmas** *invariant-reaches = inv.invariant-reaches*

**lemma** *invariant-reaches1*:

*P b if a →<sup>+</sup> b*

**using that by** (*induction; blast intro: invariant*)

**end**

## 2.8 Simulations and Bisimulations

**locale** *Simulation-Defs =*

**fixes** *A :: 'a ⇒ 'a ⇒ bool and B :: 'b ⇒ 'b ⇒ bool*

**and sim :: 'a ⇒ 'b ⇒ bool (infixr <~> 60)**

**begin**

**sublocale** *A: Graph-Defs A .*

**sublocale** *B: Graph-Defs B .*

**end**

**locale** *Simulation = Simulation-Defs +*

**assumes** *A-B-step: ∧ a b a'. A a b ⇒ a ~ a' ⇒ (∃ b'. B a' b' ∧ b ~ b')*

**begin**

**lemma** *simulation-reaches*:

*∃ b'. B\*\* b b' ∧ a' ~ b' if A\*\* a a' a ~ b*

**using that by** (*induction rule: rtranclp-induct*) (*auto intro: rtranclp.intros(2)*)  
*dest: A-B-step*)

**lemma** *simulation-reaches1*:

*∃ b'. B<sup>++</sup> b b' ∧ a' ~ b' if A<sup>++</sup> a a' a ~ b*

**using that by** (*induction rule: tranclp-induct*) (*auto 4 3 intro: tran-clp.intros(2) dest: A-B-step*)

**lemma** *simulation-steps:*

$\exists bs. B.steps (b \# bs) \wedge list-all2 (\lambda a b. a \sim b) as bs$  **if**  $A.steps (a \# as)$   
 $a \sim b$

**using that**

**apply** (*induction a # as arbitrary: a b as*)

**apply** *force*

**apply** (*frule A-B-step, auto*)

**done**

**lemma** *simulation-run:*

$\exists ys. B.run (y \#\# ys) \wedge stream-all2 (\sim) xs ys$  **if**  $A.run (x \#\# xs) x \sim y$

**proof** –

**let**  $?ys = sscan (\lambda a' b. SOME b'. B b b' \wedge a' \sim b')$   $xs y$

**have**  $B.run (y \#\# ?ys)$

**using that by** (*coinduction arbitrary: x y xs*) (*force dest!: someI-ex A-B-step elim: A.run.cases*)

**moreover have**  $stream-all2 (\sim) xs ?ys$

**using that by** (*coinduction arbitrary: x y xs*) (*force dest!: someI-ex A-B-step elim: A.run.cases*)

**ultimately show** *?thesis by blast*

**qed**

**end**

**lemma** (*in Subgraph*) *Subgraph-Simulation:*

*Simulation E' E (=)*

**by** *standard auto*

**locale** *Simulation-Invariant = Simulation-Defs +*

**fixes**  $PA :: 'a \Rightarrow bool$  **and**  $PB :: 'b \Rightarrow bool$

**assumes** *A-B-step*:  $\bigwedge a b a'. A a b \Longrightarrow PA a \Longrightarrow PB a' \Longrightarrow a \sim a' \Longrightarrow$   
 $(\exists b'. B a' b' \wedge b \sim b')$

**assumes** *A-invariant[intro]*:  $\bigwedge a b. PA a \Longrightarrow A a b \Longrightarrow PA b$

**assumes** *B-invariant[intro]*:  $\bigwedge a b. PB a \Longrightarrow B a b \Longrightarrow PB b$

**begin**

**definition**  $equiv' \equiv \lambda a b. a \sim b \wedge PA a \wedge PB b$

**sublocale** *Simulation A B equiv'* **by** *standard (auto dest: A-B-step simp: equiv'-def)*

**sublocale** *PA-invariant*: *Graph-Invariant A PA* **by** *standard blast*

**sublocale** *PB-invariant*: *Graph-Invariant B PB* **by** *standard blast*

**lemma** *simulation-reaches*:

$\exists b'. B^{**} b b' \wedge a' \sim b' \wedge PA a' \wedge PB b'$  **if**  $A^{**} a a' a \sim b PA a PB b$   
**using** *simulation-reaches*[of  $a a' b$ ] **that** **unfolding** *equiv'-def* **by** *simp*

**lemma** *simulation-steps*:

$\exists bs. B.steps (b \# bs) \wedge list-all2 (\lambda a b. a \sim b \wedge PA a \wedge PB b) as bs$   
**if**  $A.steps (a \# as) a \sim b PA a PB b$   
**using** *simulation-steps*[of  $a as b$ ] **that** **unfolding** *equiv'-def* **by** *simp*

**lemma** *simulation-steps'*:

$\exists bs. B.steps (b \# bs) \wedge list-all2 (\lambda a b. a \sim b) as bs \wedge list-all PA as \wedge list-all PB bs$   
**if**  $A.steps (a \# as) a \sim b PA a PB b$   
**using** *simulation-steps'*[OF *that*]  
**by** (*force dest: list-all2-set1 list-all2-set2 simp: list-all-iff elim: list-all2-mono*)

**context**

**fixes**  $f$

**assumes**  $eq: a \sim b \implies b = f a$

**begin**

**lemma** *simulation-steps'-map*:

$\exists bs.$   
 $B.steps (b \# bs) \wedge bs = map f as$   
 $\wedge list-all2 (\lambda a b. a \sim b) as bs$   
 $\wedge list-all PA as \wedge list-all PB bs$   
**if**  $A.steps (a \# as) a \sim b PA a PB b$

**proof** –

**from** *simulation-steps'*[OF *that*] **obtain**  $bs$  **where** *guessed*:

$B.steps (b \# bs)$   
 $list-all2 (\sim) as bs$   
 $list-all PA as$   
 $list-all PB bs$

**by** *safe*

**from** *this*(2) **have**  $bs = map f as$

**by** (*induction; simp add: eq*)

**with** *guessed* **show** *?thesis*

**by** *auto*

**qed**

**end**

**end**

**locale** *Simulation-Invariants* = *Simulation-Defs* +  
  **fixes** *PA QA* :: 'a ⇒ bool **and** *PB QB* :: 'b ⇒ bool  
  **assumes** *A-B-step*:  $\bigwedge a b a'. A a b \implies PA a \implies PB a' \implies a \sim a' \implies$   
  ( $\exists b'. B a' b' \wedge b \sim b'$ )  
  **assumes** *A-invariant[intro]*:  $\bigwedge a b. PA a \implies A a b \implies QA b$   
  **assumes** *B-invariant[intro]*:  $\bigwedge a b. PB a \implies B a b \implies QB b$   
  **assumes** *PA-QA[intro]*:  $\bigwedge a. QA a \implies PA a$  **and** *PB-QB[intro]*:  $\bigwedge a.$   
  *QB a* ⇒ *PB a*  
**begin**

**sublocale** *Pre: Simulation-Invariant A B (∼) PA PB*  
  **by standard** (*auto intro: A-B-step*)

**sublocale** *Post: Simulation-Invariant A B (∼) QA QB*  
  **by standard** (*auto intro: A-B-step*)

**sublocale** *A-invs: Graph-Invariants A PA QA*  
  **by standard auto**

**sublocale** *B-invs: Graph-Invariants B PB QB*  
  **by standard auto**

**lemma** *simulation-reaches1*:  
   $\exists b2. B.reaches1 b1 b2 \wedge a2 \sim b2 \wedge QB b2$  **if** *A.reaches1 a1 a2 a1* ∼ *b1*  
  *PA a1 PB b1*  
  **using that**  
  **by** – (*drule Pre.simulation-reaches1, auto intro: B-invs.invariant-reaches1*  
  *simp: Pre.equiv'-def*)

**lemma** *reaches1-unique*:  
  **assumes** *unique*:  $\bigwedge b2. a \sim b2 \implies QB b2 \implies b2 = b$   
  **and that**: *A.reaches1 a a a* ∼ *b PA a PB b*  
  **shows** *B.reaches1 b b*  
  **using that by** (*auto dest: unique simulation-reaches1*)

**end**

**locale** *Bisimulation* = *Simulation-Defs* +  
  **assumes** *A-B-step*:  $\bigwedge a b a'. A a b \implies a \sim a' \implies (\exists b'. B a' b' \wedge b \sim$   
  *b')*

**assumes** *B-A-step*:  $\bigwedge a a' b'. B a' b' \implies a \sim a' \implies (\exists b. A a b \wedge b \sim b')$

**begin**

**sublocale** *A-B: Simulation A B* ( $\sim$ ) **by** *standard* (*rule A-B-step*)

**sublocale** *B-A: Simulation B A*  $\lambda x y. y \sim x$  **by** *standard* (*rule B-A-step*)

**lemma** *A-B-reaches*:

$\exists b'. B^{**} b b' \wedge a' \sim b'$  **if**  $A^{**} a a' a \sim b$   
**using** *A-B.simulation-reaches*[*OF that*] .

**lemma** *B-A-reaches*:

$\exists b'. A^{**} b b' \wedge b' \sim a'$  **if**  $B^{**} a a' b \sim a$   
**using** *B-A.simulation-reaches*[*OF that*] .

**end**

**locale** *Bisimulation-Invariant = Simulation-Defs* +

**fixes** *PA* ::  $'a \Rightarrow \text{bool}$  **and** *PB* ::  $'b \Rightarrow \text{bool}$

**assumes** *A-B-step*:  $\bigwedge a b a'. A a b \implies a \sim a' \implies PA a \implies PB a' \implies (\exists b'. B a' b' \wedge b \sim b')$

**assumes** *B-A-step*:  $\bigwedge a a' b'. B a' b' \implies a \sim a' \implies PA a \implies PB a' \implies (\exists b. A a b \wedge b \sim b')$

**assumes** *A-invariant*[*intro*]:  $\bigwedge a b. PA a \implies A a b \implies PA b$

**assumes** *B-invariant*[*intro*]:  $\bigwedge a b. PB a \implies B a b \implies PB b$

**begin**

**sublocale** *PA-invariant: Graph-Invariant A PA* **by** *standard blast*

**sublocale** *PB-invariant: Graph-Invariant B PB* **by** *standard blast*

**lemmas** *B-steps-invariant*[*intro*] = *PB-invariant.invariant-reaches*

**definition** *equiv'*  $\equiv \lambda a b. a \sim b \wedge PA a \wedge PB b$

**sublocale** *bisim: Bisimulation A B equiv'*

**by** *standard* (*clarsimp simp add: equiv'-def, frule A-B-step B-A-step, assumption; auto*)+

**sublocale** *A-B: Simulation-Invariant A B* ( $\sim$ ) *PA PB*

**by** (*standard; blast intro: A-B-step B-A-step*)

**sublocale** *B-A: Simulation-Invariant B A*  $\lambda x y. y \sim x$  *PB PA*

by (standard; blast intro: A-B-step B-A-step)

**context**

fixes  $f$

assumes  $eq: a \sim b \iff b = f a$

and  $inj: \forall a b. PB (f a) \wedge PA b \wedge f a = f b \longrightarrow a = b$

**begin**

**lemma** *list-all2-inj-map-eq*:

$as = bs$  **if**  $list\text{-}all2 (\lambda a b. a = f b) (map\ f\ as) bs$   $list\text{-}all\ PB (map\ f\ as)$   
 $list\text{-}all\ PA\ bs$

**using** *that inj*

**by** (*induction map f as bs arbitrary: as rule: list-all2-induct*) (*auto simp: inj-on-def*)

**lemma** *steps-map-equiv*:

$A.steps (a \# as) \iff B.steps (b \# map\ f\ as)$  **if**  $a \sim b$   $PA\ a\ PB\ b$

**using**  $A\text{-}B.simulation\text{-}steps'\text{-}map[of\ f\ a\ as\ b]$   $B\text{-}A.simulation\text{-}steps'[of\ b\ map\ f\ as\ a]$  *that eq*

**by** (*auto dest: list-all2-inj-map-eq*)

**lemma** *steps-map*:

$\exists as. bs = map\ f\ as$  **if**  $B.steps (f\ a \# bs)$   $PA\ a\ PB (f\ a)$

**proof** –

**have**  $a \sim f\ a$  **unfolding** *eq ..*

**from**  $B\text{-}A.simulation\text{-}steps'[OF\ that(1)\ this\ \langle PB\ \rightarrow \rangle\ \langle PA\ \rightarrow \rangle]$  **obtain**  $as$

**where**

$A.steps (a \# as)$

$list\text{-}all2 (\lambda a b. b \sim a) bs\ as$

$list\text{-}all\ PB\ bs$

$list\text{-}all\ PA\ as$

**by** *safe*

**from** *this(2)* **show** *?thesis*

**unfolding** *eq* **by** (*inst-existentials as, induction rule: list-all2-induct, auto*)

**qed**

**lemma** *reaches-equiv*:

$A.reaches\ a\ a' \iff B.reaches (f\ a) (f\ a')$  **if**  $PA\ a\ PB (f\ a)$

**apply** *safe*

**apply** (*drule A-B.simulation-reaches[of a a' f a]; simp add: eq that*)

**apply** (*drule B-A.simulation-reaches*)

**defer**

**apply** (*rule that | clarsimp simp: eq | metis inj*)**+**

done

end

**lemma** *equiv'-D*:  
 $a \sim b$  **if**  $A\text{-}B.\text{equiv}' a b$   
**using** *that unfolding A-B.equiv'-def by auto*

**lemma** *equiv'-rotate-1*:  
 $B\text{-}A.\text{equiv}' b a$  **if**  $A\text{-}B.\text{equiv}' a b$   
**using** *that by (auto simp: B-A.equiv'-def A-B.equiv'-def)*

**lemma** *equiv'-rotate-2*:  
 $A\text{-}B.\text{equiv}' a b$  **if**  $B\text{-}A.\text{equiv}' b a$   
**using** *that by (auto simp: B-A.equiv'-def A-B.equiv'-def)*

**lemma** *stream-all2-equiv'-D*:  
 $\text{stream-all2 } (\sim) xs ys$  **if**  $\text{stream-all2 } A\text{-}B.\text{equiv}' xs ys$   
**using** *stream-all2-weaken[OF that equiv'-D] by fast*

**lemma** *stream-all2-equiv'-D2*:  
 $\text{stream-all2 } B\text{-}A.\text{equiv}' ys xs \implies \text{stream-all2 } ((\sim)^{-1-1}) ys xs$   
**by** *(coinduction arbitrary: xs ys) (auto simp: B-A.equiv'-def)*

**lemma** *stream-all2-rotate-1*:  
 $\text{stream-all2 } B\text{-}A.\text{equiv}' ys xs \implies \text{stream-all2 } A\text{-}B.\text{equiv}' xs ys$   
**by** *(coinduction arbitrary: xs ys) (auto simp: B-A.equiv'-def A-B.equiv'-def)*

**lemma** *stream-all2-rotate-2*:  
 $\text{stream-all2 } A\text{-}B.\text{equiv}' xs ys \implies \text{stream-all2 } B\text{-}A.\text{equiv}' ys xs$   
**by** *(coinduction arbitrary: xs ys) (auto simp: B-A.equiv'-def A-B.equiv'-def)*

end

**locale** *Bisimulation-Invariants = Simulation-Defs +*  
**fixes**  $PA QA :: 'a \Rightarrow \text{bool}$  **and**  $PB QB :: 'b \Rightarrow \text{bool}$   
**assumes**  $A\text{-}B\text{-step}: \bigwedge a b a'. A a b \implies a \sim a' \implies PA a \implies PB a' \implies$   
 $(\exists b'. B a' b' \wedge b \sim b')$   
**assumes**  $B\text{-}A\text{-step}: \bigwedge a a' b'. B a' b' \implies a \sim a' \implies PA a \implies PB a'$   
 $\implies (\exists b. A a b \wedge b \sim b')$   
**assumes**  $A\text{-invariant[intro]}: \bigwedge a b. PA a \implies A a b \implies QA b$   
**assumes**  $B\text{-invariant[intro]}: \bigwedge a b. PB a \implies B a b \implies QB b$   
**assumes**  $PA\text{-}QA\text{[intro]}: \bigwedge a. QA a \implies PA a$  **and**  $PB\text{-}QB\text{[intro]}: \bigwedge a.$   
 $QB a \implies PB a$

**begin**

**sublocale** *PA-invariant: Graph-Invariant A PA* **by** *standard blast*

**sublocale** *PB-invariant: Graph-Invariant B PB* **by** *standard blast*

**sublocale** *QA-invariant: Graph-Invariant A QA* **by** *standard blast*

**sublocale** *QB-invariant: Graph-Invariant B QB* **by** *standard blast*

**sublocale** *Pre-Bisim: Bisimulation-Invariant A B ( $\sim$ ) PA PB*  
**by** *standard (auto intro: A-B-step B-A-step)*

**sublocale** *Post-Bisim: Bisimulation-Invariant A B ( $\sim$ ) QA QB*  
**by** *standard (auto intro: A-B-step B-A-step)*

**sublocale** *A-B: Simulation-Invariants A B ( $\sim$ ) PA QA PB QB*  
**by** *standard (blast intro: A-B-step)+*

**sublocale** *B-A: Simulation-Invariants B A  $\lambda x y. y \sim x$  PB QB PA QA*  
**by** *standard (blast intro: B-A-step)+*

**context**

**fixes** *f*

**assumes** *eq[simp]:  $a \sim b \longleftrightarrow b = f a$*

**and** *inj:  $\forall a b. QB (f a) \wedge QA b \wedge f a = f b \longrightarrow a = b$*

**begin**

**lemmas** *list-all2-inj-map-eq = Post-Bisim.list-all2-inj-map-eq[OF eq inj]*

**lemmas** *steps-map-equiv' = Post-Bisim.steps-map-equiv[OF eq inj]*

**lemma** *list-all2-inj-map-eq'*:

*as = bs* **if** *list-all2 ( $\lambda a b. a = f b$ ) (map f as) bs list-all QB (map f as)*  
*list-all QA bs*

**using** *that* **by** *(rule list-all2-inj-map-eq)*

**lemma** *steps-map-equiv*:

*A.steps (a # as)  $\longleftrightarrow$  B.steps (b # map f as)* **if** *a  $\sim$  b PA a PB b*

**proof**

**assume** *A.steps (a # as)*

**then show** *B.steps (b # map f as)*

**proof** *cases*

**case** *Single*

**then show** *?thesis* **by** *auto*

```

next
  case prems: (Cons a' xs)
  from A-B-step[OF <A a a'> <a ~ b> <PA a> <PB b>] obtain b' where
B b b' a' ~ b'
  by auto
  with steps-map-equiv'[OF <a' ~ b'>, of xs] prems that show ?thesis
  by auto
qed
next
assume B.steps (b # map f as)
then show A.steps (a # as)
proof cases
  case Single
  then show ?thesis by auto
next
case prems: (Cons b' xs)
from B-A-step[OF <B b b'> <a ~ b> <PA a> <PB b>] obtain a' where
A a a' a' ~ b'
  by auto
with that prems have QA a' QB b'
  by auto
with <A a a'> <a' ~ b'> steps-map-equiv'[OF <a' ~ b'>, of tl as] prems
that show ?thesis
  apply clarsimp
  subgoal for z zs
    using inj[rule-format, of z a'] by auto
  done
qed
qed

lemma steps-map:
   $\exists$  as. bs = map f as if B.steps (f a # bs) PA a PB (f a)
  using that proof cases
  case Single
  then show ?thesis by simp
next
case prems: (Cons b' xs)
from B-A-step[OF <B - b'> - <PA a> <PB (f a)>] obtain a' where A a a'
a' ~ b'
  by auto
with that prems have QA a' QB b'
  by auto
with Post-Bisim.steps-map[OF eq inj, of a' xs] prems <a' ~ b'> obtain
ys where xs = map f ys

```

by *auto*  
 with  $\langle bs = \rightarrow \langle a' \sim b' \rangle$  **show** *?thesis*  
 by (*inst-existentials a' # ys*) *auto*  
**qed**

$\llbracket \bigwedge a b. a \sim b = (b = ?f a); \forall a b. QB (?f a) \wedge QA b \wedge ?f a = ?f b \longrightarrow a = b; QA ?a; QB (?f ?a) \rrbracket \Longrightarrow A.reaches ?a ?a' = B.reaches (?f ?a) (?f ?a')$   
 cannot be lifted directly: injectivity cannot be applied for the reflexive case.

**lemma** *reaches1-equiv:*

$A.reaches1 a a' \longleftrightarrow B.reaches1 (f a) (f a')$  **if**  $PA a PB (f a)$

**proof** *safe*

**assume**  $A.reaches1 a a'$

**then obtain**  $a''$  **where** *prems:*  $A a a'' A.reaches a'' a'$

**including** *graph-automation-aggressive* **by** *blast*

**from** *A-B-step[OF  $\langle A a \rightarrow - \text{that} \rangle$ ]* **obtain**  $b$  **where**  $B (f a) b a'' \sim b$

**by** *auto*

**with** *that prems* **have**  $QA a'' QB b$

**by** *auto*

**with** *Post-Bisim.reaches-equiv[OF eq inj, of  $a'' a'$ ]* *prems*  $\langle B (f a) b \rangle \langle a'' \sim b \rangle$

**show**  $B.reaches1 (f a) (f a')$

**by** *auto*

**next**

**assume**  $B.reaches1 (f a) (f a')$

**then obtain**  $b$  **where** *prems:*  $B (f a) b B.reaches b (f a')$

**including** *graph-automation-aggressive* **by** *blast*

**from** *B-A-step[OF  $\langle B - b \rangle - \langle PA a \rangle \langle PB (f a) \rangle$ ]* **obtain**  $a''$  **where**  $A a a'' a'' \sim b$

**by** *auto*

**with** *that prems* **have**  $QA a'' QB b$

**by** *auto*

**with** *Post-Bisim.reaches-equiv[OF eq inj, of  $a'' a'$ ]* *prems*  $\langle A a a'' \rangle \langle a'' \sim b \rangle$

**show**  $A.reaches1 a a'$

**by** *auto*

**qed**

**end**

**end**

**lemma** *Bisimulation-Invariant-composition:*

**assumes**

*Bisimulation-Invariant A B sim1 PA PB*

*Bisimulation-Invariant B C sim2 PB PC*  
**shows**  
*Bisimulation-Invariant A C* ( $\lambda a c. \exists b. PB b \wedge sim1 a b \wedge sim2 b c$ )  
*PA PC*  
**proof** –  
**interpret** *A: Bisimulation-Invariant A B sim1 PA PB*  
**by** (*rule* *assms(1)*)  
**interpret** *B: Bisimulation-Invariant B C sim2 PB PC*  
**by** (*rule* *assms(2)*)  
**show** *?thesis*  
**by** (*standard*; (*blast dest: A.A-B-step B.A-B-step* | *blast dest: A.B-A-step B.B-A-step*))  
**qed**

**lemma** *Bisimulation-Invariant-filter:*

**assumes**  
*Bisimulation-Invariant A B sim PA PB*  
 $\bigwedge a b. sim a b \implies PA a \implies PB b \implies FA a \longleftrightarrow FB b$   
 $\bigwedge a b. A a b \wedge FA b \longleftrightarrow A' a b$   
 $\bigwedge a b. B a b \wedge FB b \longleftrightarrow B' a b$   
**shows**  
*Bisimulation-Invariant A' B' sim PA PB*  
**proof** –  
**interpret** *Bisimulation-Invariant A B sim PA PB*  
**by** (*rule* *assms(1)*)  
**have** *unfold*:  
 $A' = (\lambda a b. A a b \wedge FA b)$   $B' = (\lambda a b. B a b \wedge FB b)$   
**using** *assms(3,4)* **by** *auto*  
**show** *?thesis*  
**unfolding** *unfold*  
**apply** *standard*  
**using** *assms(2)* **apply** (*blast dest: A-B-step*)  
**using** *assms(2)* **apply** (*blast dest: B-A-step*)  
**by** *blast+*  
**qed**

**lemma** *Bisimulation-Invariants-filter:*

**assumes**  
*Bisimulation-Invariants A B sim PA QA PB QB*  
 $\bigwedge a b. QA a \implies QB b \implies FA a \longleftrightarrow FB b$   
 $\bigwedge a b. A a b \wedge FA b \longleftrightarrow A' a b$   
 $\bigwedge a b. B a b \wedge FB b \longleftrightarrow B' a b$   
**shows**  
*Bisimulation-Invariants A' B' sim PA QA PB QB*

**proof** –

**interpret** *Bisimulation-Invariants*  $A B \text{ sim } PA QA PB QB$

**by** (*rule* *assms*(1))

**have** *unfold*:

$A' = (\lambda a b. A a b \wedge FA b) B' = (\lambda a b. B a b \wedge FB b)$

**using** *assms*(3,4) **by** *auto*

**show** *?thesis*

**unfolding** *unfold*

**apply** *standard*

**using** *assms*(2) **apply** (*blast dest: A-B-step*)

**using** *assms*(2) **apply** (*blast dest: B-A-step*)

**by** *blast+*

**qed**

**lemma** *Bisimulation-Invariants-composition*:

**assumes**

*Bisimulation-Invariants*  $A B \text{ sim1 } PA QA PB QB$

*Bisimulation-Invariants*  $B C \text{ sim2 } PB QB PC QC$

**shows**

*Bisimulation-Invariants*  $A C (\lambda a c. \exists b. PB b \wedge \text{sim1 } a b \wedge \text{sim2 } b c)$

$PA QA PC QC$

**proof** –

**interpret**  $A$ : *Bisimulation-Invariants*  $A B \text{ sim1 } PA QA PB QB$

**by** (*rule* *assms*(1))

**interpret**  $B$ : *Bisimulation-Invariants*  $B C \text{ sim2 } PB QB PC QC$

**by** (*rule* *assms*(2))

**show** *?thesis*

**by** (*standard*, *blast dest: A.A-B-step B.A-B-step*) (*blast dest: A.B-A-step*

*B.B-A-step*)+

**qed**

**lemma** *Bisimulation-Invariant-Invariants-composition*:

**assumes**

*Bisimulation-Invariant*  $A B \text{ sim1 } PA PB$

*Bisimulation-Invariants*  $B C \text{ sim2 } PB QB PC QC$

**shows**

*Bisimulation-Invariants*  $A C (\lambda a c. \exists b. PB b \wedge \text{sim1 } a b \wedge \text{sim2 } b c)$

$PA PA PC QC$

**proof** –

**interpret** *Bisimulation-Invariant*  $A B \text{ sim1 } PA PB$

**by** (*rule* *assms*(1))

**interpret**  $B$ : *Bisimulation-Invariants*  $B C \text{ sim2 } PB QB PC QC$

**by** (*rule* *assms*(2))

**interpret**  $A$ : *Bisimulation-Invariants*  $A B \text{ sim1 } PA PA PB QB$

**by** (*standard; blast intro: A-B-step B-A-step*)+  
**show** ?thesis  
**by** (*standard; (blast dest: A.A-B-step B.A-B-step | blast dest: A.B-A-step B.B-A-step)*)  
**qed**

**lemma** *Bisimulation-Invariant-Bisimulation-Invariants*:  
**assumes** *Bisimulation-Invariant A B sim PA PB*  
**shows** *Bisimulation-Invariants A B sim PA PA PB PB*  
**proof** –  
**interpret** *Bisimulation-Invariant A B sim PA PB*  
**by** (*rule assms*)  
**show** ?thesis  
**by** (*standard; blast intro: A-B-step B-A-step*)  
**qed**

**lemma** *Bisimulation-Invariant-strengthen-post*:  
**assumes**  
*Bisimulation-Invariant A B sim PA PB*  
 $\bigwedge a b. PA' a \implies PA b \implies A a b \implies PA' b$   
 $\bigwedge a. PA' a \implies PA a$   
**shows** *Bisimulation-Invariant A B sim PA' PB*  
**proof** –  
**interpret** *Bisimulation-Invariant A B sim PA PB*  
**by** (*rule assms*)  
**show** ?thesis  
**by** (*standard; blast intro: A-B-step B-A-step assms*)  
**qed**

**lemma** *Bisimulation-Invariant-strengthen-post'*:  
**assumes**  
*Bisimulation-Invariant A B sim PA PB*  
 $\bigwedge a b. PB' a \implies PB b \implies B a b \implies PB' b$   
 $\bigwedge a. PB' a \implies PB a$   
**shows** *Bisimulation-Invariant A B sim PA PB'*  
**proof** –  
**interpret** *Bisimulation-Invariant A B sim PA PB*  
**by** (*rule assms*)  
**show** ?thesis  
**by** (*standard; blast intro: A-B-step B-A-step assms*)  
**qed**

**lemma** *Simulation-Invariant-strengthen-post*:  
**assumes**

$$\text{Simulation-Invariant } A \ B \ \text{sim } PA \ PB$$

$$\bigwedge a \ b. \ PA \ a \implies PA \ b \implies A \ a \ b \implies PA' \ b$$

$$\bigwedge a. \ PA' \ a \implies PA \ a$$
**shows** *Simulation-Invariant*  $A \ B \ \text{sim } PA' \ PB$   
**proof** –  
**interpret** *Simulation-Invariant*  $A \ B \ \text{sim } PA \ PB$   
**by** (*rule assms*)  
**show** *?thesis*  
**by** (*standard; blast intro: A-B-step assms*)  
**qed**

**lemma** *Simulation-Invariant-strengthen-post'*:  
**assumes**  

$$\text{Simulation-Invariant } A \ B \ \text{sim } PA \ PB$$

$$\bigwedge a \ b. \ PB \ a \implies PB \ b \implies B \ a \ b \implies PB' \ b$$

$$\bigwedge a. \ PB' \ a \implies PB \ a$$
**shows** *Simulation-Invariant*  $A \ B \ \text{sim } PA \ PB'$   
**proof** –  
**interpret** *Simulation-Invariant*  $A \ B \ \text{sim } PA \ PB$   
**by** (*rule assms*)  
**show** *?thesis*  
**by** (*standard; blast intro: A-B-step assms*)  
**qed**

**lemma** *Simulation-Invariants-strengthen-post*:  
**assumes**  

$$\text{Simulation-Invariants } A \ B \ \text{sim } PA \ QA \ PB \ QB$$

$$\bigwedge a \ b. \ PA \ a \implies QA \ b \implies A \ a \ b \implies QA' \ b$$

$$\bigwedge a. \ QA' \ a \implies QA \ a$$
**shows** *Simulation-Invariants*  $A \ B \ \text{sim } PA \ QA' \ PB \ QB$   
**proof** –  
**interpret** *Simulation-Invariants*  $A \ B \ \text{sim } PA \ QA \ PB \ QB$   
**by** (*rule assms*)  
**show** *?thesis*  
**by** (*standard; blast intro: A-B-step assms*)  
**qed**

**lemma** *Simulation-Invariants-strengthen-post'*:  
**assumes**  

$$\text{Simulation-Invariants } A \ B \ \text{sim } PA \ QA \ PB \ QB$$

$$\bigwedge a \ b. \ PB \ a \implies QB \ b \implies B \ a \ b \implies QB' \ b$$

$$\bigwedge a. \ QB' \ a \implies QB \ a$$
**shows** *Simulation-Invariants*  $A \ B \ \text{sim } PA \ QA \ PB \ QB'$   
**proof** –

**interpret** *Simulation-Invariants A B sim PA QA PB QB*  
 by (rule *assms*)  
**show** *?thesis*  
 by (standard; blast intro: *A-B-step assms*)  
**qed**

**lemma** *Bisimulation-Invariant-sim-replace:*  
**assumes** *Bisimulation-Invariant A B sim PA PB*  
**and**  $\bigwedge a b. PA\ a \implies PB\ b \implies sim\ a\ b \longleftrightarrow sim'\ a\ b$   
**shows** *Bisimulation-Invariant A B sim' PA PB*  
**proof** –  
**interpret** *Bisimulation-Invariant A B sim PA PB*  
 by (rule *assms(1)*)  
**show** *?thesis*  
**apply** *standard*  
**using** *assms(2)* **apply** (*blast dest: A-B-step*)  
**using** *assms(2)* **apply** (*blast dest: B-A-step*)  
 by *blast+*  
**qed**

**end**

## 2.9 CTL

**theory** *CTL*  
**imports** *Graphs*  
**begin**

**lemmas** [*simp*] = *holds.simps*

**context** *Graph-Defs*  
**begin**

**definition**  
 $Alw\text{-}ev\ \varphi\ x \equiv \forall\ xs. run\ (x\ \#\#\ xs) \longrightarrow ev\ (holds\ \varphi)\ (x\ \#\#\ xs)$

**definition**  
 $Alw\text{-}alw\ \varphi\ x \equiv \forall\ xs. run\ (x\ \#\#\ xs) \longrightarrow alw\ (holds\ \varphi)\ (x\ \#\#\ xs)$

**definition**  
 $Ex\text{-}ev\ \varphi\ x \equiv \exists\ xs. run\ (x\ \#\#\ xs) \wedge ev\ (holds\ \varphi)\ (x\ \#\#\ xs)$

**definition**  
 $Ex\text{-}alw\ \varphi\ x \equiv \exists\ xs. run\ (x\ \#\#\ xs) \wedge alw\ (holds\ \varphi)\ (x\ \#\#\ xs)$

**definition**

$leadsto \varphi \psi x \equiv Alw-alw (\lambda x. \varphi x \longrightarrow Alw-ev \psi x) x$

**definition**

$deadlocked x \equiv \neg (\exists y. x \rightarrow y)$

**definition**

$deadlock x \equiv \exists y. reaches\ x\ y \wedge deadlocked\ y$

**lemma no-deadlockD:**

$\neg deadlocked\ y$  **if**  $\neg deadlock\ x\ reaches\ x\ y$   
**using that unfolding deadlock-def by auto**

**lemma not-deadlockedE:**

**assumes**  $\neg deadlocked\ x$   
**obtains**  $y$  **where**  $x \rightarrow y$   
**using** *assms* **unfolding** *deadlocked-def* **by auto**

**lemma holds-Not:**

$holds\ (Not \circ \varphi) = (\lambda x. \neg holds\ \varphi\ x)$   
**by auto**

**lemma Alw-alw-iff:**

$Alw-alw\ \varphi\ x \longleftrightarrow \neg Ex-ev\ (Not \circ \varphi)\ x$   
**unfolding** *Alw-alw-def* *Ex-ev-def* *holds-Not* *not-ev-not[symmetric]* **by simp**

**lemma Ex-alw-iff:**

$Ex-alw\ \varphi\ x \longleftrightarrow \neg Alw-ev\ (Not \circ \varphi)\ x$   
**unfolding** *Alw-ev-def* *Ex-alw-def* *holds-Not* *not-ev-not[symmetric]* **by simp**

**lemma leadsto-iff:**

$leadsto\ \varphi\ \psi\ x \longleftrightarrow \neg Ex-ev\ (\lambda x. \varphi\ x \wedge \neg Alw-ev\ \psi\ x)\ x$   
**unfolding** *leadsto-def* *Alw-alw-iff* **by** (*simp add: comp-def*)

**lemma run-siterate-from:**

**assumes**  $\forall y. x \rightarrow^* y \longrightarrow (\exists z. y \rightarrow z)$   
**shows**  $run\ (siterate\ (\lambda x. SOME\ y. x \rightarrow y))\ x$  **(is**  $run\ (siterate\ ?f\ x)$ **)**  
**using** *assms*

**proof** (*coinduction arbitrary: x*)

**case** (*run x*)  
**let**  $?y = SOME\ y. x \rightarrow y$   
**from** *run* **have**  $x \rightarrow ?y$   
**by** (*auto intro: someI*)

**with** *run show ?case including graph-automation-aggressive by auto*  
**qed**

**lemma** *extend-run'*:

*run zs if steps xs run ys last xs = shd ys xs @- stl ys = zs*

**by** (*metis*

*Graph-Defs.run.cases Graph-Defs.steps-non-empty' extend-run  
stream.exhaust-sel stream.inject that*)

**lemma** *no-deadlock-run-extend*:

$\exists ys. run (x \#\# xs @- ys) \text{ if } \neg \text{deadlock } x \text{ steps } (x \# xs)$

**proof** –

**include** *graph-automation*

**let**  $?x = last (x \# xs)$  **let**  $?f = \lambda x. SOME y. x \rightarrow y$  **let**  $?ys = siterate$   
 $?f ?x$

**have**  $\exists z. y \rightarrow z$  **if**  $?x \rightarrow^* y$  **for**  $y$

**proof** –

**from**  $\langle steps (x \# xs) \rangle$  **have**  $x \rightarrow^* ?x$

**by** *auto*

**from**  $\langle x \rightarrow^* ?x \rangle \langle ?x \rightarrow^* y \rangle$  **have**  $x \rightarrow^* y$

**by** *auto*

**with**  $\langle \neg \text{deadlock } x \rangle$  **show** *?thesis*

**by** (*auto dest: no-deadlockD elim: not-deadlockedE*)

**qed**

**then have** *run ?ys*

**by** (*blast intro: run-siterate-from*)

**with**  $\langle steps (x \# xs) \rangle$  **show** *?thesis*

**by** (*fastforce intro: extend-run'*)

**qed**

**lemma** *Ex-ev*:

*Ex-ev*  $\varphi x \longleftrightarrow (\exists y. x \rightarrow^* y \wedge \varphi y) \text{ if } \neg \text{deadlock } x$

**unfolding** *Ex-ev-def*

**proof** *safe*

**fix**  $xs$  **assume** *prems: run (x \#\# xs) ev (holds \varphi) (x \#\# xs)*

**show**  $\exists y. x \rightarrow^* y \wedge \varphi y$

**proof** (*cases \varphi x*)

**case** *True*

**then show** *?thesis*

**by** *auto*

**next**

**case** *False*

```

with prems obtain  $y$   $ys$   $zs$  where
   $\varphi$   $y$   $xs = ys @ - y \#\# zs$   $y \notin set$   $ys$ 
  unfolding  $ev$ -holds-sset by (auto elim!:split-stream-first')
with prems have  $steps$  ( $x \# ys @ [y]$ )
  by (auto intro: run-decomp[THEN conjunct1])
with  $\langle \varphi y \rangle$  show  $?thesis$ 
  including  $graph$ -automation by (auto 4 3)
qed
next
fix  $y$  assume  $x \rightarrow^* y$   $\varphi y$ 
then obtain  $xs$  where
   $\varphi$  ( $last$   $xs$ )  $x = hd$   $xs$   $steps$   $xs$   $y = last$   $xs$ 
  by (auto dest: reaches-steps)
then show  $\exists xs. run$  ( $x \#\# xs$ )  $\wedge ev$  ( $holds$   $\varphi$ ) ( $x \#\# xs$ )
  by (cases xs)
  (auto split: if-split-asm simp: ev-holds-sset dest!: no-deadlock-run-extend[OF that])
qed

lemma  $Alw$ - $ev$ :
   $Alw$ - $ev$   $\varphi$   $x \longleftrightarrow \neg (\exists xs. run$  ( $x \#\# xs$ )  $\wedge alw$  ( $holds$  ( $Not \circ \varphi$ )) ( $x \#\# xs$ ))
  unfolding  $Alw$ - $ev$ - $def$ 
proof (safe, goal-cases)
  case  $prems$ : ( $1$   $xs$ )
  then have  $ev$  ( $holds$   $\varphi$ ) ( $x \#\# xs$ ) by auto
  then show  $?case$ 
    using  $prems(2,3)$  by induction (auto intro: run-stl)
next
  case  $prems$ : ( $2$   $xs$ )
  then have  $\neg alw$  ( $holds$  ( $Not \circ \varphi$ )) ( $x \#\# xs$ )
    by auto
  moreover have  $(\lambda x. \neg holds$  ( $Not \circ \varphi$ )  $x) = holds$   $\varphi$ 
    by (rule ext) simp
  ultimately show  $?case$ 
    unfolding  $not$ - $alw$ - $iff$  by simp
qed

lemma  $leadsto$ - $iff'$ :
   $leadsto$   $\varphi$   $\psi$   $x \longleftrightarrow (\nexists y. x \rightarrow^* y \wedge \varphi y \wedge \neg Alw$ - $ev$   $\psi$   $y)$  if  $\neg$   $deadlock$   $x$ 
  unfolding  $leadsto$ - $iff$   $Ex$ - $ev$ [ $OF \langle \neg deadlock x \rangle$ ] ..

end

```

**context** *Bisimulation-Invariant*  
**begin**

**context**  
  **fixes**  $\varphi :: 'a \Rightarrow \text{bool}$  **and**  $\psi :: 'b \Rightarrow \text{bool}$   
  **assumes** *compatible*:  $A\text{-}B.\text{equiv}' a b \implies \varphi a \longleftrightarrow \psi b$   
**begin**

**lemma** *ev- $\psi$ - $\varphi$* :  
   $ev$  (*holds*  $\varphi$ )  $xs$  **if** *stream-all2*  $B\text{-}A.\text{equiv}' ys xs$   $ev$  (*holds*  $\psi$ )  $ys$   
  **using** *that*  
  **apply**  $-$   
  **apply** (*drule* *stream-all2-rotate-1*)  
  **apply** (*drule* *ev-imp-shift*)  
  **apply** *clarify*  
  **unfolding** *stream-all2-shift2*  
  **apply** (*subst* (*asm*) *stream.rel-sel*)  
  **apply** (*auto intro!*: *ev-shift dest!*: *compatible[symmetric]*)  
  **done**

**lemma** *ev- $\varphi$ - $\psi$* :  
   $ev$  (*holds*  $\psi$ )  $ys$  **if** *stream-all2*  $A\text{-}B.\text{equiv}' xs ys$   $ev$  (*holds*  $\varphi$ )  $xs$   
  **using** *that*  
  **apply**  $-$   
  **apply** (*subst* (*asm*) *stream.rel-flip[symmetric]*)  
  **apply** (*drule* *ev-imp-shift*)  
  **apply** *clarify*  
  **unfolding** *stream-all2-shift2*  
  **apply** (*subst* (*asm*) *stream.rel-sel*)  
  **apply** (*auto intro!*: *ev-shift dest!*: *compatible*)  
  **done**

**lemma** *Ex-ev-iff*:  
   $A.\text{Ex-ev } \varphi a \longleftrightarrow B.\text{Ex-ev } \psi b$  **if**  $A\text{-}B.\text{equiv}' a b$   
  **unfolding** *Graph-Defs.Ex-ev-def*  
  **apply** *safe*  
  **subgoal for**  $xs$   
    **apply** (*drule*  $A\text{-}B.\text{simulation-run}[of a xs b]$ )  
    **subgoal**  
      **using** *that* .  
    **apply** *clarify*  
  **subgoal for**  $ys$   
    **apply** (*inst-existentials*  $ys$ )  
    **using** *that*

```

    apply (auto intro!: ev-φ-ψ dest: stream-all2-rotate-1)
  done
done
subgoal for ys
  apply (drule B-A.simulation-run[of b ys a])
  subgoal
    using that by (rule equiv'-rotate-1)
  apply clarify
  subgoal for xs
    apply (inst-existentials xs)
    using that
    apply (auto intro!: ev-ψ-φ dest: equiv'-rotate-1)
  done
done
done

```

**lemma** *Alw-ev-iff*:

$A.Alw-ev\ \varphi\ a \longleftrightarrow B.Alw-ev\ \psi\ b$  **if**  $A-B.equiv'\ a\ b$

**unfolding** *Graph-Defs.Alw-ev-def*

**apply** *safe*

**subgoal for** *ys*

**apply** (*drule B-A.simulation-run*[of *b ys a*])

**subgoal**

**using that by** (*rule equiv'-rotate-1*)

**apply** *safe*

**subgoal for** *xs*

**apply** (*inst-existentials xs*)

**apply** (*elim allE impE, assumption*)

**using that**

**apply** (*auto intro!: ev-φ-ψ dest: stream-all2-rotate-1*)

**done**

**done**

**subgoal for** *xs*

**apply** (*drule A-B.simulation-run*[of *a xs b*])

**subgoal**

**using that** .

**apply** *safe*

**subgoal for** *ys*

**apply** (*inst-existentials ys*)

**apply** (*elim allE impE, assumption*)

**using that**

**apply** (*auto intro!: ev-ψ-φ elim!: equiv'-rotate-1 stream-all2-rotate-2*)

**done**

**done**

```

done

end

context
  fixes  $\varphi :: 'a \Rightarrow \text{bool}$  and  $\psi :: 'b \Rightarrow \text{bool}$ 
  assumes compatible1:  $A\text{-}B.\text{equiv}' a b \implies \varphi a \longleftrightarrow \psi b$ 
begin

lemma Alw-alw-iff-strong:
   $A.\text{Alw-alw } \varphi a \longleftrightarrow B.\text{Alw-alw } \psi b$  if  $A\text{-}B.\text{equiv}' a b$ 
  unfolding Graph-Defs.Alw-alw-iff using that by (auto dest: compatible1
intro!: Ex-ev-iff)

lemma Ex-alw-iff:
   $A.\text{Ex-alw } \varphi a \longleftrightarrow B.\text{Ex-alw } \psi b$  if  $A\text{-}B.\text{equiv}' a b$ 
  unfolding Graph-Defs.Ex-alw-iff using that by (auto dest: compatible1
intro!: Alw-ev-iff)

end

context
  fixes  $\varphi :: 'a \Rightarrow \text{bool}$  and  $\psi :: 'b \Rightarrow \text{bool}$ 
  and  $\varphi' :: 'a \Rightarrow \text{bool}$  and  $\psi' :: 'b \Rightarrow \text{bool}$ 
  assumes compatible1:  $A\text{-}B.\text{equiv}' a b \implies \varphi a \longleftrightarrow \psi b$ 
  assumes compatible2:  $A\text{-}B.\text{equiv}' a b \implies \varphi' a \longleftrightarrow \psi' b$ 
begin

lemma Leadsto-iff:
   $A.\text{leadsto } \varphi \varphi' a \longleftrightarrow B.\text{leadsto } \psi \psi' b$  if  $A\text{-}B.\text{equiv}' a b$ 
  unfolding Graph-Defs.leadsto-def
  by (auto
    dest: Alw-ev-iff[of  $\varphi' \psi'$ , rotated] compatible1 compatible2 equiv'-D
    intro!: Alw-alw-iff-strong[OF - that]
  )

end

lemma deadlock-iff:
   $A.\text{deadlock } a \longleftrightarrow B.\text{deadlock } b$  if  $a \sim b$  PA a PB b
  using that unfolding A.deadlock-def A.deadlocked-def B.deadlock-def B.deadlocked-def
  by (force dest: A-B-step B-A-step B-A.simulation-reaches A-B.simulation-reaches)

end

```

**lemmas** [*simp del*] = *holds.simps*

**end**

**theory** *Timed-Automata*

**imports** *library/Graphs Difference-Bound-Matrices.Zones*

**begin**

### 3 Basic Definitions and Semantics

#### 3.1 Syntactic Definition

Clock constraints

**datatype** (*'c, 't*) *aconstraint* =

*LT 'c 't* |

*LE 'c 't* |

*EQ 'c 't* |

*GT 'c 't* |

*GE 'c 't*

**type-synonym** (*'c, 't*) *cconstraint* = (*'c, 't*) *aconstraint list*

For an informal description of timed automata we refer to Bengtsson and Yi [BY03]. We define a timed automaton  $A$

**type-synonym**

(*'c, 'time, 's*) *invassn* = *'s*  $\Rightarrow$  (*'c, 'time*) *cconstraint*

**type-synonym**

(*'a, 'c, 'time, 's*) *transition* = *'s* \* (*'c, 'time*) *cconstraint* \* *'a* \* *'c list* \* *'s*

**type-synonym**

(*'a, 'c, 'time, 's*) *ta* = (*'a, 'c, 'time, 's*) *transition set* \* (*'c, 'time, 's*) *invassn*

**definition** *trans-of* :: (*'a, 'c, 'time, 's*) *ta*  $\Rightarrow$  (*'a, 'c, 'time, 's*) *transition set*

**where**

*trans-of*  $\equiv$  *fst*

**definition** *inv-of* :: (*'a, 'c, 'time, 's*) *ta*  $\Rightarrow$  (*'c, 'time, 's*) *invassn* **where**

*inv-of*  $\equiv$  *snd*

**abbreviation** *transition* ::

(*'a, 'c, 'time, 's*) *ta*  $\Rightarrow$  *'s*  $\Rightarrow$  (*'c, 'time*) *cconstraint*  $\Rightarrow$  *'a*  $\Rightarrow$  *'c list*  $\Rightarrow$  *'s*  $\Rightarrow$  *bool*

$(\langle - \vdash - \longrightarrow^{g,a,r} - \rangle [61,61,61,61,61,61] 61)$  **where**  
 $(A \vdash l \longrightarrow^{g,a,r} l') \equiv (l,g,a,r,l') \in \text{trans-of } A$

### 3.1.1 Collecting Information About Clocks

**fun** *constraint-clk* :: ('c, 't) *aconstraint*  $\Rightarrow$  'c

**where**

*constraint-clk* (LT c -) = c |  
*constraint-clk* (LE c -) = c |  
*constraint-clk* (EQ c -) = c |  
*constraint-clk* (GE c -) = c |  
*constraint-clk* (GT c -) = c

**definition** *collect-clks* :: ('c, 't) *cconstraint*  $\Rightarrow$  'c *set*

**where**

*collect-clks* cc  $\equiv$  *constraint-clk* ' set cc

**fun** *constraint-pair* :: ('c, 't) *aconstraint*  $\Rightarrow$  ('c \* 't)

**where**

*constraint-pair* (LT x m) = (x, m) |  
*constraint-pair* (LE x m) = (x, m) |  
*constraint-pair* (EQ x m) = (x, m) |  
*constraint-pair* (GE x m) = (x, m) |  
*constraint-pair* (GT x m) = (x, m)

**definition** *collect-clock-pairs* :: ('c, 't) *cconstraint*  $\Rightarrow$  ('c \* 't) *set*

**where**

*collect-clock-pairs* cc = *constraint-pair* ' set cc

**definition** *collect-clkt* :: ('a, 'c, 't, 's) *transition set*  $\Rightarrow$  ('c \* 't) *set*

**where**

*collect-clkt* S =  $\bigcup$  {*collect-clock-pairs* (fst (snd t)) | t . t  $\in$  S}

**definition** *collect-clki* :: ('c, 't, 's) *invassn*  $\Rightarrow$  ('c \* 't) *set*

**where**

*collect-clki* I =  $\bigcup$  {*collect-clock-pairs* (I x) | x. True}

**definition** *clkp-set* :: ('a, 'c, 't, 's) *ta*  $\Rightarrow$  ('c \* 't) *set*

**where**

*clkp-set* A = *collect-clki* (inv-of A)  $\cup$  *collect-clkt* (trans-of A)

**definition** *collect-clkvt* :: ('a, 'c, 't, 's) *transition set*  $\Rightarrow$  'c *set*

**where**

*collect-clkvt* S =  $\bigcup$  {set ((fst o snd o snd o snd) t) | t . t  $\in$  S}

**abbreviation** *clk-set* **where**  $\text{clk-set } A \equiv \text{fst } \text{‘ clkp-set } A \cup \text{collect-clkvt}$   
*(trans-of A)*

**inductive** *valid-abstraction*

**where**

$\llbracket \forall (x, m) \in \text{clkp-set } A. m \leq k \ x \wedge x \in X \wedge m \in \mathbb{N}; \text{collect-clkvt (trans-of}$   
 $A) \subseteq X; \text{finite } X \rrbracket$   
 $\implies \text{valid-abstraction } A \ X \ k$

### 3.2 Operational Semantics

**inductive** *clock-val-a* ( $\langle \cdot \vdash_a \cdot \rangle$  [62, 62] 62) **where**

$\llbracket u \ c \ < \ d \rrbracket \implies u \vdash_a \text{LT } c \ d \ |$   
 $\llbracket u \ c \ \leq \ d \rrbracket \implies u \vdash_a \text{LE } c \ d \ |$   
 $\llbracket u \ c \ = \ d \rrbracket \implies u \vdash_a \text{EQ } c \ d \ |$   
 $\llbracket u \ c \ \geq \ d \rrbracket \implies u \vdash_a \text{GE } c \ d \ |$   
 $\llbracket u \ c \ > \ d \rrbracket \implies u \vdash_a \text{GT } c \ d$

**inductive-cases**[*elim!*]:  $u \vdash_a \text{LT } c \ d$

**inductive-cases**[*elim!*]:  $u \vdash_a \text{LE } c \ d$

**inductive-cases**[*elim!*]:  $u \vdash_a \text{EQ } c \ d$

**inductive-cases**[*elim!*]:  $u \vdash_a \text{GE } c \ d$

**inductive-cases**[*elim!*]:  $u \vdash_a \text{GT } c \ d$

**declare** *clock-val-a.intros*[*intro*]

**definition** *clock-val* :: ( $'c, 't$ ) *cval*  $\implies ('c, 't::\text{time})$  *cconstraint*  $\implies \text{bool}$  ( $\langle \cdot \vdash$   
 $\cdot \rangle$  [62, 62] 62)

**where**

$u \vdash \text{cc} = \text{list-all (clock-val-a } u) \ \text{cc}$

**lemma** *atomic-guard-continuous*:

**assumes**  $u \vdash_a \ g \ u \oplus t \vdash_a \ g \ 0 \leq (t'::'t::\text{time}) \ t' \leq t$

**shows**  $u \oplus t' \vdash_a \ g$

**using** *assms*

**by** (*induction*  $g$ ;

*auto* 4 3

*simp: cval-add-def order-le-less-subst2 order-subst2 add-increasing2*

*intro: less-le-trans*

)

**lemma** *guard-continuous*:

**assumes**  $u \vdash g \ u \oplus t \vdash g \ 0 \leq t' \ t' \leq t$   
**shows**  $u \oplus t' \vdash g$   
**using** *assms by (auto intro: atomic-guard-continuous simp: clock-val-def list-all-iff)*

**inductive** *step-t* ::

$(\ 'a, \ 'c, \ 't, \ 's) \ ta \Rightarrow \ 's \Rightarrow \ ('c, \ 't) \ cval \Rightarrow \ ('t::time) \Rightarrow \ 's \Rightarrow \ ('c, \ 't) \ cval \Rightarrow$   
*bool*  
 $(\langle - \vdash \langle -, - \rangle \rightarrow \langle -, - \rangle \rangle [61,61,61] \ 61)$

**where**

$\llbracket u \oplus d \vdash \text{inv-of } A \ l; \ d \geq 0 \rrbracket \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^d \langle l, u \oplus d \rangle$

**lemmas** [*intro*] = *step-t.intros*

**context**

**notes** *step-t.cases[elim!]* *step-t.intros[intro!]*

**begin**

**lemma** *step-t-determinacy1*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^d \langle l'', u'' \rangle \Longrightarrow l' = l''$

**by** *auto*

**lemma** *step-t-determinacy2*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^d \langle l'', u'' \rangle \Longrightarrow u' = u''$

**by** *auto*

**lemma** *step-t-cont1*:

$d \geq 0 \Longrightarrow e \geq 0 \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow A \vdash \langle l', u' \rangle \rightarrow^e \langle l'', u'' \rangle$   
 $\Longrightarrow A \vdash \langle l, u \rangle \rightarrow^{d+e} \langle l'', u'' \rangle$

**proof** –

**assume**  $A: d \geq 0 \ e \geq 0 \ A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \ A \vdash \langle l', u' \rangle \rightarrow^e \langle l'', u'' \rangle$

**hence**  $u' = (u \oplus d) \ u'' = (u' \oplus e)$  **by** *auto*

**hence**  $u'' = (u \oplus (d + e))$  **unfolding** *cval-add-def* **by** *auto*

**with**  $A$  **show** *?thesis* **by** *auto*

**qed**

**end**

**inductive** *step-a* ::

$(\ 'a, \ 'c, \ 't, \ 's) \ ta \Rightarrow \ 's \Rightarrow \ ('c, \ ('t::time)) \ cval \Rightarrow \ 'a \Rightarrow \ 's \Rightarrow \ ('c, \ 't) \ cval \Rightarrow$   
*bool*  
 $(\langle - \vdash \langle -, - \rangle \rightarrow \langle -, - \rangle \rangle [61,61,61] \ 61)$

**where**

$\llbracket A \vdash l \longrightarrow^{g,a,r} l'; u \vdash g; u' \vdash \text{inv-of } A \ l'; u' = [r \rightarrow 0]u \rrbracket \Longrightarrow (A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle)$

**inductive** *step* ::

$(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, ('t::\text{time})) \text{ cval} \Rightarrow 's \Rightarrow ('c, 't) \text{ cval} \Rightarrow \text{bool}$   
 $(\langle - \vdash \langle -, - \rangle \rightarrow \langle -, - \rangle \rangle [61,61,61] \ 61)$

**where**

*step-a*:  $A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle \Longrightarrow (A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle) \mid$

*step-t*:  $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow (A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle)$

**declare** *step.intros*[*intro*]

**declare** *step.cases*[*elim*]

**inductive**

*steps* ::  $(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, ('t::\text{time})) \text{ cval} \Rightarrow 's \Rightarrow ('c, 't) \text{ cval} \Rightarrow \text{bool}$

$(\langle - \vdash \langle -, - \rangle \rightarrow^* \langle -, - \rangle \rangle [61,61,61] \ 61)$

**where**

*refl*:  $A \vdash \langle l, u \rangle \rightarrow^* \langle l, u \rangle \mid$

*step*:  $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \Longrightarrow A \vdash \langle l', u' \rangle \rightarrow^* \langle l'', u'' \rangle \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^* \langle l'', u'' \rangle$

**declare** *steps.intros*[*intro*]

### 3.3 Contracting Runs

**inductive** *step'* ::

$(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, ('t::\text{time})) \text{ cval} \Rightarrow 's \Rightarrow ('c, 't) \text{ cval} \Rightarrow \text{bool}$   
 $(\langle - \vdash'' \langle -, - \rangle \rightarrow \langle -, - \rangle \rangle [61,61,61] \ 61)$

**where**

*step'*:  $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow A \vdash \langle l', u' \rangle \rightarrow_a \langle l'', u'' \rangle \Longrightarrow A \vdash' \langle l, u \rangle \rightarrow \langle l'', u'' \rangle$

**lemmas** *step'*[*intro*]

**lemma** *step'-altI*:

**assumes**

$A \vdash l \longrightarrow^{g,a,r} l' \ u \oplus d \vdash g \ u \oplus d \vdash \text{inv-of } A \ l \ 0 \leq d$

$u' = [r \rightarrow 0](u \oplus d) \ u' \vdash \text{inv-of } A \ l'$

**shows**  $A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$

**using** *assms* **by** (*auto intro: step-a.intros*)

**inductive**

*steps'* ::  $(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, ('t::\text{time})) \text{ cval} \Rightarrow 's \Rightarrow ('c, 't) \text{ cval}$

$\Rightarrow \text{bool}$   
 $(\langle \vdash'' \langle -, - \rangle \rightarrow^* \langle -, - \rangle \rangle [61,61,61] \ 61)$   
**where**  
 $\text{refl}' : A \vdash' \langle l, u \rangle \rightarrow^* \langle l, u \rangle \mid$   
 $\text{step}' : A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle \Longrightarrow A \vdash' \langle l', u' \rangle \rightarrow^* \langle l'', u'' \rangle \Longrightarrow A \vdash' \langle l, u \rangle$   
 $\rightarrow^* \langle l'', u'' \rangle$

**lemmas**  $\text{steps}'.\text{intros}[\text{intro}]$

**lemma**  $\text{steps}'\text{-altI}$ :

$A \vdash' \langle l, u \rangle \rightarrow^* \langle l'', u'' \rangle$  **if**  $A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$   $A \vdash' \langle l', u' \rangle \rightarrow \langle l'', u'' \rangle$   
**using that by induction auto**

**lemma**  $\text{step-d-refl}[\text{intro}]$ :

$A \vdash \langle l, u \rangle \rightarrow^0 \langle l, u \rangle$  **if**  $u \vdash \text{inv-of } A \ l$

**proof** –

**from that have**  $A \vdash \langle l, u \rangle \rightarrow^0 \langle l, u \oplus 0 \rangle$  **by** – (rule  $\text{step-t.intros}$ ; force  $\text{simp: cval-add-def}$ )

**then show**  $?thesis$  **by** ( $\text{simp add: cval-add-def}$ )

**qed**

**lemma**  $\text{cval-add-simp}$ :

$(u \oplus d) \oplus d' = u \oplus (d + d')$  **for**  $d \ d' :: 't :: \text{time}$   
**unfolding cval-add-def by auto**

**context**

**notes**  $[\text{elim!}] = \text{step}'.\text{cases } \text{step-t.cases}$

**and**  $[\text{intro!}] = \text{step-t.intros}$

**begin**

**lemma**  $\text{step-t-trans}$ :

$A \vdash \langle l, u \rangle \rightarrow^{d+d'} \langle l, u' \rangle$  **if**  $A \vdash \langle l, u \rangle \rightarrow^d \langle l, u' \rangle$   $A \vdash \langle l, u' \rangle \rightarrow^{d'} \langle l, u'' \rangle$   
**using that by** ( $\text{auto simp add: cval-add-simp}$ )

**lemma**  $\text{steps}'\text{-complete}$ :

$\exists u'. A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$  **if**  $A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$   $u \vdash \text{inv-of } A \ l$   
**using that**

**proof** ( $\text{induction}$ )

**case** ( $\text{refl } A \ l \ u$ )

**then show**  $?case$  **by blast**

**next**

**case** ( $\text{step } A \ l \ u \ l' \ u' \ l'' \ u''$ )

**then have**  $u' \vdash \text{inv-of } A \ l'$  **by** ( $\text{auto elim: step-a.cases}$ )

**from**  $\text{step}(1)$  **show**  $?case$

```

proof cases
  case (step-a a)
  with  $\langle u \vdash - \rangle \langle u' \vdash - \rangle$  step( $\mathcal{Z}$ ) show ?thesis by (auto 4 5)
next
  case (step-t d)
  then have [simp]:  $l' = l$  by auto
  from step( $\mathcal{Z}$ )  $\langle u' \vdash - \rangle$  obtain u0 where  $A \vdash' \langle l, u \rangle \rightarrow^* \langle l'', u0 \rangle$  by
auto
  then show ?thesis
proof cases
  case refl'
  then show ?thesis by blast
next
  case (step' l1 u1)
  with step-t show ?thesis by (auto 4 7 intro: step-t-trans)
qed
qed
qed

```

**lemma** steps'-sound:  
 $A \vdash \langle l, u \rangle \rightarrow^* \langle l', u \rangle$  **if**  $A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u \rangle$   
**using that by** (induction; blast)

**lemma** steps-steps'-equiv:  
 $(\exists u'. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u \rangle) \iff (\exists u'. A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u \rangle)$  **if**  $u \vdash$   
*inv-of*  $A \ l$   
**using that** steps'-sound steps'-complete **by** metis

**end**

### 3.4 Zone Semantics

**datatype** 'a action = Tau ( $\langle \tau \rangle$ ) | Action 'a ( $\langle \dashv \rangle$ )

**inductive** step-z ::  
 $(\text{'a}, \text{'c}, \text{'t}, \text{'s}) \text{ta} \Rightarrow \text{'s} \Rightarrow (\text{'c}, (\text{'t}::\text{time})) \text{zone} \Rightarrow \text{'a} \text{action} \Rightarrow \text{'s} \Rightarrow (\text{'c}, \text{'t})$   
 $\text{zone} \Rightarrow \text{bool}$   
 $(\langle - \vdash \langle -, - \rangle \rightsquigarrow_- \langle -, - \rangle \rangle [61,61,61,61] 61)$   
**where**  
step-t-z:  
 $A \vdash \langle l, Z \rangle \rightsquigarrow_\tau \langle l, Z^\dagger \cap \{u. u \vdash \text{inv-of } A \ l\} \rangle$  |  
step-a-z:  
 $A \vdash \langle l, Z \rangle \rightsquigarrow_{1a} \langle l', \text{zone-set } (Z \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\} \rangle$   
**if**  $A \vdash l \longrightarrow_{g,a,r} l'$

**lemmas** *step-z.intros*[*intro*]

**inductive-cases** *step-t-z-E*[*elim*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{\tau} \langle l', u' \rangle$

**inductive-cases** *step-a-z-E*[*elim*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{1a} \langle l', u' \rangle$

### 3.4.1 Zone Semantics for Compressed Runs

**definition**

*step-z* :: (*'a*, *'c*, *'t*, *'s*) *ta*  $\Rightarrow$  *'s*  $\Rightarrow$  (*'c*, (*'t*::*time*)) *zone*  $\Rightarrow$  *'s*  $\Rightarrow$  (*'c*, *'t*) *zone*  $\Rightarrow$  *bool*

( $\langle - \vdash \langle -, - \rangle \rightsquigarrow \langle -, - \rangle$ ) [61,61,61] 61)

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle \equiv (\exists Z' a. A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l, Z' \rangle \wedge A \vdash \langle l, Z' \rangle \rightsquigarrow_{1a} \langle l', Z'' \rangle)$

**abbreviation**

*steps-z* :: (*'a*, *'c*, *'t*, *'s*) *ta*  $\Rightarrow$  *'s*  $\Rightarrow$  (*'c*, (*'t*::*time*)) *zone*  $\Rightarrow$  *'s*  $\Rightarrow$  (*'c*, *'t*) *zone*  $\Rightarrow$  *bool*

( $\langle - \vdash \langle -, - \rangle \rightsquigarrow^* \langle -, - \rangle$ ) [61,61,61] 61)

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z'' \rangle \equiv (\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle)^{**} (l, Z) (l', Z'')$

**context**

**notes** [*elim!*] = *step.cases step'.cases step-t.cases step-z.cases*

**begin**

**lemma** *step-t-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l', Z' \rangle \implies \forall u' \in Z'. \exists u \in Z. \exists d. A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$

**by** (*auto* 4 5 *simp*: *zone-delay-def zone-set-def*)

**lemma** *step-a-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{1a} \langle l', Z' \rangle \implies \forall u' \in Z'. \exists u \in Z. \exists d. A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$

**by** (*auto* 4 4 *simp*: *zone-delay-def zone-set-def intro: step-a.intros*)

**lemma** *step-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \forall u' \in Z'. \exists u \in Z. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$

**by** (*auto* 4 6 *simp*: *zone-delay-def zone-set-def intro: step-a.intros*)

**lemma** *step-a-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow_{1a} \langle l', Z' \rangle \wedge u' \in Z'$

**by** (*auto* 4 4 *simp*: *zone-delay-def zone-set-def elim!: step-a.cases*)

**lemma** *step-t-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l', Z' \rangle \wedge u' \in Z'$

**by** (*auto* 4 4 *simp*: *zone-delay-def zone-set-def elim!*: *step-a.cases*)

**lemma** *step-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies u \in Z \implies \exists Z' a. A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \wedge u' \in Z'$

**by** (*auto* 4 4 *simp*: *zone-delay-def zone-set-def elim!*: *step-a.cases*)

**end**

**lemma** *step-z-sound'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies \forall u' \in Z'. \exists u \in Z. A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$

**unfolding** *step-z'-def* **by** (*fastforce dest!*: *step-t-z-sound step-a-z-sound*)

**lemma** *step-z-complete'*:

$A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \wedge u' \in Z'$

**unfolding** *step-z'-def* **by** (*auto dest!*: *step-a-z-complete step-t-z-complete elim!*: *step'.cases*)

**lemma** *steps-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z' \rangle \implies u' \in Z' \implies \exists u \in Z. A \vdash' \langle l, u \rangle \rightarrow_* \langle l', u' \rangle$

**by** (*induction arbitrary*: *u' rule*: *rtranclp-induct2*;  
*fastforce intro*: *steps'-altI dest!*: *step-z-sound'*)

**lemma** *steps-z-complete*:

$A \vdash' \langle l, u \rangle \rightarrow_* \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z' \rangle \wedge u' \in Z'$

**oops**

**lemma** *ta-zone-sim*:

*Simulation*

$(\lambda(l, u) (l', u'). A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle)$

$(\lambda(l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle)$

$(\lambda(l, u) (l', Z). u \in Z \wedge l = l')$

**by** *standard* (*auto dest!*: *step-z-complete'*)

**lemma** *steps'-iff*:

$(\lambda(l, u) (l', u'). A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle)^* (l, u) (l', u') \iff A \vdash' \langle l, u \rangle \rightarrow_* \langle l', u' \rangle$

**apply** *standard*

**subgoal**

**by** (*induction rule: rtranclp-induct2; blast intro: steps'-altI*)  
**subgoal**  
**by** (*induction rule: steps'.induct; blast intro: converse-rtranclp-into-rtranclp*)  
**done**

**lemma** *steps-z-complete:*

$A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \wedge u' \in Z'$

**using** *Simulation.simulation-reaches[OF ta-zone-sim, of A (l, u) (l', u')]*  
**unfolding** *steps'-iff* **by** *auto*

**end**

### 3.5 From Clock Constraints to DBMs

**theory** *TA-DBM-Operations*

**imports** *Timed-Automata Difference-Bound-Matrices.DBM-Operations*

**begin**

**fun** *abstra* ::

$('c, 't::\{\text{linordered-cancel-ab-monoid-add, uminus}\}) \text{acconstraint} \Rightarrow 't \text{DBM}$   
 $\Rightarrow ('c \Rightarrow \text{nat}) \Rightarrow 't \text{DBM}$

**where**

$\text{abstra } (EQ \ c \ d) \ M \ v =$   
 $(\lambda \ i \ j. \text{if } i = 0 \wedge j = v \ c \ \text{then } \min (M \ i \ j) (Le \ (-d)) \ \text{else if } i = v \ c \wedge$   
 $j = 0 \ \text{then } \min (M \ i \ j) (Le \ d) \ \text{else } M \ i \ j) \ |$   
 $\text{abstra } (LT \ c \ d) \ M \ v =$   
 $(\lambda \ i \ j. \text{if } i = v \ c \wedge j = 0 \ \text{then } \min (M \ i \ j) (Lt \ d) \ \text{else } M \ i \ j) \ |$   
 $\text{abstra } (LE \ c \ d) \ M \ v =$   
 $(\lambda \ i \ j. \text{if } i = v \ c \wedge j = 0 \ \text{then } \min (M \ i \ j) (Le \ d) \ \text{else } M \ i \ j) \ |$   
 $\text{abstra } (GT \ c \ d) \ M \ v =$   
 $(\lambda \ i \ j. \text{if } i = 0 \wedge j = v \ c \ \text{then } \min (M \ i \ j) (Lt \ (-d)) \ \text{else } M \ i \ j) \ |$   
 $\text{abstra } (GE \ c \ d) \ M \ v =$   
 $(\lambda \ i \ j. \text{if } i = 0 \wedge j = v \ c \ \text{then } \min (M \ i \ j) (Le \ (-d)) \ \text{else } M \ i \ j)$

**fun** *abstr* ::  $('c, 't::\{\text{linordered-cancel-ab-monoid-add, uminus}\}) \text{cconstraint}$   
 $\Rightarrow 't \text{DBM} \Rightarrow ('c \Rightarrow \text{nat}) \Rightarrow 't \text{DBM}$

**where**

$\text{abstr } cc \ M \ v = \text{fold } (\lambda \ ac \ M. \ \text{abstra } ac \ M \ v) \ cc \ M$

**lemma** *collect-clks-Cons[simp]:*

$\text{collect-clks } (ac \ \# \ cc) = \text{insert } (\text{constraint-clk } ac) (\text{collect-clks } cc)$

**unfolding** *collect-clks-def* **by** *auto*

**lemma** *abstr-id1*:

$c \notin \text{collect-clks } cc \implies \text{clock-numbering}' v n \implies \forall c \in \text{collect-clks } cc. v c \leq n$

$\implies \text{abstr } cc M v 0 (v c) = M 0 (v c)$

**apply** (*induction cc arbitrary*:  $M c$ )

**apply** (*simp*; *fail*)

**subgoal for**  $a$

**apply** *simp*

**apply** (*cases a*)

**by** *auto*

**done**

**lemma** *abstr-id2*:

$c \notin \text{collect-clks } cc \implies \text{clock-numbering}' v n \implies \forall c \in \text{collect-clks } cc. v c \leq n$

$\implies \text{abstr } cc M v (v c) 0 = M (v c) 0$

**apply** (*induction cc arbitrary*:  $M c$ )

**apply** (*simp*; *fail*)

**subgoal for**  $a$

**apply** *simp*

**apply** (*cases a*)

**by** *auto*

**done**

This lemma is trivial because we constrained our theory to difference constraints.

**lemma** *abstra-id3*:

**assumes** *clock-numbering v*

**shows** *abstra ac M v (v c1) (v c2) = M (v c1) (v c2)*

**proof** –

**have**  $\bigwedge c. v c = 0 \implies \text{False}$

**proof** –

**fix**  $c$  **assume**  $v c = 0$

**moreover from** *assms* **have**  $v c > 0$  **by** *auto*

**ultimately show** *False* **by** *linarith*

**qed**

**then show** *?thesis* **by** (*cases ac*) *auto*

**qed**

**lemma** *abstr-id3*:

$\text{clock-numbering } v \implies \text{abstr } cc M v (v c1) (v c2) = M (v c1) (v c2)$

**by** (*induction cc arbitrary*:  $M$ ) (*auto simp add: abstra-id3*)

**lemma** *abstra-id3'*:  
**assumes**  $\forall c. 0 < v\ c$   
**shows** *abstra ac M v 0 0 = M 0 0*  
**proof** –  
**have**  $\bigwedge c. v\ c = 0 \implies \text{False}$   
**proof** –  
**fix** *c* **assume**  $v\ c = 0$   
**moreover from** *assms* **have**  $v\ c > 0$  **by** *auto*  
**ultimately show** *False* **by** *linarith*  
**qed**  
**then show** *?thesis* **by** (*cases ac*) *auto*  
**qed**

**lemma** *abstr-id3'*:  
*clock-numbering v*  $\implies$  *abstr cc M v 0 0 = M 0 0*  
**by** (*induction cc arbitrary: M*) (*auto simp add: abstra-id3'*)

**lemma** *clock-numberingD*:  
**assumes** *clock-numbering v v c = 0*  
**shows** *A*  
**proof**–  
**from** *assms(1)* **have**  $v\ c > 0$  **by** *auto*  
**with**  $\langle v\ c = 0 \rangle$  **show** *?thesis* **by** *linarith*  
**qed**

**lemma** *dbm-abstra-soundness*:  
 $\llbracket u \vdash_a ac; u \vdash_{v,n} M; \text{clock-numbering}'\ v\ n; v\ (\text{constraint-clk}\ ac) \leq n \rrbracket$   
 $\implies$  *DBM-val-bounded v u (abstra ac M v) n*  
**proof** (*unfold DBM-val-bounded-def, auto, goal-cases*)  
**case** *prems: 1*  
**from** *abstra-id3'* [*OF this(4)*] **have** *abstra ac M v 0 0 = M 0 0* .  
**with** *prems* **show** *?case unfolding dbm-le-def* **by** *auto*  
**next**  
**case** *prems: (2 c)*  
**then have** *clock-numbering' v n* **by** *auto*  
**note**  $A = \text{prems}(1)$  *this prems(6,3)*  
**let**  $?c = \text{constraint-clk}\ ac$   
**show** *?case*  
**proof** (*cases c = ?c*)  
**case** *True*  
**then show** *?thesis using prems* **by** (*cases ac*) (*auto split: split-min*  
*intro: clock-numberingD*)  
**next**

```

    case False
    then show ?thesis using  $A(\exists)$  prems by (cases ac) auto
qed
next
case prems: ( $\exists c$ )
then have clock-numbering'  $v n$  by auto
then have gt0:  $v c > 0$  by auto
let ?c = constraint-clk ac
show ?case
proof (cases  $c = ?c$ )
  case True
  then show ?thesis using prems gt0 by (cases ac) (auto split: split-min
intro: clock-numberingD)
  next
  case False
  then show ?thesis using <clock-numbering'  $v n$ > prems by (cases ac)
auto
qed
next

```

Trivial because of missing difference constraints

```

case prems: ( $\exists c1 c2$ )
from abstra-id3[OF this( $\exists$ )] have abstra ac  $M v (v c1) (v c2) = M (v
c1) (v c2)$  by auto
with prems show ?case by auto
qed

```

**lemma** *dbm-abstr-soundness'*:

$\llbracket u \vdash cc; u \vdash_{v,n} M; \text{clock-numbering}' v n; \forall c \in \text{collect-clks } cc. v c \leq n \rrbracket$   
 $\implies \text{DBM-val-bounded } v u (\text{abstr } cc M v) n$

by (induction cc arbitrary:  $M$ ) (auto simp: clock-val-def dest: dbm-abstra-soundness)

**lemmas** *dbm-abstr-soundness* = *dbm-abstr-soundness'*[OF - *DBM-triv*]

**lemma** *dbm-abstra-completeness*:

$\llbracket \text{DBM-val-bounded } v u (\text{abstr } ac M v) n; \forall c. v c > 0; v (\text{constraint-clk }
ac) \leq n \rrbracket$

$\implies u \vdash_a ac$

**proof** (cases ac, goal-cases)

case prems: ( $\exists c d$ )

then have  $v c \leq n$  by auto

with prems( $\exists, \exists$ ) have dbm-entry-val  $u (\text{Some } c) \text{None} ((\text{abstr } (LT c d)
M v) (v c) 0)$

by (auto simp: DBM-val-bounded-def)

**moreover from**  $\text{prems}(2)$  **have**  $v\ c > 0$  **by** *auto*  
**ultimately show**  $?case$  **using**  $\text{prems}(4)$  **by** (*auto dest: dbm-entry-dbm-min3*)  
**next**  
**case**  $\text{prems}: (2\ c\ d)$   
**from** *this* **have**  $v\ c \leq n$  **by** *auto*  
**with**  $\text{prems}(1,4)$  **have**  $\text{dbm-entry-val}\ u\ (\text{Some}\ c)\ \text{None}\ ((\text{abstra}\ (LE\ c\ d)\ M\ v)\ (v\ c)\ 0)$   
**by** (*auto simp: DBM-val-bounded-def*)  
**moreover from**  $\text{prems}(2)$  **have**  $v\ c > 0$  **by** *auto*  
**ultimately show**  $?case$  **using**  $\text{prems}(4)$  **by** (*auto dest: dbm-entry-dbm-min3*)  
**next**  
**case**  $\text{prems}: (3\ c\ d)$   
**from** *this* **have**  $c: v\ c > 0\ v\ c \leq n$  **by** *auto*  
**with**  $\text{prems}(1,4)$  **have**  $B$ :  
 $\text{dbm-entry-val}\ u\ (\text{Some}\ c)\ \text{None}\ ((\text{abstra}\ (EQ\ c\ d)\ M\ v)\ (v\ c)\ 0)$   
 $\text{dbm-entry-val}\ u\ \text{None}\ (\text{Some}\ c)\ ((\text{abstra}\ (EQ\ c\ d)\ M\ v)\ 0\ (v\ c))$   
**by** (*auto simp: DBM-val-bounded-def*)  
**from**  $c\ B$  **have**  $u\ c \leq d - u\ c \leq -d$  **by** (*auto dest: dbm-entry-dbm-min2 dbm-entry-dbm-min3*)  
**with**  $\text{prems}(4)$  **show**  $?case$  **by** *auto*  
**next**  
**case**  $\text{prems}: (4\ c\ d)$   
**from** *this* **have**  $v\ c \leq n$  **by** *auto*  
**with**  $\text{prems}(1,4)$  **have**  $\text{dbm-entry-val}\ u\ \text{None}\ (\text{Some}\ c)\ ((\text{abstra}\ (GT\ c\ d)\ M\ v)\ 0\ (v\ c))$   
**by** (*auto simp: DBM-val-bounded-def*)  
**moreover from**  $\text{prems}(2)$  **have**  $v\ c > 0$  **by** *auto*  
**ultimately show**  $?case$  **using**  $\text{prems}(4)$  **by** (*auto dest!: dbm-entry-dbm-min2*)  
**next**  
**case**  $\text{prems}: (5\ c\ d)$   
**from** *this* **have**  $v\ c \leq n$  **by** *auto*  
**with**  $\text{prems}(1,4)$  **have**  $\text{dbm-entry-val}\ u\ \text{None}\ (\text{Some}\ c)\ ((\text{abstra}\ (GE\ c\ d)\ M\ v)\ 0\ (v\ c))$   
**by** (*auto simp: DBM-val-bounded-def*)  
**moreover from**  $\text{prems}(2)$  **have**  $v\ c > 0$  **by** *auto*  
**ultimately show**  $?case$  **using**  $\text{prems}(4)$  **by** (*auto dest!: dbm-entry-dbm-min2*)  
**qed**

**lemma** *abstra-mono*:  
 $\text{abstra}\ ac\ M\ v\ i\ j\ \leq\ M\ i\ j$   
**by** (*cases ac*) *auto*

**lemma** *abstra-subset*:  
 $[\text{abstra}\ ac\ M\ v]_{v,n} \subseteq [M]_{v,n}$

**using** *abstra-mono*  
**apply** (*simp add: less-eq*)  
**apply** *safe*  
**by** (*rule DBM-le-subset; force*)

**lemma** *abstr-subset*:  
 $[abstr\ cc\ M\ v]_{v,n} \subseteq [M]_{v,n}$   
**apply** (*induction cc arbitrary: M*)  
**apply** (*simp; fail*)  
**using** *abstra-subset* **by** *fastforce*

**lemma** *dbm-abstra-zone-eq*:  
**assumes** *clock-numbering' v n v (constraint-clk ac) ≤ n*  
**shows**  $[abstra\ ac\ M\ v]_{v,n} = \{u. u \vdash_a ac\} \cap [M]_{v,n}$   
**apply** *safe*  
**subgoal**  
**unfolding** *DBM-zone-repr-def* **using** *assms* **by** (*auto intro: dbm-abstra-completeness*)  
**subgoal**  
**using** *abstra-subset* **by** *blast*  
**subgoal**  
**unfolding** *DBM-zone-repr-def* **using** *assms* **by** (*auto intro: dbm-abstra-soundness*)  
**done**

**lemma** [*simp*]:  
 $u \vdash \square$   
**by** (*force simp: clock-val-def*)

**lemma** *clock-val-Cons*:  
**assumes**  $u \vdash_a ac\ u \vdash cc$   
**shows**  $u \vdash (ac \# cc)$   
**using** *assms* **by** (*induction cc*) (*auto simp: clock-val-def*)

**lemma** *abstra-commute*:  
 $abstra\ ac1\ (abstra\ ac2\ M\ v)\ v = abstra\ ac2\ (abstra\ ac1\ M\ v)\ v$   
**by** (*cases ac1; cases ac2; fastforce simp: min.commute min.left-commute clock-val-def*)

**lemma** *dbm-abstr-completeness-aux*:  
 $\llbracket DBM\text{-val-bounded } v\ u\ (abstr\ cc\ (abstra\ ac\ M\ v)\ v)\ n; \forall c. v\ c > 0; v\ (constraint\text{-clk}\ ac) \leq n \rrbracket$   
 $\implies u \vdash_a ac$   
**apply** (*induction cc arbitrary: M*)

**apply** (*auto intro: dbm-abstra-completeness; fail*)  
**apply** *simp*  
**apply** (*subst (asm) abstra-commute*)  
**by** *auto*

**lemma** *dbm-abstr-completeness:*

$\llbracket \text{DBM-val-bounded } v \ u \ (\text{abstr } cc \ M \ v) \ n; \forall c. v \ c > 0; \forall c \in \text{collect-clks} \ cc. v \ c \leq n \rrbracket$   
 $\implies u \vdash cc$

**apply** (*induction cc arbitrary: M*)  
**apply** (*simp; fail*)  
**apply** (*rule clock-val-Cons*)  
**apply** (*rule dbm-abstr-completeness-aux*)  
**by** *auto*

**lemma** *dbm-abstr-zone-eq:*

**assumes** *clock-numbering' v n  $\forall c \in \text{collect-clks } cc. v \ c \leq n$*   
**shows**  $[\text{abstr } cc \ (\lambda i \ j. \ \infty) \ v]_{v,n} = \{u. u \vdash cc\}$   
**using** *dbm-abstr-soundness dbm-abstr-completeness assms* **unfolding** *DBM-zone-repr-def*  
**by** *metis*

**lemma** *dbm-abstr-zone-eq2:*

**assumes** *clock-numbering' v n  $\forall c \in \text{collect-clks } cc. v \ c \leq n$*   
**shows**  $[\text{abstr } cc \ M \ v]_{v,n} = [M]_{v,n} \cap \{u. u \vdash cc\}$   
**apply** *standard*  
**apply** (*rule Int-greatest*)  
**apply** (*rule abstr-subset*)  
**unfolding** *DBM-zone-repr-def*  
**apply** *safe*  
**apply** (*rule dbm-abstr-completeness*)  
**using** *assms* **apply** *auto[3]*  
**apply** (*rule dbm-abstr-soundness'*)  
**using** *assms* **by** *auto*

**abbreviation** *global-clock-numbering* ::

$('a, 'c, 't, 's) \ ta \Rightarrow ('c \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{bool}$

**where**

$\text{global-clock-numbering } A \ v \ n \equiv$   
 $\text{clock-numbering}' \ v \ n \wedge (\forall c \in \text{clk-set } A. v \ c \leq n) \wedge (\forall k \leq n. k > 0 \longrightarrow$   
 $(\exists c. v \ c = k))$

**lemma** *dbm-int-all-abstra:*

**assumes** *dbm-int-all M snd (constraint-pair ac)  $\in \mathbb{Z}$*

**shows**  $dbm\text{-int}\text{-all}$  ( $abstra$   $ac$   $M$   $v$ )  
**using**  $assms$  **by** ( $cases$   $ac$ ) ( $auto$   $split$ :  $split\text{-min}$ )

**lemma**  $dbm\text{-int}\text{-all}\text{-abstr}$ :  
**assumes**  $dbm\text{-int}\text{-all}$   $M$   $\forall (x, m) \in collect\text{-clock}\text{-pairs}$   $g$ .  $m \in \mathbb{Z}$   
**shows**  $dbm\text{-int}\text{-all}$  ( $abstr$   $g$   $M$   $v$ )  
**using**  $assms$   
**proof** ( $induction$   $g$   $arbitrary$ :  $M$ )  
**case**  $Nil$   
**then show**  $?case$  **by**  $auto$   
**next**  
**case** ( $Cons$   $ac$   $cc$ )  
**from**  $Cons.IH[OF\ dbm\text{-int}\text{-all}\text{-abstra}, OF\ Cons.prem\{1\}]$   $Cons.prem\{2\}$   
**have**  
 $dbm\text{-int}\text{-all}$  ( $abstr$   $cc$  ( $abstra$   $ac$   $M$   $v$ )  $v$ )  
**unfolding**  $collect\text{-clock}\text{-pairs}\text{-def}$  **by**  $force$   
**then show**  $?case$  **by**  $auto$   
**qed**

**lemma**  $dbm\text{-int}\text{-all}\text{-abstr}'$ :  
**assumes**  $\forall (x, m) \in collect\text{-clock}\text{-pairs}$   $g$ .  $m \in \mathbb{Z}$   
**shows**  $dbm\text{-int}\text{-all}$  ( $abstr$   $g$  ( $\lambda i$   $j$ .  $\infty$ )  $v$ )  
**apply** ( $rule\ dbm\text{-int}\text{-all}\text{-abstr}$ )  
**using**  $assms$  **by**  $auto$

**lemma**  $dbm\text{-int}\text{-all}\text{-inv}\text{-abstr}$ :  
**assumes**  $\forall (x, m) \in clkp\text{-set}$   $A$ .  $m \in \mathbb{N}$   
**shows**  $dbm\text{-int}\text{-all}$  ( $abstr$  ( $inv\text{-of}$   $A$   $l$ ) ( $\lambda i$   $j$ .  $\infty$ )  $v$ )  
**proof** –  
**from**  $assms$  **have**  $\forall (x, m) \in collect\text{-clock}\text{-pairs}$  ( $inv\text{-of}$   $A$   $l$ ).  $m \in \mathbb{Z}$   
**unfolding**  $clkp\text{-set}\text{-def}$   $collect\text{-clki}\text{-def}$   $inv\text{-of}\text{-def}$  **using**  $Nats\text{-subset}\text{-Ints}$   
**by**  $auto$   
**from**  $dbm\text{-int}\text{-all}\text{-abstr}'[OF\ this]$  **show**  $?thesis$  .  
**qed**

**lemma**  $dbm\text{-int}\text{-all}\text{-guard}\text{-abstr}$ :  
**assumes**  $\forall (x, m) \in clkp\text{-set}$   $A$ .  $m \in \mathbb{N}$   $A \vdash l \xrightarrow{g, a, r} l'$   
**shows**  $dbm\text{-int}\text{-all}$  ( $abstr$   $g$  ( $\lambda i$   $j$ .  $\infty$ )  $v$ )  
**proof** –  
**from**  $assms$  **have**  $\forall (x, m) \in collect\text{-clock}\text{-pairs}$   $g$ .  $m \in \mathbb{Z}$   
**unfolding**  $clkp\text{-set}\text{-def}$   $collect\text{-clkt}\text{-def}$  **using**  $assms\{2\}$   $Nats\text{-subset}\text{-Ints}$   
**by**  $fastforce$   
**from**  $dbm\text{-int}\text{-all}\text{-abstr}'[OF\ this]$  **show**  $?thesis$  .  
**qed**

**lemma** *dbm-int-abstra*:

**assumes** *dbm-int*  $M$   $n$  *snd* (*constraint-pair*  $ac$ )  $\in \mathbb{Z}$   
**shows** *dbm-int* (*abstra*  $ac$   $M$   $v$ )  $n$   
**using** *assms* **by** (*cases*  $ac$ ) (*auto split: split-min*)

**lemma** *dbm-int-abstr*:

**assumes** *dbm-int*  $M$   $n$   $\forall (x, m) \in \text{collect-clock-pairs } g. m \in \mathbb{Z}$   
**shows** *dbm-int* (*abstr*  $g$   $M$   $v$ )  $n$   
**using** *assms*  
**proof** (*induction*  $g$  *arbitrary: M*)  
**case** *Nil*  
**then show** *?case* **by** *auto*  
**next**  
**case** (*Cons*  $ac$   $cc$ )  
**from** *Cons.IH*[*OF dbm-int-abstra, OF Cons.prem*s(1)] *Cons.prem*s(2-)  
**have**  
*dbm-int* (*abstr*  $cc$  (*abstra*  $ac$   $M$   $v$ )  $v$ )  $n$   
**unfolding** *collect-clock-pairs-def* **by** *force*  
**then show** *?case* **by** *auto*  
**qed**

**lemma** *dbm-int-abstr'*:

**assumes**  $\forall (x, m) \in \text{collect-clock-pairs } g. m \in \mathbb{Z}$   
**shows** *dbm-int* (*abstr*  $g$  ( $\lambda i j. \infty$ )  $v$ )  $n$   
**apply** (*rule dbm-int-abstr*)  
**using** *assms* **by** *auto*

**lemma** *int-zone-dbm*:

**assumes** *clock-numbering'*  $v$   $n$   
 $\forall (-, d) \in \text{collect-clock-pairs } cc. d \in \mathbb{Z} \forall c \in \text{collect-clks } cc. v c \leq n$   
**obtains**  $M$  **where**  $\{u. u \vdash cc\} = [M]_{v, n}$   
**and**  $\forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z}$   
**proof** –  
**let**  $?M = \text{abstr } cc (\lambda i j. \infty) v$   
**from** *assms*(2) **have**  $\forall i \leq n. \forall j \leq n. ?M i j \neq \infty \longrightarrow \text{get-const } (?M i j) \in \mathbb{Z}$   
**by** (*rule dbm-int-abstr'*)  
**with** *dbm-abstr-zone-eq*[*OF assms*(1) *assms*(3)] **show** *?thesis* **by** (*auto intro: that*)  
**qed**

**lemma** *dbm-int-inv-abstr*:

**assumes**  $\forall (x, m) \in \text{clkp-set } A. m \in \mathbb{N}$

**shows** *dbm-int* (*abstr* (*inv-of* *A l*) ( $\lambda i j. \infty$ ) *v*) *n*  
**proof** –  
**from** *assms* **have**  $\forall (x, m) \in \text{collect-clock-pairs } (inv\text{-of } A\ l). m \in \mathbb{Z}$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **using** *Nats-subset-Ints*  
**by** *auto*  
**from** *dbm-int-abstr'*[*OF this*] **show** *?thesis* .  
**qed**

**lemma** *dbm-int-guard-abstr*:  
**assumes**  $\forall (x, m) \in \text{clkp-set } A. m \in \mathbb{N} \ A \vdash l \longrightarrow^{g,a,r} l'$   
**shows** *dbm-int* (*abstr* *g* ( $\lambda i j. \infty$ ) *v*) *n*  
**proof** –  
**from** *assms* **have**  $\forall (x, m) \in \text{collect-clock-pairs } g. m \in \mathbb{Z}$   
**unfolding** *clkp-set-def collect-clkt-def* **using** *assms(2) Nats-subset-Ints*  
**by** *fastforce*  
**from** *dbm-int-abstr'*[*OF this*] **show** *?thesis* .  
**qed**

**lemma** *collect-clks-id*: *collect-clks cc = fst ' collect-clock-pairs cc*  
**proof** –  
**have** *constraint-clk ac = fst (constraint-pair ac)* **for** *ac* **by** (*cases ac*) *auto*  
**then show** *?thesis* **unfolding** *collect-clks-def collect-clock-pairs-def* **by**  
*auto*  
**qed**

**end**

## 3.6 Semantics Based on DBMs

**theory** *DBM-Zone-Semantics*  
**imports** *TA-DBM-Operations*  
**begin**

**no-notation** *infinity* ( $\langle \infty \rangle$ )  
**hide-const** (*open*) *D*

### 3.6.1 Single Step

**inductive** *step-z-dbm* ::  
 $(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow 't :: \{\text{linordered-cancel-ab-monoid-add, uminus}\}$   
*DBM*  
 $\Rightarrow (\ 'c \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow 'a \text{ action} \Rightarrow 's \Rightarrow 't \text{ DBM} \Rightarrow \text{bool}$   
 $(\ \langle - \vdash \langle -, - \rangle \rightsquigarrow -, -, \langle -, - \rangle \rangle [61, 61, 61, 61] \ 61)$   
**where**

*step-t-z-dbm:*  
 $D\text{-inv} = \text{abstr } (\text{inv-of } A \ l) \ (\lambda i \ j. \ \infty) \ v \implies A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, \tau} \langle l, \text{And } (\text{up } D) \ D\text{-inv} \rangle \mid$   
*step-a-z-dbm:*  
 $A \vdash l \xrightarrow{g, a, r} l'$   
 $\implies A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, \mid a} \langle l', \text{And } (\text{reset}' (\text{And } D \ (\text{abstr } g \ (\lambda i \ j. \ \infty) \ v)) \ n \ r \ v \ 0) \ (\text{abstr } (\text{inv-of } A \ l') \ (\lambda i \ j. \ \infty) \ v) \rangle$

**inductive-cases** *step-z-t-cases:*  $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, \tau} \langle l', D' \rangle$   
**inductive-cases** *step-z-a-cases:*  $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, \mid a} \langle l', D' \rangle$   
**lemmas** *step-z-cases = step-z-a-cases step-z-t-cases*

**declare** *step-z-dbm.intros*[intro]

**lemma** *step-z-dbm-preserves-int-all:*  
**fixes**  $D \ D' :: ('t :: \{\text{time}, \text{ring-1}\} \text{DBM})$   
**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, a} \langle l', D' \rangle$  *global-clock-numbering*  $A \ v \ n \ \forall \ (x, m) \in \text{clk-set } A. \ m \in \mathbb{N}$   
*dbm-int-all*  $D$   
**shows** *dbm-int-all*  $D'$   
**using** *assms*  
**proof** (*cases, goal-cases*)  
**case** (1  $D''$ )  
**hence**  $\forall c \in \text{clk-set } A. \ v \ c \leq n$  **by** *blast+*  
**from** *dbm-int-all-inv-abstr*[OF 1(2)] 1 **have**  $D''\text{-int}: \text{dbm-int-all } D''$  **by** *simp*  
**show** *?thesis* **unfolding** 1(6)  
**by** (*intro And-int-all-preservation up-int-all-preservation dbm-int-inv-abstr*  $D''\text{-int}$  1)  
**next**  
**case** (2  $g \ a \ r$ )  
**hence** *assms: clock-numbering'*  $v \ n \ \forall c \in \text{clk-set } A. \ v \ c \leq n$   
**by** *blast+*  
**from** *dbm-int-all-inv-abstr*[OF 2(2)] **have**  $D'\text{-int}: \text{dbm-int-all } (\text{abstr } (\text{inv-of } A \ l') \ (\lambda i \ j. \ \infty) \ v)$   
**by** *simp*  
**from** *dbm-int-all-guard-abstr* 2 **have**  $D''\text{-int}: \text{dbm-int-all } (\text{abstr } g \ (\lambda i \ j. \ \infty) \ v)$  **by** *simp*  
**have**  $\text{set } r \subseteq \text{clk-set } A$  **using** 2(6) **unfolding** *trans-of-def collect-ckvt-def*  
**by** *fastforce*  
**hence**  $\forall c \in \text{set } r. \ v \ c \leq n$  **using** *assms*(2) **by** *fastforce*  
**show** *?thesis* **unfolding** 2(5)  
**by** (*intro And-int-all-preservation DBM-reset'-int-all-preservation dbm-int-all-inv-abstr* 2  $D''\text{-int}$ )

(*simp-all add: assms(1) \**)  
**qed**

**lemma** *step-z-dbm-preserves-int:*

**fixes**  $D D' :: ('t :: \{time, ring-1\} DBM)$   
**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, a} \langle l', D' \rangle$  *global-clock-numbering*  $A v n \forall (x, m)$   
 $\in \text{clkp-set } A. m \in \mathbb{N}$

*dbm-int*  $D n$

**shows** *dbm-int*  $D' n$

**using** *assms*

**proof** (*cases, goal-cases*)

**case** (1  $D''$ )

**from** *dbm-int-inv-abstr*[*OF 1(2)*] 1 **have**  $D''\text{-int}: \text{dbm-int } D'' n$  **by** *simp*

**show** *?thesis unfolding 1(6)*

**by** (*intro And-int-preservation up-int-preservation dbm-int-inv-abstr*  
 $D''\text{-int } 1$ )

**next**

**case** (2  $g a r$ )

**hence** *assms: clock-numbering' v n*  $\forall c \in \text{clk-set } A. v c \leq n$

**by** *blast+*

**from** *dbm-int-inv-abstr*[*OF 2(2)*] **have**  $D'\text{-int}: \text{dbm-int } (\text{abstr } (\text{inv-of } A$   
 $l') (\lambda i j. \infty) v) n$

**by** *simp*

**from** *dbm-int-guard-abstr 2* **have**  $D''\text{-int}: \text{dbm-int } (\text{abstr } g (\lambda i j. \infty) v)$   
 $n$  **by** *simp*

**have**  $\text{set } r \subseteq \text{clk-set } A$  **using** 2(6) **unfolding** *trans-of-def collect-ckvt-def*  
**by** *fastforce*

**hence**  $*\forall c \in \text{set } r. v c \leq n$  **using** *assms(2)* **by** *fastforce*

**show** *?thesis unfolding 2(5)*

**by** (*intro And-int-preservation DBM-reset'-int-preservation dbm-int-inv-abstr*  
 $2 D''\text{-int}$ )

(*simp-all add: assms(1) 2(2) \**)

**qed**

**lemma** *up-correct:*

**assumes** *clock-numbering' v n*

**shows**  $[\text{up } M]_{v, n} = [M]_{v, n}^\uparrow$

**using** *assms*

**apply** *safe*

**apply** (*rule DBM-up-sound'*)

**apply** *assumption+*

**apply** (*rule DBM-up-complete'*)

**apply** *auto*

**done**

**lemma** *step-z-dbm-sound*:

**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$  *global-clock-numbering*  $A \ v \ n$

**shows**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', [D']_{v,n} \rangle$

**using** *assms*

**proof** (*cases, goal-cases*)

**case** (1  $D''$ )

**hence** *clock-numbering'*  $v \ n \ \forall c \in \text{clk-set } A. \ v \ c \leq n$  **by** *blast+*

**note**  $\text{assms} = \text{assms}(1)$  *this*

**from**  $\text{assms}(3)$  **have**  $*$ :  $\forall c \in \text{collect-clks} (\text{inv-of } A \ l). \ v \ c \leq n$

**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** (*fastforce simp: collect-clks-id*)

**from** 1 **have**  $D''$ :  $[D'']_{v,n} = \{u. u \vdash \text{inv-of } A \ l\}$  **using** *dbm-abstr-zone-eq[OF assms(2) \*]* **by** *metis*

**with** *And-correct* **have**  $A11$ :  $[And \ D \ D'']_{v,n} = ([D]_{v,n}) \cap (\{u. u \vdash \text{inv-of } A \ l\})$  **by** *blast*

**from**  $D''$  **have**

$[D']_{v,n} = ([up \ D]_{v,n}) \cap (\{u. u \vdash \text{inv-of } A \ l\})$

**unfolding** 1(4) *And-correct[symmetric]* **by** *simp*

**with** *up-correct[OF assms(2)] A11* **have**  $[D']_{v,n} = ([D]_{v,n})^\dagger \cap \{u. u \vdash \text{inv-of } A \ l\}$  **by** *metis*

**then show** *?thesis* **by** (*auto simp: 1(2,3)*)

**next**

**case** (2  $g \ a \ r$ )

**hence** *clock-numbering'*  $v \ n \ \forall c \in \text{clk-set } A. \ v \ c \leq n \ \forall k \leq n. \ k > 0 \longrightarrow (\exists c. \ v \ c = k)$  **by** *blast+*

**note**  $\text{assms} = \text{assms}(1)$  *this*

**from**  $\text{assms}(3)$  **have**  $*$ :  $\forall c \in \text{collect-clks} (\text{inv-of } A \ l'). \ v \ c \leq n$

**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** (*fastforce simp: collect-clks-id*)

**have**  $D'$ :

$[abstr (\text{inv-of } A \ l') (\lambda i \ j. \ \infty) \ v]_{v,n} = \{u. u \vdash \text{inv-of } A \ l'\}$

**using** 2 *dbm-abstr-zone-eq[OF assms(2) \*]* **by** *simp*

**from**  $\text{assms}(3)$  2(4) **have**  $*$ :  $\forall c \in \text{collect-clks } g. \ v \ c \leq n$

**unfolding** *clkp-set-def collect-clkt-def inv-of-def* **by** (*fastforce simp: collect-clks-id*)

**have**  $D''$ :  $[abstr \ g \ (\lambda i \ j. \ \infty) \ v]_{v,n} = \{u. u \vdash g\}$  **using** 2 *dbm-abstr-zone-eq[OF assms(2) \*]* **by** *auto*

**with** *And-correct* **have**  $A11$ :  $[And \ D \ (abstr \ g \ (\lambda i \ j. \ \infty) \ v)]_{v,n} = ([D]_{v,n}) \cap (\{u. u \vdash g\})$  **by** *blast*

**let**  $?D = \text{reset}' (And \ D \ (abstr \ g \ (\lambda i \ j. \ \infty) \ v)) \ n \ r \ v \ 0$

**have**  $\text{set } r \subseteq \text{clk-set } A$  **using** 2(4) **unfolding** *trans-of-def collect-clkt-def* **by** *fastforce*

**hence**  $**$ :  $\forall c \in \text{set } r. \ v \ c \leq n$  **using**  $\text{assms}(3)$  **by** *fastforce*

**have**  $D$ -reset:  $[?D]_{v,n} = \text{zone-set } (([D]_{v,n}) \cap \{u. u \vdash g\})$   $r$   
**proof** *safe*  
**fix**  $u$  **assume**  $u: u \in [?D]_{v,n}$   
**from**  $DBM$ -reset'-sound[ $OF$   $assms(4,2)$  \*\*  $this$ ] **obtain**  $ts$  **where**  
 $set\text{-clocks } r$   $ts$   $u \in [And$   $D$  ( $abstr$   $g$  ( $\lambda i j. \infty$ )  $v$ )] $_{v,n}$   
**by** *auto*  
**with**  $A11$  **have** \*:  $set\text{-clocks } r$   $ts$   $u \in ([D]_{v,n}) \cap (\{u. u \vdash g\})$  **by** *blast*  
**from**  $DBM$ -reset'-resets[ $OF$   $assms(4,2)$  \*\*]  $u$   
**have**  $\forall c \in set$   $r. u$   $c = 0$  **unfolding**  $DBM$ -zone-repr-def **by** *auto*  
**from**  $reset\text{-set}[OF$   $this]$  **have**  $[r \rightarrow 0]$   $set\text{-clocks } r$   $ts$   $u = u$  **by** *simp*  
**with** \* **show**  $u \in \text{zone-set } (([D]_{v,n}) \cap \{u. u \vdash g\})$   $r$  **unfolding**  
 $zone\text{-set-def}$  **by** *force*  
**next**  
**fix**  $u$  **assume**  $u: u \in \text{zone-set } (([D]_{v,n}) \cap \{u. u \vdash g\})$   $r$   
**from**  $DBM$ -reset'-complete[ $OF$  -  $assms(2)$  \*\*]  $u$   $A11$   
**show**  $u \in [?D]_{v,n}$  **unfolding**  $DBM$ -zone-repr-def  $zone\text{-set-def}$  **by** *force*  
**qed**  
**from**  $D'$   $And$ -correct  $D$ -reset **have**  $A22$ :  
 $[And$   $?D$  ( $abstr$  ( $inv\text{-of } A$   $l'$ ) ( $\lambda i j. \infty$ )  $v$ )] $_{v,n} = ([?D]_{v,n}) \cap (\{u. u \vdash$   
 $inv\text{-of } A$   $l'\})$   
**by** *blast*  
**with**  $D$ -reset  $2(2-4)$  **show**  $?thesis$  **by** *auto*  
**qed**

**lemma**  $step\text{-z-dbm-DBM}$ :

**assumes**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', Z \rangle$   $global\text{-clock-numbering } A$   $v$   $n$   
**obtains**  $D'$  **where**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$   $Z = [D']_{v,n}$   
**using**  $assms$   
**proof** ( $cases$ ,  $goal\text{-cases}$ )  
**case**  $1$   
**hence**  $clock\text{-numbering}'$   $v$   $n$   $\forall c \in clk\text{-set } A. v$   $c \leq n$  **by**  $metis+$   
**note**  $assms = assms(1)$   $this$   
**from**  $assms(3)$  **have** \*:  $\forall c \in collect\text{-clks } (inv\text{-of } A$   $l). v$   $c \leq n$   
**unfolding**  $clkp\text{-set-def}$   $collect\text{-clki-def}$   $inv\text{-of-def}$  **by** ( $fastforce$   $simp$ :  $collect\text{-clks-id}$ )  
**obtain**  $D''$  **where**  $D''\text{-def}$ :  $D'' = abstr$  ( $inv\text{-of } A$   $l$ ) ( $\lambda i j. \infty$ )  $v$  **by** *auto*  
**hence**  $D''$ :  $[D'']_{v,n} = \{u. u \vdash inv\text{-of } A$   $l\}$  **using**  $dbm\text{-abstr-zone-eq}[OF$   
 $assms(2)$  \*] **by**  $metis$   
**obtain**  $D\text{-up}$  **where**  $D\text{-up}'$ :  $D\text{-up} = up$   $D$  **by** *blast*  
**with**  $up\text{-correct}$   $assms(2)$  **have**  $D\text{-up}$ :  $[D\text{-up}]_{v,n} = ([D]_{v,n})^\dagger$  **by**  $metis$   
**obtain**  $A2$  **where**  $A2$ :  $A2 = And$   $D\text{-up}$   $D''$  **by** *fast*  
**with**  $And\text{-correct}$   $D''$  **have**  $A22$ :  $[A2]_{v,n} = ([D\text{-up}]_{v,n}) \cap (\{u. u \vdash inv\text{-of}$   
 $A$   $l'\})$  **by** *blast*  
**have**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,\tau} \langle l, A2 \rangle$  **unfolding**  $A2$   $D\text{-up}'$   $D''\text{-def}$  **by** *blast*

**moreover have**  
 $[A2]_{v,n} = ([D]_{v,n})^\dagger \cap \{u. u \vdash \text{inv-of } A \ l\}$   
**unfolding** *A22 D-up ..*  
**ultimately show thesis using 1 by** (*intro that[of A2]*) *auto*  
**next**  
**case** (*2 g a r*)  
**hence** *clock-numbering' v n*  $\forall c \in \text{clk-set } A. v \ c \leq n \ \forall k \leq n. k > 0 \longrightarrow (\exists c. v \ c = k)$  **by** *metis+*  
**note** *assms = assms(1) this*  
**from** *assms(3) have \**:  $\forall c \in \text{collect-clks } (\text{inv-of } A \ l'). v \ c \leq n$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** (*fastforce simp: collect-clks-id*)  
**obtain** *D'* **where** *D'-def: D' = abstr (inv-of A l') (λi j. ∞) v* **by** *blast*  
**hence** *D':[D']<sub>v,n</sub> = {u. u ⊢ inv-of A l'}* **using** *dbm-abstr-zone-eq[OF assms(2) \*]* **by** *simp*  
**from** *assms(3) 2(5) have \**:  $\forall c \in \text{collect-clks } g. v \ c \leq n$   
**unfolding** *clkp-set-def collect-clkt-def inv-of-def* **by** (*fastforce simp: collect-clks-id*)  
**obtain** *D''* **where** *D''-def: D'' = abstr g (λi j. ∞) v* **by** *blast*  
**hence** *D'':[D']<sub>v,n</sub> = {u. u ⊢ g}* **using** *dbm-abstr-zone-eq[OF assms(2) \*]*  
**by** *auto*  
**obtain** *A1* **where** *A1: A1 = And D D''* **by** *fast*  
**with** *And-correct D''* **have** *A11: [A1]<sub>v,n</sub> = ([D]<sub>v,n</sub>) ∩ ({u. u ⊢ g})* **by** *blast*  
**let** *?D = reset' A1 n r v 0*  
**have** *set r ⊆ clk-set A* **using** *2(5) unfolding trans-of-def collect-clkvt-def*  
**by** *fastforce*  
**hence** *\*\*:*  $\forall c \in \text{set } r. v \ c \leq n$  **using** *assms(3) by fastforce*  
**have** *D-reset: [?D]<sub>v,n</sub> = zone-set (([D]<sub>v,n</sub>) ∩ {u. u ⊢ g}) r*  
**proof safe**  
**fix** *u* **assume** *u: u ∈ [?D]<sub>v,n</sub>*  
**from** *DBM-reset'-sound[OF assms(4,2) \*\* this]* **obtain** *ts* **where**  
*set-clocks r ts u ∈ [A1]<sub>v,n</sub>*  
**by** *auto*  
**with** *A11* **have** *\**: *set-clocks r ts u ∈ ([D]<sub>v,n</sub>) ∩ ({u. u ⊢ g})* **by** *blast*  
**from** *DBM-reset'-resets[OF assms(4,2) \*\*] u*  
**have**  $\forall c \in \text{set } r. u \ c = 0$  **unfolding** *DBM-zone-repr-def* **by** *auto*  
**from** *reset-set[OF this]* **have** *[r→0]set-clocks r ts u = u* **by** *simp*  
**with** *\** **show** *u ∈ zone-set (([D]<sub>v,n</sub>) ∩ {u. u ⊢ g}) r* **unfolding**  
*zone-set-def* **by** *force*  
**next**  
**fix** *u* **assume** *u: u ∈ zone-set (([D]<sub>v,n</sub>) ∩ {u. u ⊢ g}) r*  
**from** *DBM-reset'-complete[OF - assms(2) \*\*] u A11*  
**show** *u ∈ [?D]<sub>v,n</sub>* **unfolding** *DBM-zone-repr-def zone-set-def* **by** *force*

**qed**  
**obtain**  $A2$  **where**  $A2$ :  $A2 = \text{And } ?D D'$  **by** *fast*  
**with** *And-correct*  $D'$  **have**  $A22$ :  $[A2]_{v,n} = ([?D]_{v,n}) \cap (\{u. u \vdash \text{inv-of } A l'\})$  **by** *blast*  
**from**  $2(5)$   $A2$   $D'$ -def  $D''$ -def  $A1$  **have**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,1a} \langle l', A2 \rangle$  **by** *blast*  
**moreover from**  $A22$   $D$ -reset **have**  
 $[A2]_{v,n} = \text{zone-set } (([D]_{v,n}) \cap \{u. u \vdash g\}) r \cap \{u. u \vdash \text{inv-of } A l'\}$   
**by** *auto*  
**ultimately show** *?thesis* **using**  $2$  **by** (*intro that[of A2]*) *simp+*  
**qed**

**lemma** *step-z-computable*:

**assumes**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', Z \rangle$  *global-clock-numbering*  $A$   $v$   $n$   
**obtains**  $D'$  **where**  $Z = [D']_{v,n}$   
**using** *step-z-dbm-DBM[OF assms]* **by** *blast*

**lemma** *step-z-dbm-complete*:

**assumes** *global-clock-numbering*  $A$   $v$   $n$   $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$   
**and**  $u \in [(D)]_{v,n}$   
**shows**  $\exists D' a. A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle \wedge u' \in [D']_{v,n}$

**proof** –

**note**  $A = \text{assms}$   
**from** *step-z-complete[OF A(2,3)]* **obtain**  $Z' a$  **where**  $Z'$ :  
 $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', Z' \rangle$   $u' \in Z'$  **by** *auto*  
**with** *step-z-dbm-DBM[OF Z'(1) A(1)]* **obtain**  $D'$  **where**  $D'$ :  
 $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$   $Z' = [D']_{v,n}$   
**by** *metis*  
**with**  $Z'(2)$  **show** *?thesis* **by** *auto*  
**qed**

### 3.6.2 Additional Useful Properties

**lemma** *step-z-equiv*:

**assumes** *global-clock-numbering*  $A$   $v$   $n$   $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', Z \rangle$   $[D]_{v,n} = [M]_{v,n}$   
**shows**  $A \vdash \langle l, [M]_{v,n} \rangle \rightsquigarrow_a \langle l', Z \rangle$   
**using** *step-z-dbm-complete[OF assms(1)]* *step-z-dbm-sound[OF - assms(1), THEN step-z-sound]*  
*assms(2,3)* **by** *force*

**lemma** *step-z-dbm-equiv*:

**assumes** *global-clock-numbering*  $A$   $v$   $n$   $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$   $[D]_{v,n} = [M]_{v,n}$   
**shows**  $\exists M'. A \vdash \langle l, M \rangle \rightsquigarrow_{v,n,a} \langle l', M' \rangle \wedge [D']_{v,n} = [M']_{v,n}$

**proof** –

**from** *step-z-dbm-sound*[*OF assms*(2,1)] **have**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', [D']_{v,n} \rangle$  .

**with** *step-z-equiv*[*OF assms*(1) *this assms*(3)] **have**  $A \vdash \langle l, [M]_{v,n} \rangle \rightsquigarrow_a \langle l', [D']_{v,n} \rangle$  **by** *auto*

**from** *step-z-dbm-DBM*[*OF this assms*(1)] **show** *?thesis* **by** *auto*  
**qed**

**lemma** *step-z-empty*:

**assumes**  $A \vdash \langle l, \{\} \rangle \rightsquigarrow_a \langle l', Z \rangle$

**shows**  $Z = \{\}$

**using** *step-z-sound*[*OF assms*] **by** *auto*

**lemma** *step-z-dbm-empty*:

**assumes** *global-clock-numbering*  $A \ v \ n \ A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle [D]_{v,n} = \{\}$

**shows**  $[D']_{v,n} = \{\}$

**using** *step-z-dbm-sound*[*OF assms*(2,1)] *assms*(3) **by** – (*rule step-z-empty, auto*)

**end**

**theory** *Regions-Beta*

**imports**

*TA-Misc*

*Difference-Bound-Matrices.DBM-Normalization*

*Difference-Bound-Matrices.DBM-Operations*

*Difference-Bound-Matrices.Zones*

**begin**

## 4 Refinement to $\beta$ -regions

### 4.1 Definition

**type-synonym** *'c ceiling* = (*'c*  $\Rightarrow$  *nat*)

**datatype** *intv* =

*Const nat* |

*Intv nat* |

*Greater nat*

**datatype** *intv'* =

*Const' int* |

*Intv' int* |

*Greater' int* |

*Smaller' int*

**type-synonym**  $t = \text{real}$

**inductive**  $\text{valid-intv} :: \text{nat} \Rightarrow \text{intv} \Rightarrow \text{bool}$

**where**

$0 \leq d \Longrightarrow d \leq c \Longrightarrow \text{valid-intv } c \text{ (Const } d) \mid$

$0 \leq d \Longrightarrow d < c \Longrightarrow \text{valid-intv } c \text{ (Intv } d) \mid$

$\text{valid-intv } c \text{ (Greater } c)$

**inductive**  $\text{valid-intv}' :: \text{int} \Rightarrow \text{int} \Rightarrow \text{intv}' \Rightarrow \text{bool}$

**where**

$\text{valid-intv}' l \text{ - (Smaller' (-l))} \mid$

$-l \leq d \Longrightarrow d \leq u \Longrightarrow \text{valid-intv}' l u \text{ (Const' } d) \mid$

$-l \leq d \Longrightarrow d < u \Longrightarrow \text{valid-intv}' l u \text{ (Intv' } d) \mid$

$\text{valid-intv}' - u \text{ (Greater' } u)$

**inductive**  $\text{intv-elem} :: 'c \Rightarrow ('c, t) \text{ cval} \Rightarrow \text{intv} \Rightarrow \text{bool}$

**where**

$u x = d \Longrightarrow \text{intv-elem } x u \text{ (Const } d) \mid$

$d < u x \Longrightarrow u x < d + 1 \Longrightarrow \text{intv-elem } x u \text{ (Intv } d) \mid$

$c < u x \Longrightarrow \text{intv-elem } x u \text{ (Greater } c)$

**inductive**  $\text{intv}'\text{-elem} :: 'c \Rightarrow 'c \Rightarrow ('c, t) \text{ cval} \Rightarrow \text{intv}' \Rightarrow \text{bool}$

**where**

$u x - u y < c \Longrightarrow \text{intv}'\text{-elem } x y u \text{ (Smaller' } c) \mid$

$u x - u y = d \Longrightarrow \text{intv}'\text{-elem } x y u \text{ (Const' } d) \mid$

$d < u x - u y \Longrightarrow u x - u y < d + 1 \Longrightarrow \text{intv}'\text{-elem } x y u \text{ (Intv' } d) \mid$

$c < u x - u y \Longrightarrow \text{intv}'\text{-elem } x y u \text{ (Greater' } c)$

**abbreviation**  $\text{total-preorder } r \equiv \text{refl } r \wedge \text{trans } r$

**inductive**  $\text{isConst} :: \text{intv} \Rightarrow \text{bool}$

**where**

$\text{isConst (Const -)}$

**inductive**  $\text{isIntv} :: \text{intv} \Rightarrow \text{bool}$

**where**

$\text{isIntv (Intv -)}$

**inductive**  $\text{isGreater} :: \text{intv} \Rightarrow \text{bool}$

**where**

$\text{isGreater (Greater -)}$

**declare** *isIntv.intros*[intro!] *isConst.intros*[intro!] *isGreater.intros*[intro!]

**declare** *isIntv.cases*[elim!] *isConst.cases*[elim!] *isGreater.cases*[elim!]

**inductive** *valid-region* :: '*c* set  $\Rightarrow$  ('*c*  $\Rightarrow$  nat)  $\Rightarrow$  ('*c*  $\Rightarrow$  intv)  $\Rightarrow$  ('*c*  $\Rightarrow$  '*c*  $\Rightarrow$  intv')  $\Rightarrow$  '*c* rel  $\Rightarrow$  bool

**where**

$\llbracket X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}; r \subseteq X_0 \times X_0; \text{refl-on } X_0 r; \text{trans } r;$   
 $\text{total-on } X_0 r;$   
 $\forall x \in X. \text{valid-intv } (k x) (I x);$   
 $\forall x \in X. \forall y \in X. \text{isGreater } (I x) \vee \text{isGreater } (I y) \longrightarrow \text{valid-intv}' (k y) (k x) (J x y)\rrbracket$   
 $\implies \text{valid-region } X k I J r$

**inductive-set** *region* for *X I J r*

**where**

$\forall x \in X. u x \geq 0 \implies \forall x \in X. \text{intv-elem } x u (I x) \implies X_0 = \{x \in X. \exists d. I x = \text{Intv } d\} \implies$   
 $\forall x \in X_0. \forall y \in X_0. (x, y) \in r \iff \text{frac } (u x) \leq \text{frac } (u y) \implies$   
 $\forall x \in X. \forall y \in X. \text{isGreater } (I x) \vee \text{isGreater } (I y) \longrightarrow \text{intv}'\text{-elem } x y u (J x y)$   
 $\implies u \in \text{region } X I J r$

Defining the unique element of a partition that contains a valuation

**definition** *part* ( $\langle[-]\rangle$  [61,61] 61) **where** *part* *v*  $\mathcal{R} \equiv \text{THE } R. R \in \mathcal{R} \wedge v \in R$

First we need to show that the set of regions is a partition of the set of all clock assignments. This property is only claimed by P. Bouyer.

**inductive-cases**[elim!]: *intv-elem* *x u* (*Const* *d*)  
**inductive-cases**[elim!]: *intv-elem* *x u* (*Intv* *d*)  
**inductive-cases**[elim!]: *intv-elem* *x u* (*Greater* *d*)  
**inductive-cases**[elim!]: *valid-intv* *c* (*Greater* *d*)  
**inductive-cases**[elim!]: *valid-intv* *c* (*Const* *d*)  
**inductive-cases**[elim!]: *valid-intv* *c* (*Intv* *d*)  
**inductive-cases**[elim!]: *intv'*-elem *x y u* (*Const'* *d*)  
**inductive-cases**[elim!]: *intv'*-elem *x y u* (*Intv'* *d*)  
**inductive-cases**[elim!]: *intv'*-elem *x y u* (*Greater'* *d*)  
**inductive-cases**[elim!]: *intv'*-elem *x y u* (*Smaller'* *d*)  
**inductive-cases**[elim!]: *valid-intv'* *l u* (*Greater'* *d*)  
**inductive-cases**[elim!]: *valid-intv'* *l u* (*Smaller'* *d*)  
**inductive-cases**[elim!]: *valid-intv'* *l u* (*Const'* *d*)  
**inductive-cases**[elim!]: *valid-intv'* *l u* (*Intv'* *d*)

```

declare valid-intv.intros[intro]
declare valid-intv'.intros[intro]
declare intv-elem.intros[intro]
declare intv'-elem.intros[intro]

declare region.cases[elim]
declare valid-region.cases[elim]

```

## 4.2 Basic Properties

First we show that all valid intervals are distinct

**lemma** *valid-intv-distinct*:

```

valid-intv c I  $\implies$  valid-intv c I'  $\implies$  intv-elem x u I  $\implies$  intv-elem x u I'
 $\implies I = I'$ 
by (cases I) (cases I', auto)+

```

**lemma** *valid-intv'-distinct*:

```

 $-c \leq d \implies$  valid-intv' c d I  $\implies$  valid-intv' c d I'  $\implies$  intv'-elem x y u I
 $\implies$  intv'-elem x y u I'
 $\implies I = I'$ 
by (cases I) (cases I', auto)+

```

From this we show that all valid regions are distinct

**lemma** *valid-regions-distinct*:

```

valid-region X k I J r  $\implies$  valid-region X k I' J' r'  $\implies v \in$  region X I J
 $\implies v \in$  region X I' J' r'
 $\implies$  region X I J r = region X I' J' r'

```

**proof** *goal-cases*

**case** 1

**note**  $A = 1$

{ **fix**  $x$  **assume**  $x: x \in X$

**with**  $A(1)$  **have** *valid-intv (k x) (I x)* **by** *auto*

**moreover from**  $A(2)$   $x$  **have** *valid-intv (k x) (I' x)* **by** *auto*

**moreover from**  $A(3)$   $x$  **have** *intv-elem x v (I x)* **by** *auto*

**moreover from**  $A(4)$   $x$  **have** *intv-elem x v (I' x)* **by** *auto*

**ultimately have**  $I x = I' x$  **using** *valid-intv-distinct* **by** *fastforce*

} **note**  $* = \text{this}$

{ **fix**  $x y$  **assume**  $x: x \in X$  **and**  $y: y \in X$  **and**  $B: \text{isGreater } (I x) \vee \text{isGreater } (I y)$

**with**  $*$  **have**  $C: \text{isGreater } (I' x) \vee \text{isGreater } (I' y)$  **by** *auto*

**from**  $A(1)$   $x y B$  **have** *valid-intv' (k y) (k x) (J x y)* **by** *fastforce*

**moreover from**  $A(2)$   $x y C$  **have** *valid-intv' (k y) (k x) (J' x y)* **by**

*fastforce*

**moreover from**  $A(3)$   $x y B$  **have** *intv'-elem x y v (J x y)* **by** *force*

```

    moreover from  $A(4)$   $x y C$  have  $intv\text{-}elem\ x\ y\ v\ (J'\ x\ y)$  by force
    moreover from  $x\ y\ valid\text{-}intv\text{'-}distinct$  have  $- int\ (k\ y) \leq int\ (k\ x)$ 
  by simp
    ultimately have  $J\ x\ y = J'\ x\ y$  by (blast intro: valid-intv'-distinct)
  } note ** = this
  from  $A$  show ?thesis
  proof (auto, goal-cases)
    case (1  $u$ )
    note  $A = this$ 
    { fix  $x$  assume  $x: x \in X$ 
      from  $A(5)$   $x$  have  $intv\text{-}elem\ x\ u\ (I\ x)$  by auto
      with  $*\ x$  have  $intv\text{-}elem\ x\ u\ (I'\ x)$  by auto
    }
    then have  $\forall x \in X. intv\text{-}elem\ x\ u\ (I'\ x)$  by auto
    note  $B = this$ 
    { fix  $x\ y$  assume  $x: x \in X$  and  $y: y \in X$  and  $B: isGreater\ (I'\ x) \vee$ 
       $isGreater\ (I'\ y)$ 
      with  $*$  have  $isGreater\ (I\ x) \vee isGreater\ (I\ y)$  by auto
      with  $x\ y\ A(5)$  have  $intv\text{'-}elem\ x\ y\ u\ (J\ x\ y)$  by force
      with  $**[OF\ x\ y\ \langle isGreater\ (I\ x) \vee \rightarrow \rangle]$  have  $intv\text{'-}elem\ x\ y\ u\ (J'\ x\ y)$ 
    }
  by simp
  } note  $C = this$ 
  let  $?X_0 = \{x \in X. \exists d. I'\ x = Intv\ d\}$ 
  { fix  $x\ y$  assume  $x: x \in ?X_0$  and  $y: y \in ?X_0$ 
    have  $(x, y) \in r' \longleftrightarrow frac\ (u\ x) \leq frac\ (u\ y)$ 
    proof
      assume  $frac\ (u\ x) \leq frac\ (u\ y)$ 
      with  $A(5)$   $x\ y\ *$  have  $(x, y) \in r$  by auto
      with  $A(3)$   $x\ y\ *$  have  $frac\ (v\ x) \leq frac\ (v\ y)$  by auto
      with  $A(4)$   $x\ y$  show  $(x, y) \in r'$  by auto
    next
      assume  $(x, y) \in r'$ 
      with  $A(4)$   $x\ y$  have  $frac\ (v\ x) \leq frac\ (v\ y)$  by auto
      with  $A(3)$   $x\ y\ *$  have  $(x, y) \in r$  by auto
      with  $A(5)$   $x\ y\ *$  show  $frac\ (u\ x) \leq frac\ (u\ y)$  by auto
    qed
  }
  then have  $*: \forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r' \longleftrightarrow frac\ (u\ x) \leq frac$ 
   $(u\ y)$  by auto
  from  $A(5)$  have  $\forall x \in X. 0 \leq u\ x$  by auto
  from region.intros[ $OF\ this\ B\ -\ *$ ]  $C$  show ?case by auto
  next
  case (2  $u$ )
  note  $A = this$ 

```

```

{ fix x assume x: x ∈ X
  from A(5) x have intv-elem x u (I' x) by auto
  with * x have intv-elem x u (I x) by auto
}
then have ∀ x ∈ X. intv-elem x u (I x) by auto
note B = this
{ fix x y assume x: x ∈ X and y: y ∈ X and B: isGreater (I x) ∨
isGreater (I y)
  with * have isGreater (I' x) ∨ isGreater (I' y) by auto
  with x y A(5) have intv'-elem x y u (J' x y) by force
  with **[OF x y <isGreater (I x) ∨ ->] have intv'-elem x y u (J x y)
by simp
} note C = this
let ?X0 = {x ∈ X. ∃ d. I x = Intv d}
{ fix x y assume x: x ∈ ?X0 and y: y ∈ ?X0
  have (x, y) ∈ r ⟷ frac (u x) ≤ frac (u y)
  proof
    assume frac (u x) ≤ frac (u y)
    with A(5) x y * have (x,y) ∈ r' by auto
    with A(4) x y * have frac (v x) ≤ frac (v y) by auto
    with A(3) x y show (x,y) ∈ r by auto
  next
    assume (x,y) ∈ r
    with A(3) x y have frac (v x) ≤ frac (v y) by auto
    with A(4) x y * have (x,y) ∈ r' by auto
    with A(5) x y * show frac (u x) ≤ frac (u y) by auto
  qed
}
then have *:∀ x ∈ ?X0. ∀ y ∈ ?X0. (x, y) ∈ r ⟷ frac (u x) ≤ frac
(u y) by auto
from A(5) have ∀ x ∈ X. 0 ≤ u x by auto
from region.intros[OF this B - *] C show ?case by auto
qed
qed

```

```

locale Beta-Regions =
  fixes X :: 'c set and k :: 'c ⇒ nat
  assumes finite: finite X
  assumes non-empty: X ≠ {}
begin

```

**definition**

```

 $\mathcal{R} \equiv \{region\ X\ I\ J\ r \mid I\ J\ r.\ valid-region\ X\ k\ I\ J\ r\}$ 

```

**definition**  $V :: ('c, t)$  *cval set* **where**

$$V \equiv \{v . \forall x \in X. v x \geq 0\}$$

**lemma**  $\mathcal{R}$ -regions-distinct:

$$\llbracket R \in \mathcal{R}; v \in R; R' \in \mathcal{R}; R \neq R' \rrbracket \implies v \notin R'$$

**unfolding**  $\mathcal{R}$ -def **using** valid-regions-distinct **by** blast

Secondly, we also need to show that every valuations belongs to a region which is part of the partition.

**definition**  $intv\text{-of} :: nat \Rightarrow t \Rightarrow intv$  **where**

$$\begin{aligned} intv\text{-of } c \ v &\equiv \\ &\text{if } (v > c) \text{ then } Greater \ c \\ &\text{else if } (\exists x :: nat. x = v) \text{ then } (Const \ (nat \ (floor \ v))) \\ &\text{else } (Intv \ (nat \ (floor \ v))) \end{aligned}$$

**definition**  $intv'\text{-of} :: int \Rightarrow int \Rightarrow t \Rightarrow intv'$  **where**

$$\begin{aligned} intv'\text{-of } l \ u \ v &\equiv \\ &\text{if } (v > u) \text{ then } Greater' \ u \\ &\text{else if } (v < l) \text{ then } Smaller' \ l \\ &\text{else if } (\exists x :: int. x = v) \text{ then } (Const' \ (floor \ v)) \\ &\text{else } (Intv' \ (floor \ v)) \end{aligned}$$

**lemma** region-cover:

$$\forall x \in X. v x \geq 0 \implies \exists R. R \in \mathcal{R} \wedge v \in R$$

**proof** (standard, standard)

**assume** *assm*:  $\forall x \in X. 0 \leq v x$

**let**  $?I = \lambda x. intv\text{-of} \ (k \ x) \ (v \ x)$

**let**  $?J = \lambda x \ y. intv'\text{-of} \ (-k \ y) \ (k \ x) \ (v \ x - v \ y)$

**let**  $?X_0 = \{x \in X. \exists d. ?I \ x = Intv \ d\}$

**let**  $?r = \{(x,y). x \in ?X_0 \wedge y \in ?X_0 \wedge frac \ (v \ x) \leq frac \ (v \ y)\}$

**{ fix**  $x \ y \ d$  **assume**  $A: x \in X \ y \in X$

**then have**  $intv'\text{-elem} \ x \ y \ v \ (intv'\text{-of} \ (-int \ (k \ y)) \ (int \ (k \ x)) \ (v \ x - v \ y))$  **unfolding**  $intv'\text{-of-def}$

**proof** (auto, goal-cases)

**case** (1 a)

**then have**  $\lfloor v \ x - v \ y \rfloor = v \ x - v \ y$  **by** (metis of-int-floor-cancel)

**then show**  $?case$  **by** auto

**next**

**case** 2

**then have**  $\lfloor v \ x - v \ y \rfloor < v \ x - v \ y$  **by** (meson eq-iff floor-eq-iff not-less)

**with** 2 **show**  $?case$  **by** auto

**qed**

**} note**  $intro = this$

```

show  $v \in \text{region } X \text{ ?}I \text{ ?}J \text{ ?}r$ 
proof (standard, auto simp: assem intro: intro, goal-cases)
  case (1  $x$ )
  thus ?case unfolding intv-of-def
  proof (auto, goal-cases)
    case (1  $a$ )
    note  $A = \text{this}$ 
    from  $A(2)$  have  $\lfloor v x \rfloor = v x$  by (metis floor-of-int of-int-of-nat-eq)
    with assem A(1) have  $v x = \text{real } (\text{nat } \lfloor v x \rfloor)$  by auto
    then show ?case by auto
  next
  case 2
  note  $A = \text{this}$ 
  from  $A(1,2)$  have  $\text{real } (\text{nat } \lfloor v x \rfloor) < v x$ 
  proof –
    have  $f1: 0 \leq v x$ 
    using assem 1 by blast
    have  $v x \neq \text{real-of-int } (\text{int } (\text{nat } \lfloor v x \rfloor))$ 
    by (metis (no-types) 2(2) of-int-of-nat-eq)
    then show ?thesis
    using  $f1$  by linarith
  qed
  moreover from assem have  $v x < \text{real } (\text{nat } (\lfloor v x \rfloor) + 1)$  by linarith
  ultimately show ?case by auto
  qed
qed
{ fix  $x y$  assume  $x \in X y \in X$ 
  then have valid-intv' (int (k y)) (int (k x)) (intv'-of (- int (k y)) (int (k x)) (v x - v y))
  unfolding intv'-of-def
  apply auto
  apply (metis floor-of-int le-floor-iff linorder-not-less of-int-minus of-int-of-nat-eq valid-intv'.simps)
  by (metis floor-less-iff less-eq-real-def not-less of-int-minus of-int-of-nat-eq valid-intv'.intros(3))
}
moreover
{ fix  $x$  assume  $x: x \in X$ 
  then have valid-intv (k x) (intv-of (k x) (v x))
  proof (auto simp: intv-of-def, goal-cases)
    case (1  $a$ )
    then show ?case
    by (intro valid-intv.intros(1)) (auto, linarith)
  next

```

```

    case 2
    then show ?case
    apply (intro valid-intv.intros(2))
    using assm floor-less-iff nat-less-iff by fastforce+
  qed
}
ultimately have valid-region X k ?I ?J ?r
by (intro valid-region.intros, auto simp: refl-on-def trans-def total-on-def)
then show region X ?I ?J ?r ∈ R unfolding R-def by auto
qed

```

**lemma** *region-cover-V*:  $v \in V \implies \exists R. R \in \mathcal{R} \wedge v \in R$  using *region-cover* unfolding *V-def* by *simp*

Note that we cannot show that every region is non-empty anymore. The problem are regions fixing differences between an 'infeasible' constant.

We can show that there is always exactly one region a valid valuation belongs to. Note that we do not need non-emptiness for that.

**lemma** *regions-partition*:

$\forall x \in X. 0 \leq v x \implies \exists! R \in \mathcal{R}. v \in R$

**proof** *goal-cases*

case 1

note  $A = \text{this}$

with *region-cover[OF ]* obtain  $R$  where  $R: R \in \mathcal{R} \wedge v \in R$  by *fastforce* moreover

{ fix  $R'$  assume  $R' \in \mathcal{R} \wedge v \in R'$

with  $R$  *valid-regions-distinct[OF - - -]* have  $R' = R$  unfolding *R-def* by *blast*

}

ultimately show *?thesis* by *auto*

qed

**lemma** *region-unique*:

$v \in R \implies R \in \mathcal{R} \implies [v]_{\mathcal{R}} = R$

**proof** *goal-cases*

case 1

note  $A = \text{this}$

from  $A$  obtain  $I J r$  where \*:

*valid-region X k I J r R = region X I J r v ∈ region X I J r*

by (auto simp: *R-def*)

from *this(3)* have  $\forall x \in X. 0 \leq v x$  by *auto*

from *theI'[OF regions-partition[OF this]]* obtain  $I' J' r'$  where

$v: \text{valid-region X k I' J' r' } [v]_{\mathcal{R}} = \text{region X I' J' r' } v \in \text{region X I' J' r'}$

**unfolding part-def  $\mathcal{R}$ -def by auto**  
**from** *valid-regions-distinct*[*OF* \*(1)  $v(1)$  \*(3)  $v(3)$ ]  $v(2)$  \*(2) **show** ?*case*  
**by auto**  
**qed**

**lemma regions-partition':**

$\forall x \in X. 0 \leq v x \implies \forall x \in X. 0 \leq v' x \implies v' \in [v]_{\mathcal{R}} \implies [v']_{\mathcal{R}} = [v]_{\mathcal{R}}$

**proof goal-cases**

**case 1**

**note**  $A = \text{this}$

**from** *theI'*[*OF* *regions-partition*[*OF*  $A(1)$ ]]  $A(3)$  **obtain**  $I J r$  **where**

$v$ : *valid-region*  $X k I J r$   $[v]_{\mathcal{R}} = \text{region } X I J r$   $v' \in \text{region } X I J r$

**unfolding part-def  $\mathcal{R}$ -def by blast**

**from** *theI'*[*OF* *regions-partition*[*OF*  $A(2)$ ]] **obtain**  $I' J' r'$  **where**

$v'$ : *valid-region*  $X k I' J' r'$   $[v']_{\mathcal{R}} = \text{region } X I' J' r'$   $v' \in \text{region } X I' J' r'$

**unfolding part-def  $\mathcal{R}$ -def by auto**

**from** *valid-regions-distinct*[*OF*  $v'(1)$   $v(1)$   $v'(3)$   $v(3)$ ]  $v(2)$   $v'(2)$  **show**

?*case* **by simp**

**qed**

**lemma regions-closed:**

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies [v \oplus t]_{\mathcal{R}} \in \mathcal{R}$

**proof goal-cases**

**case 1**

**note**  $A = \text{this}$

**then obtain**  $I J r$  **where**  $v \in \text{region } X I J r$  **unfolding  $\mathcal{R}$ -def by auto**

**from** *this*(1) **have**  $\forall x \in X. v x \geq 0$  **by auto**

**with**  $A(3)$  **have**  $\forall x \in X. (v \oplus t) x \geq 0$  **unfolding eval-add-def by simp**

**from** *regions-partition*[*OF* *this*] **obtain**  $R'$  **where**  $R' \in \mathcal{R}$   $(v \oplus t) \in R'$  **by auto**

**with** *region-unique*[*OF* *this*(2,1)] **show** ?*case* **by auto**

**qed**

**lemma regions-closed':**

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies (v \oplus t) \in [v \oplus t]_{\mathcal{R}}$

**proof goal-cases**

**case 1**

**note**  $A = \text{this}$

**then obtain**  $I J r$  **where**  $v \in \text{region } X I J r$  **unfolding  $\mathcal{R}$ -def by auto**

**from** *this*(1) **have**  $\forall x \in X. v x \geq 0$  **by auto**

**with**  $A(3)$  **have**  $\forall x \in X. (v \oplus t) x \geq 0$  **unfolding eval-add-def by simp**

**from** *regions-partition*[*OF this*] **obtain**  $R'$  **where**  $R' \in \mathcal{R}$   $(v \oplus t) \in R'$   
**by** *auto*  
**with** *region-unique*[*OF this(2,1)*] **show** *?case* **by** *auto*  
**qed**

**lemma** *valid-regions-I-cong*:

*valid-region*  $X k I J r \implies \forall x \in X. I x = I' x$

$\implies \forall x \in X. \forall y \in X. (isGreater (I x) \vee isGreater (I y)) \longrightarrow J x y = J' x y$

$\implies region X I J r = region X I' J' r \wedge valid-region X k I' J' r$

**proof** (*auto, goal-cases*)

**case** (1 *v*)

**note**  $A = this$

**then have** [*simp*]:

$\bigwedge x. x \in X \implies I' x = I x$

$\bigwedge x y. x \in X \implies y \in X \implies isGreater (I x) \vee isGreater (I y) \implies J x y = J' x y$

**by** *metis+*

**show** *?case*

**proof** (*standard, goal-cases*)

**case** 1 **from**  $A(4)$  **show** *?case* **by** *auto*

**next**

**case** 2 **from**  $A(4)$  **show** *?case* **by** *auto*

**next**

**case** 3 **show**  $\{x \in X. \exists d. I x = Intv d\} = \{x \in X. \exists d. I' x = Intv d\}$

**by** *auto*

**next**

**case** 4

**let**  $?X_0 = \{x \in X. \exists d. I x = Intv d\}$

**from**  $A(4)$  **show**  $\forall x \in ?X_0. \forall y \in ?X_0. ((x, y) \in r) = (frac (v x) \le frac (v y))$  **by** *auto*

**next**

**case** 5 **from**  $A(4)$  **show** *?case* **by** *force*

**qed**

**next**

**case** (2 *v*)

**note**  $A = this$

**then have** [*simp*]:

$\bigwedge x. x \in X \implies I' x = I x$

$\bigwedge x y. x \in X \implies y \in X \implies isGreater (I x) \vee isGreater (I y) \implies J x y = J' x y$

**by** *metis+*

**show** *?case*

**proof** (*standard, goal-cases*)

```

    case 1 from A(4) show ?case by auto
next
    case 2 from A(4) show ?case by auto
next
    case 3
    show {x ∈ X. ∃ d. I' x = Intv d} = {x ∈ X. ∃ d. I x = Intv d} by auto
next
    case 4
    let ?X0 = {x ∈ X. ∃ d. I' x = Intv d}
    from A(4) show ∀ x ∈ ?X0. ∀ y ∈ ?X0. ((x, y) ∈ r) = (frac (v x) ≤
frac (v y)) by auto
    next
    case 5 from A(4) show ?case by force
qed
next
    case 3
    note A = this
    then have [simp]:
      ∧ x. x ∈ X ⇒ I' x = I x
      ∧ x y. x ∈ X ⇒ y ∈ X ⇒ isGreater (I x) ∨ isGreater (I y) ⇒ J x
y = J' x y
    by metis+
    show ?case
    apply rule
      apply (subgoal-tac {x ∈ X. ∃ d. I x = Intv d} = {x ∈ X. ∃ d. I' x
= Intv d})
      apply assumption
    using A by force+
qed

fun intv-const :: intv ⇒ nat
where
  intv-const (Const d) = d |
  intv-const (Intv d) = d |
  intv-const (Greater d) = d

fun intv'-const :: intv' ⇒ int
where
  intv'-const (Smaller' d) = d |
  intv'-const (Const' d) = d |
  intv'-const (Intv' d) = d |
  intv'-const (Greater' d) = d

lemma finite-ℛ-aux:

```

**fixes**  $P A B$  **assumes**  $\text{finite } \{x. A x\} \text{ finite } \{x. B x\}$   
**shows**  $\text{finite } \{(I, J) \mid I J. P I J r \wedge A I \wedge B J\}$   
**using** *assms* **by** (*fastforce intro: pairwise-finiteI finite-ex-and1 finite-ex-and2*)

**lemma** *finite- $\mathcal{R}$* :

**notes** [*simproc add: finite-Collect*]

**shows** *finite  $\mathcal{R}$*

**proof** –

**{** **fix**  $I J r$  **assume**  $A: \text{valid-region } X k I J r$   
**let**  $?X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}$   
**from**  $A$  **have** *refl-on*  $?X_0 r$  **by** *auto*  
**from**  $A$  **have**  $r \subseteq X \times X$  **by** *auto*  
**then** **have**  $r \in \text{Pow } (X \times X)$  **by** *auto*  
**}**  
**then** **have**  $\{r. \exists I J. \text{valid-region } X k I J r\} \subseteq \text{Pow } (X \times X)$  **by** *auto*  
**from** *finite-subset[OF this]* **finite** **have**  $\text{fin}: \text{finite } \{r. \exists I J. \text{valid-region } X k I J r\}$  **by** *auto*  
**let**  $?u = \text{Max } \{k x \mid x. x \in X\}$   
**let**  $?l = - \text{Max } \{k x \mid x. x \in X\}$   
**let**  $?I = \{\text{intv}. \text{intv-const } \text{intv} \leq ?u\}$   
**let**  $?J = \{\text{intv}. ?l \leq \text{intv}'\text{-const } \text{intv} \wedge \text{intv}'\text{-const } \text{intv} \leq ?u\}$   
**let**  $?S = \{r. \exists I J. \text{valid-region } X k I J r\}$   
**let**  $?fin\text{-map}I = \lambda I. \forall x. (x \in X \longrightarrow I x \in ?I) \wedge (x \notin X \longrightarrow I x = \text{Const } 0)$   
**let**  $?fin\text{-map}J = \lambda J. \forall x. \forall y. (x \in X \wedge y \in X \longrightarrow J x y \in ?J) \wedge (x \notin X \longrightarrow J x y = \text{Const}' 0) \wedge (y \notin X \longrightarrow J x y = \text{Const}' 0)$   
**let**  $?R = \{\text{region } X I J r \mid I J r. \text{valid-region } X k I J r \wedge ?fin\text{-map}I I \wedge ?fin\text{-map}J J\}$   
**let**  $?f = \lambda r. \{\text{region } X I J r \mid I J. \text{valid-region } X k I J r \wedge ?fin\text{-map}I I \wedge ?fin\text{-map}J J\}$   
**let**  $?g = \lambda r. \{(I, J) \mid I J. \text{valid-region } X k I J r \wedge ?fin\text{-map}I I \wedge ?fin\text{-map}J J\}$   
**have**  $?I = (\text{Const } ' \{d. d \leq ?u\}) \cup (\text{Intv } ' \{d. d \leq ?u\}) \cup (\text{Greater } ' \{d. d \leq ?u\})$   
**by** *auto (case-tac x, auto)*  
**then** **have** *finite*  $?I$  **by** *auto*  
**from** *finite-set-of-finite-funs[OF <finite X> this]* **have**  $\text{fin}I: \text{finite } \{I. ?fin\text{-map}I I\}$  .  
**have**  $?J = (\text{Smaller}' ' \{d. ?l \leq d \wedge d \leq ?u\}) \cup (\text{Const}' ' \{d. ?l \leq d \wedge d \leq ?u\}) \cup (\text{Intv}' ' \{d. ?l \leq d \wedge d \leq ?u\}) \cup (\text{Greater}' ' \{d. ?l \leq d \wedge d \leq ?u\})$   
**by** *auto (case-tac x, auto)*

**then have** *finite ?J* **by** *auto*  
**from** *finite-set-of-finite-funs2[OF ‹finite X› ‹finite X› this]* **have** *finJ:*  
*finite {J. ?fin-mapJ J}* .  
**from** *finite-ℛ-aux[OF finI finJ, of valid-region X k]* **have**  $\forall r \in ?S.$  *finite*  
*(?g r)* **by** *simp*  
**moreover have**  $\forall r \in ?S.$  *?f r = (λ (I, J). region X I J r) ‘ ?g r* **by**  
*auto*  
**ultimately have**  $\forall r \in ?S.$  *finite (?f r)* **by** *auto*  
**moreover have**  $?ℛ = \bigcup (?f ‘ ?S)$  **by** *auto*  
**ultimately have** *finite ?ℛ* **using** *fin* **by** *auto*  
**moreover have**  $ℛ \subseteq ?ℛ$   
**proof**  
**fix** *R* **assume** *R: R ∈ ℛ*  
**then obtain** *I J r* **where** *I: R = region X I J r* *valid-region X k I J r*  
**unfolding** *ℛ-def* **by** *auto*  
**let** *?I = λ x. if x ∈ X then I x else Const 0*  
**let** *?J = λ x y. if x ∈ X ∧ y ∈ X ∧ (isGreater (I x) ∨ isGreater (I y))*  
*then J x y else Const' 0*  
**let** *?R = region X ?I ?J r*  
**from** *valid-regions-I-cong[OF I(2)] I* **have**  $*$ : *R = ?R* *valid-region X k*  
*?I ?J r* **by** *auto*  
**have**  $\forall x. x \notin X \longrightarrow ?I x = \text{Const } 0$  **by** *auto*  
**moreover have**  $\forall x. x \in X \longrightarrow \text{intv-const } (I x) \leq ?u$   
**proof** *auto*  
**fix** *x* **assume** *x: x ∈ X*  
**with** *I(2)* **have** *valid-intv (k x) (I x)* **by** *auto*  
**moreover from** *‹finite X› x* **have**  $k x \leq ?u$  **by** *(auto intro: Max-ge)*  
**ultimately show**  $\text{intv-const } (I x) \leq \text{Max } \{k x \mid x. x \in X\}$  **by** *(cases*  
*I x) auto*  
**qed**  
**ultimately have**  $*$ : *?fin-mapI ?I* **by** *auto*  
**have**  $\forall x y. x \notin X \longrightarrow ?J x y = \text{Const}' 0$  **by** *auto*  
**moreover have**  $\forall x y. y \notin X \longrightarrow ?J x y = \text{Const}' 0$  **by** *auto*  
**moreover have**  $\forall x. \forall y. x \in X \wedge y \in X \longrightarrow ?l \leq \text{intv}'\text{-const } (?J x y)$   
 $\wedge \text{intv}'\text{-const } (?J x y) \leq ?u$   
**proof** *clarify*  
**fix** *x y* **assume** *x: x ∈ X* **assume** *y: y ∈ X*  
**show**  $?l \leq \text{intv}'\text{-const } (?J x y) \wedge \text{intv}'\text{-const } (?J x y) \leq ?u$   
**proof** *(cases isGreater (I x) ∨ isGreater (I y))*  
**case** *True*  
**with** *x y I(2)* **have** *valid-intv' (k y) (k x) (J x y)* **by** *fastforce*  
**moreover from** *‹finite X› x* **have**  $k x \leq ?u$  **by** *(auto intro: Max-ge)*  
**moreover from** *‹finite X› y* **have**  $?l \leq -k y$  **by** *(auto intro: Max-ge)*  
**ultimately show** *?thesis* **by** *(cases J x y) auto*

**next**  
 case *False* **then show** *?thesis* **by** *auto*  
**qed**  
**qed**  
 ultimately **have** *?fin-mapJ ?J* **by** *auto*  
 with *\*\** **show**  $R \in ?\mathcal{R}$  **by** *blast*  
**qed**  
 ultimately **show** *finite*  $\mathcal{R}$  **by** (*blast intro: finite-subset*)  
**qed**  
**end**

### 4.3 Approximation with $\beta$ -regions

**locale** *Beta-Regions'* = *Beta-Regions* +  
**fixes**  $v\ n$  *not-in-X*  
**assumes** *clock-numbering*:  $\forall c. v\ c > 0 \wedge (\forall x. \forall y. v\ x \leq n \wedge v\ y \leq n \wedge v\ x = v\ y \longrightarrow x = y)$   
 $\forall k :: \text{nat} \leq n. k > 0 \longrightarrow (\exists c \in X. v\ c = k) \forall c \in X. v\ c \leq n$   
**assumes** *not-in-X*:  $\text{not-in-X} \notin X$   
**begin**

**definition**  $v' \equiv \lambda i. \text{if } 0 < i \wedge i \leq n \text{ then } (THE\ c. c \in X \wedge v\ c = i) \text{ else } \text{not-in-X}$

**lemma**  $v-v'$ :  
 $\forall c \in X. v'\ (v\ c) = c$   
**using** *clock-numbering unfolding v'-def* **by** *auto*

#### abbreviation

$vabstr\ (S :: ('a, t)\ \text{zone})\ M \equiv S = [M]_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M\ i\ j \neq \infty \longrightarrow \text{get-const}\ (M\ i\ j) \in \mathbf{Z})$

#### definition *normalized*:

$\text{normalized}\ M \equiv$   
 $(\forall i\ j. 0 < i \wedge i \leq n \wedge 0 < j \wedge j \leq n \wedge M\ i\ j \neq \infty \longrightarrow$   
 $Lt\ (-\ (\text{real}((k\ o\ v')\ j))) \leq M\ i\ j \wedge M\ i\ j \leq Le\ ((k\ o\ v')\ i))$   
 $\wedge (\forall i \leq n. i > 0 \longrightarrow (M\ i\ 0 \leq Le\ ((k\ o\ v')\ i) \vee M\ i\ 0 = \infty) \wedge Lt\ (-$   
 $((k\ o\ v')\ i)) \leq M\ 0\ i)$

#### definition *apx-def*:

$\text{Approx}_\beta\ Z \equiv \bigcap \{S. \exists U\ M. S = \bigcup U \wedge U \subseteq \mathcal{R} \wedge Z \subseteq S \wedge vabstr\ S\ M \wedge \text{normalized}\ M\}$

**definition**

*normalized'*  $M \equiv$   
 $(\forall i j. 0 < i \wedge i \leq n \wedge 0 < j \wedge j \leq n \wedge M i j \neq \infty \wedge i \neq j \longrightarrow$   
 $Lt (- (real((k o v') j))) \leq M i j \wedge M i j \leq Le ((k o v') i))$   
 $\wedge (\forall i \leq n. i > 0 \longrightarrow (M i 0 \leq Le ((k o v') i) \vee M i 0 = \infty) \wedge Lt (-$   
 $((k o v') i) \leq M 0 i)$

**lemma** *normalized'-normalized*:

**assumes**  $\forall i \leq n. M i i = 0$  *normalized'*  $M$   
**shows** *normalized*  $M$   
**using** *assms unfolding normalized'-def normalized*  
**apply** *auto*  
**apply** (*smt Lt-le-LeI neutral of-nat-0-le-iff Le-le-LeI*)  
**done**

**lemma** *normalized-normalized'*:

*normalized'*  $M$  **if** *normalized*  $M$   
**using** *that unfolding normalized'-def normalized by simp*

**lemma** *apx-min*:

$S = \bigcup U \implies U \subseteq \mathcal{R} \implies S = [M]_{v,n} \implies \forall i \leq n. \forall j \leq n. M i j \neq \infty$   
 $\longrightarrow get-const (M i j) \in \mathbb{Z}$   
 $\implies normalized M \implies Z \subseteq S \implies Approx_{\beta} Z \subseteq S$   
**unfolding** *apx-def by blast*

**lemma** *R-union*:  $\bigcup \mathcal{R} = V$  **using** *region-cover unfolding V-def R-def by auto***definition** *V-dbm* where

*V-dbm*  $\equiv \lambda i j. if i = 0 then Le 0 else \infty$

**lemma** *v-not-eq-0*:

$v c \neq 0$   
**using** *clock-numbering(1) by (metis not-less-zero)*

**lemma** *V-dbm-eq-V*:  $[V-dbm]_{v,n} = V$ 

**unfolding** *V-dbm-def V-def DBM-zone-repr-def DBM-val-bounded-def*  
**proof** (*(clarsimp; safe), goal-cases*)  
**case**  $(1 u c)$   
**with** *clock-numbering have dbm-entry-val u None (Some c) (Le 0) by auto*  
**then show** *?case by auto*  
**next**

**case** ( $\lambda u c$ )  
**with** *clock-numbering* **have**  $c \in X$  **by** *blast*  
**with**  $\lambda(1)$  **show** *?case* **by** *auto*  
**qed** (*auto simp: v-not-eq-0*)

**lemma** *V-dbm-int*:  
 $\forall i \leq n. \forall j \leq n. V\text{-dbm } i j \neq \infty \longrightarrow \text{get-const } (V\text{-dbm } i j) \in \mathbb{Z}$   
**unfolding** *V-dbm-def* **by** *auto*

**lemma** *normalized-V-dbm*:  
*normalized V-dbm*  
**unfolding** *V-dbm-def normalized less-eq dbm-le-def* **by** *auto*

**lemma** *all-dbm*:  $\exists M. \text{vabstr } (\bigcup \mathcal{R}) M \wedge \text{normalized } M$   
**using** *V-dbm-eq-V V-dbm-int normalized-V-dbm* **using**  *$\mathcal{R}$ -union* **by** *auto*

**lemma**  *$\mathcal{R}$ -int*:  
 $R \in \mathcal{R} \implies R' \in \mathcal{R} \implies R \neq R' \implies R \cap R' = \{\}$  **using**  *$\mathcal{R}$ -regions-distinct*  
**by** *blast*

**lemma** *aux1*:  
 $u \in R \implies R \in \mathcal{R} \implies U \subseteq \mathcal{R} \implies u \in \bigcup U \implies R \subseteq \bigcup U$  **using**  *$\mathcal{R}$ -int*  
**by** *blast*

**lemma** *aux2*:  $x \in \bigcap U \implies U \neq \{\} \implies \exists S \in U. x \in S$  **by** *blast*

**lemma** *aux2'*:  $x \in \bigcap U \implies U \neq \{\} \implies \forall S \in U. x \in S$  **by** *blast*

**lemma** *apx-subset*:  $Z \subseteq \text{Approx}_\beta Z$  **unfolding** *apx-def* **by** *auto*

**lemma** *aux3*:  
 $\forall X \in U. \forall Y \in U. X \cap Y \in U \implies S \subseteq U \implies S \neq \{\} \implies \text{finite } S$   
 $\implies \bigcap S \in U$

**proof** *goal-cases*

**case** *1*

**with** *finite-list* **obtain**  $l$  **where**  $\text{set } l = S$  **by** *blast*

**then show** *?thesis* **using** *1*

**proof** (*induction l arbitrary: S*)

**case** *Nil* **thus** *?case* **by** *auto*

**next**

**case** (*Cons x xs*)

**show** *?case*

**proof** (*cases set xs = \{\}*)

**case** *False*

**with** *Cons* **have**  $\bigcap (set\ xs) \in U$  **by** *auto*  
**with** *Cons.prem*<sub>s</sub>(1–3) **show** *?thesis* **by** *force*  
**next**  
**case** *True*  
**with** *Cons.prem*<sub>s</sub> **show** *?thesis* **by** *auto*  
**qed**  
**qed**  
**qed**

**lemma** *empty-zone-dbm*:

$\exists M :: t\ DBM. vabstr\ \{\} M \wedge normalized\ M \wedge (\forall k \leq n. M\ k\ k \leq Le\ 0)$

**proof** –

**from** *non-empty* **obtain** *c* **where**  $c: c \in X$  **by** *auto*  
**with** *clock-numbering* **have**  $c': v\ c > 0\ v\ c \leq n$  **by** *auto*  
**let**  $?M = \lambda i\ j. if\ i = v\ c \wedge j = 0 \vee i = j\ then\ Le\ 0::t\ else\ if\ i = 0 \wedge j = v\ c\ then\ Lt\ 0\ else\ \infty$   
**have**  $[?M]_{v,n} = \{\}$  **unfolding** *DBM-zone-repr-def* *DBM-val-bounded-def*  
**using**  $c'$  **by** *auto*  
**moreover** **have**  $\forall i \leq n. \forall j \leq n. ?M\ i\ j \neq \infty \longrightarrow get\ const\ (?M\ i\ j) \in \mathbb{Z}$   
**by** *auto*  
**moreover** **have** *normalized*  $?M$  **unfolding** *normalized less-eq dbm-le-def*  
**by** *auto*  
**ultimately** **show** *?thesis* **by** *auto*  
**qed**

**lemma** *DBM-set-diag*:

**assumes**  $[M]_{v,n} \neq \{\}$   
**shows**  $[M]_{v,n} = [(\lambda i\ j. if\ i = j\ then\ Le\ 0\ else\ M\ i\ j)]_{v,n}$   
**using** *non-empty-dbm-diag-set*[*OF* *clock-numbering*(1) *assms*] **unfolding**  
*neutral* **by** *auto*

**lemma** *apx-min'*:

$S = \bigcup U \implies U \subseteq \mathcal{R} \implies S = [M]_{v,n} \implies \forall i \leq n. \forall j \leq n. M\ i\ j \neq \infty$   
 $\longrightarrow get\ const\ (M\ i\ j) \in \mathbb{Z}$   
 $\implies normalized'\ M \implies Z \subseteq S \implies Approx_{\beta}\ Z \subseteq S$

**proof** (*cases*  $S = \{\}$ , *goal-cases*)

**case** 1  
**then** **show** *?thesis*  
**using** *empty-zone-dbm apx-min* **by** *metis*  
**next**  
**case** 2  
**let**  $?M = (\lambda i\ j. if\ i = j\ then\ Le\ 0\ else\ M\ i\ j)$   
**from** *DBM-set-diag* 2 **have**  $[M]_{v,n} = [?M]_{v,n}$   
**by** *blast*

**moreover from**  $\langle \text{normalized}' \rightarrow \rangle$  **have**  $\text{normalized } ?M$   
**by**  $(\text{intro normalized}'\text{-normalized}; \text{simp add: normalized}'\text{-def neutral})$   
**ultimately show**  $?thesis$   
**using 2 by**  $(\text{intro apx-min}[\text{where } M = ?M]) \text{ auto}$   
**qed**

**lemma** *valid-dbms-int*:

$\forall X \in \{S. \exists M. \text{vabstr } S \ M\}. \forall Y \in \{S. \exists M. \text{vabstr } S \ M\}. X \cap Y \in \{S. \exists M. \text{vabstr } S \ M\}$

**proof**  $(\text{auto}, \text{goal-cases})$

**case**  $(1 \ M1 \ M2)$

**obtain**  $M'$  **where**  $M': M' = \text{And } M1 \ M2$  **by** *fast*

**from**  $\text{DBM-and-sound1}[\text{OF}] \ \text{DBM-and-sound2}[\text{OF}] \ \text{DBM-and-complete}[\text{OF}]$   
 $]$

**have**  $[M1]_{v,n} \cap [M2]_{v,n} = [M']_{v,n}$  **unfolding**  $\text{DBM-zone-repr-def } M'$  **by**  
*auto*

**moreover from 1 have**  $\forall i \leq n. \forall j \leq n. M' \ i \ j \neq \infty \longrightarrow \text{get-const } (M' \ i \ j) \in \mathbb{Z}$

**unfolding**  $M'$  **by**  $(\text{auto split: split-min})$

**ultimately show**  $?case$  **by** *auto*

**qed**

**lemma** *split-min'*:

$P \ (\text{min } i \ j) = ((\text{min } i \ j = i \longrightarrow P \ i) \wedge (\text{min } i \ j = j \longrightarrow P \ j))$

**unfolding**  $\text{min-def}$  **by** *auto*

**lemma** *normalized-and-preservation*:

$\text{normalized } M1 \Longrightarrow \text{normalized } M2 \Longrightarrow \text{normalized } (\text{And } M1 \ M2)$

**unfolding**  $\text{normalized}$  **by**  $\text{safe } (\text{subst } \text{And.simps}, \text{split } \text{split-min}', \text{fast-force})+$

**lemma** *valid-dbms-int'*:

$\forall X \in \{S. \exists M. \text{vabstr } S \ M \wedge \text{normalized } M\}. \forall Y \in \{S. \exists M. \text{vabstr } S \ M \wedge \text{normalized } M\}$

$X \cap Y \in \{S. \exists M. \text{vabstr } S \ M \wedge \text{normalized } M\}$

**proof**  $(\text{auto}, \text{goal-cases})$

**case**  $(1 \ M1 \ M2)$

**obtain**  $M'$  **where**  $M': M' = \text{And } M1 \ M2$  **by** *fast*

**from**  $\text{DBM-and-sound1} \ \text{DBM-and-sound2} \ \text{DBM-and-complete}$

**have**  $[M1]_{v,n} \cap [M2]_{v,n} = [M']_{v,n}$  **unfolding**  $M'$   $\text{DBM-zone-repr-def}$  **by**  
*auto*

**moreover from**  $M' \ 1$  **have**  $\forall i \leq n. \forall j \leq n. M' \ i \ j \neq \infty \longrightarrow \text{get-const } (M' \ i \ j) \in \mathbb{Z}$

**by**  $(\text{auto split: split-min})$

moreover from *normalized-and-preservation*[*OF 1(2,4)*] have *normalized*  $M'$  **unfolding**  $M'$  .

ultimately show *?case* **by auto**  
**qed**

**lemma** *apx-in*:

$Z \subseteq V \implies \text{Approx}_\beta Z \in \{S. \exists U M. S = \bigcup U \wedge U \subseteq \mathcal{R} \wedge Z \subseteq S \wedge \text{vabstr } S M \wedge \text{normalized } M\}$

**proof** –

assume  $Z \subseteq V$

let  $?A = \{S. \exists U M. S = \bigcup U \wedge U \subseteq \mathcal{R} \wedge Z \subseteq S \wedge \text{vabstr } S M \wedge \text{normalized } M\}$

let  $?U = \{R \in \mathcal{R}. \forall S \in ?A. R \subseteq S\}$

have  $?A \subseteq \{S. \exists U. S = \bigcup U \wedge U \subseteq \mathcal{R}\}$  **by auto**

moreover from *finite- $\mathcal{R}$*  have *finite ...* **by auto**

ultimately have *finite*  $?A$  **by** (*auto intro: finite-subset*)

from *all-dbm* obtain  $M$  where  $M$ :

$\text{vabstr } (\bigcup \mathcal{R}) M$  *normalized*  $M$

**by auto**

with  $\langle \cdot \subseteq V \rangle$   *$\mathcal{R}$ -union[symmetric]* have  $V \in ?A$

**by safe** (*intro conjI exI; auto*)

then have  $?A \neq \{\}$  **by blast**

have  $?A \subseteq \{S. \exists M. \text{vabstr } S M \wedge \text{normalized } M\}$  **by auto**

with *aux3*[*OF valid-dbms-int'* this  $\langle ?A \neq \cdot \rangle \langle \text{finite } ?A \rangle$ ] have

$\bigcap ?A \in \{S. \exists M. \text{vabstr } S M \wedge \text{normalized } M\}$

**by blast**

then obtain  $M$  where  $*$ :  $\text{vabstr } (\text{Approx}_\beta Z) M$  *normalized*  $M$  **unfolding**  
*apx-def* **by auto**

have  $\bigcup ?U = \bigcap ?A$

**proof** (*safe, goal-cases*)

case 1

show *?case*

**proof** (*cases*  $Z = \{\}$ )

case *False*

then obtain  $v$  where  $v \in Z$  **by auto**

with *region-cover*  $\langle Z \subseteq V \rangle$  obtain  $R$  where  $R \in \mathcal{R}$   $v \in R$  **unfolding**

*V-def* **by blast**

with *aux1*[*OF this(2,1)*]  $\langle v \in Z \rangle$  have  $R \in ?U$  **by blast**

with 1 show *?thesis* **by blast**

next

case *True*

with *empty-zone-dbm* have  $\{\} \in ?A$  **by auto**

with 1(1,3) show *?thesis* **by blast**

**qed**

**next**  
**case**  $\langle 2 \ v \rangle$   
**from**  $aux2[OF \ 2 \ \langle ?A \neq - \rangle]$  **obtain**  $S$  **where**  $v \in S \ S \in ?A$  **by** *blast*  
**then obtain**  $R$  **where**  $v \in R \ R \in \mathcal{R}$  **by** *auto*  
**{ fix**  $S$  **assume**  $S \in ?A$   
**with**  $aux2'[OF \ 2 \ \langle ?A \neq - \rangle]$  **have**  $v \in S$  **by** *auto*  
**with**  $\langle S \in ?A \rangle$  **obtain**  $U \ M \ R'$  **where** \*:  
 $v \in R' \ R' \in \mathcal{R} \ S = \bigcup U \ U \subseteq \mathcal{R} \ vabstr \ S \ M \ Z \subseteq S$   
**by** *blast*  
**from**  $aux1[OF \ this(1,2,4)] \ *(3) \ \langle v \in S \rangle$  **have**  $R' \subseteq S$  **by** *blast*  
**moreover from**  $\mathcal{R}$ -regions-distinct[ $OF \ *(2,1) \ \langle R \in \mathcal{R} \rangle \ \langle v \in R \rangle$ ] **have**  
 $R' = R$  **by** *fast*  
**ultimately have**  $R \subseteq S$  **by** *fast*  
**}**  
**with**  $\langle R \in \mathcal{R} \rangle$  **have**  $R \in ?U$  **by** *auto*  
**with**  $\langle v \in R \rangle$  **show** *?case* **by** *auto*  
**qed**  
**then have**  $Approx_\beta \ Z = \bigcup ?U \ ?U \subseteq \mathcal{R} \ Z \subseteq Approx_\beta \ Z$  **unfolding**  
*apx-def* **by** *auto*  
**with** \* **show** *?thesis* **by** *blast*  
**qed**

**lemma** *apx-empty*:

$$Approx_\beta \ \{\} = \{\}$$

**unfolding** *apx-def* **using** *empty-zone-dbm* **by** *blast*

**end**

## 4.4 Computing $\beta$ -Approximation

### 4.4.1 Computation

**context** *Beta-Regions'*

**begin**

**lemma** *dbm-regions*:

$$vabstr \ S \ M \implies normalized' \ M \implies [M]_{v,n} \neq \{\} \implies [M]_{v,n} \subseteq V \implies \exists U \subseteq \mathcal{R}. \ S = \bigcup U$$

**proof** *goal-cases*

**case**  $A: 1$

**let**  $?U =$

$$\{R \in \mathcal{R}. \exists I \ J \ r. R = region \ X \ I \ J \ r \wedge valid-region \ X \ k \ I \ J \ r \wedge (\forall c \in X.$$

$$(\forall d. I \ c = Const \ d \implies M \ (v \ c) \ 0 \geq Le \ d \wedge M \ 0 \ (v \ c) \geq Le \ (-d))\}$$

$\wedge$   
 $(\forall d. I c = Intv d \longrightarrow M (v c) 0 \geq Lt (d + 1) \wedge M 0 (v c) \geq Lt$   
 $(-d)) \wedge$   
 $(I c = Greater (k c) \longrightarrow M (v c) 0 = \infty)$   
 $) \wedge$   
 $(\forall x \in X. \forall y \in X.$   
 $(\forall c d. I x = Intv c \wedge I y = Intv d \longrightarrow M (v x) (v y) \geq$   
 $(if (x, y) \in r then if (y, x) \in r then Le (c - d) else Lt (c - d)$   
 $else Lt (c - d + 1))) \wedge$   
 $(\forall c d. I x = Intv c \wedge I y = Intv d \longrightarrow M (v y) (v x) \geq$   
 $(if (y, x) \in r then if (x, y) \in r then Le (d - c) else Lt (d - c)$   
 $else Lt (d - c + 1))) \wedge$   
 $(\forall c d. I x = Const c \wedge I y = Const d \longrightarrow M (v x) (v y) \geq Le (c$   
 $- d)) \wedge$   
 $(\forall c d. I x = Const c \wedge I y = Const d \longrightarrow M (v y) (v x) \geq Le (d$   
 $- c)) \wedge$   
 $(\forall c d. I x = Intv c \wedge I y = Const d \longrightarrow M (v x) (v y) \geq Lt (c -$   
 $d + 1)) \wedge$   
 $(\forall c d. I x = Intv c \wedge I y = Const d \longrightarrow M (v y) (v x) \geq Lt (d -$   
 $c)) \wedge$   
 $(\forall c d. I x = Const c \wedge I y = Intv d \longrightarrow M (v x) (v y) \geq Lt (c -$   
 $d)) \wedge$   
 $(\forall c d. I x = Const c \wedge I y = Intv d \longrightarrow M (v y) (v x) \geq Lt (d -$   
 $c + 1)) \wedge$   
 $((isGreater (I x) \vee isGreater (I y)) \wedge J x y = Greater' (k x) \longrightarrow M$   
 $(v x) (v y) = \infty) \wedge$   
 $(\forall c. (isGreater (I x) \vee isGreater (I y)) \wedge J x y = Const' c$   
 $\longrightarrow M (v x) (v y) \geq Le c \wedge M (v y) (v x) \geq Le (- c)) \wedge$   
 $(\forall c. (isGreater (I x) \vee isGreater (I y)) \wedge J x y = Intv' c$   
 $\longrightarrow M (v x) (v y) \geq Lt (c + 1) \wedge M (v y) (v x) \geq Lt (- c))$   
 $)$   
 $\}$   
**have**  $\bigcup ?U = [M]_{v,n}$  **unfolding** *DBM-zone-repr-def* *DBM-val-bounded-def*  
**proof** (*standard, goal-cases*)  
**case 1**  
**show** *?case*  
**proof** (*auto, goal-cases*)  
**case 1**  
**from** *A(3)* **show**  $Le 0 \preceq M 0 0$  **unfolding** *DBM-zone-repr-def*  
*DBM-val-bounded-def* **by** *auto*  
**next**  
**case** (*2 u I J r c*)  
**note**  $B = this$   
**from** *B(6)* *clock-numbering* **have**  $c \in X$  **by** *blast*

```

with  $B(1)$   $v-v'$  have *:  $\text{intv-elem } c \ u \ (I \ c) \ v' \ (v \ c) = c$  by auto
from  $\text{clock-numbering}(1)$  have  $v \ c > 0$  by auto
show ?case
proof (cases  $I \ c$ )
  case ( $\text{Const } d$ )
    with  $B(4)$   $\langle c \in X \rangle$  have  $M \ 0 \ (v \ c) \geq Le \ (- \ \text{real } d)$  by auto
    with *  $\text{Const}$  show ?thesis by - (rule dbm-entry-val-mono2[folded
less-eq], auto)
  next
    case ( $\text{Intv } d$ )
      with  $B(4)$   $\langle c \in X \rangle$  have  $M \ 0 \ (v \ c) \geq Lt \ (- \ \text{real } d)$  by auto
      with *  $\text{Intv}$  show ?thesis by - (rule dbm-entry-val-mono2[folded
less-eq], auto)
    next
      case ( $\text{Greater } d$ )
        with  $B(3)$   $\langle c \in X \rangle$  have  $I \ c = \text{Greater } (k \ c)$  by fastforce
        with * have -  $u \ c < - \ k \ c$  by auto
        moreover from  $A(2)$   $*(2)$   $\langle v \ c \leq n \rangle \langle v \ c > 0 \rangle$  have
           $Lt \ (- \ k \ c) \leq M \ 0 \ (v \ c)$ 
        unfolding normalized'-def by force
        ultimately show ?thesis by - (rule dbm-entry-val-mono2[folded
less-eq], auto)
      qed
    next
      case ( $\exists \ u \ I \ J \ r \ c$ )
        note  $B = \text{this}$ 
        from  $B(6)$   $\text{clock-numbering}$  have  $c \in X$  by blast
        with  $B(1)$   $v-v'$  have *:  $\text{intv-elem } c \ u \ (I \ c) \ v' \ (v \ c) = c$  by auto
        from  $\text{clock-numbering}(1)$  have  $v \ c > 0$  by auto
        show ?case
        proof (cases  $I \ c$ )
          case ( $\text{Const } d$ )
            with  $B(4)$   $\langle c \in X \rangle$  have  $M \ (v \ c) \ 0 \geq Le \ d$  by auto
            with *  $\text{Const}$  show ?thesis by - (rule dbm-entry-val-mono3[folded
less-eq], auto)
          next
            case ( $\text{Intv } d$ )
              with  $B(4)$   $\langle c \in X \rangle$  have  $M \ (v \ c) \ 0 \geq Lt \ (\text{real } d + 1)$  by auto
              with *  $\text{Intv}$  show ?thesis by - (rule dbm-entry-val-mono3[folded
less-eq], auto)
            next
              case ( $\text{Greater } d$ )
                with  $B(3)$   $\langle c \in X \rangle$  have  $I \ c = \text{Greater } (k \ c)$  by fastforce
                with  $B(4)$   $\langle c \in X \rangle$  show ?thesis by auto

```

```

qed
next
  case B: ( $\downarrow$  u I J r c1 c2)
  from B(6,7) clock-numbering have  $c1 \in X$   $c2 \in X$  by blast+
  with B(1) v-v' have *:
    intv-elem c1 u (I c1) intv-elem c2 u (I c2) v' (v c1) = c1 v' (v c2)
= c2
  by auto
  from clock-numbering(1) have  $v c1 > 0$   $v c2 > 0$  by auto
  { assume C: isGreater (I c1)  $\vee$  isGreater (I c2)
  with B(1)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  have **: intv'-elem c1 c2 u (J c1 c2)
by force
  have ?case
  proof (cases J c1 c2)
  case (Smaller' c)
  with C B(3)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  have  $c \leq -k c2$  by fastforce
  moreover from C  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  ** Smaller' have  $u c1 -$ 
u c2 < c by auto
  moreover from A(2) *(3,4) B(6,7)  $\langle v c1 > 0 \rangle \langle v c2 > 0 \rangle$  have
    M (v c1) (v c2)  $\geq$  Lt (-k c2)  $\vee$  M (v c1) (v c2) =  $\infty$   $\vee$  v c1
= v c2
  unfolding normalized'-def by fastforce
  ultimately show ?thesis
  by - (safe, rule dbm-entry-val-mono1[folded less-eq], auto,
    smt *(3,4) int-le-real-less of-int-1 of-nat-0-le-iff)
next
  case (Const' c)
  with C B(5)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  have M (v c1) (v c2)  $\geq$  Le c by
auto
  with Const' **  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  show ?thesis
  by (auto intro: dbm-entry-val-mono1[folded less-eq])
next
  case (Intv' c)
  with C B(5)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  have M (v c1) (v c2)  $\geq$  Lt
(real-of-int c + 1) by auto
  with Intv' **  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  show ?thesis
  by (auto intro: dbm-entry-val-mono1[folded less-eq])
next
  case (Greater' c)
  with C B(3)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  have  $c = k c1$  by fastforce
  with Greater' C B(5)  $\langle c1 \in X \rangle \langle c2 \in X \rangle$  show ?thesis by auto
qed
} note GreaterI = this
show ?case

```

```

proof (cases I c1)
  case (Const c)
  show ?thesis
  proof (cases I c2, goal-cases)
    case (1 d)
    with Const ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ *(1,2) have u c1 = c u c2 = d
by auto
    moreover from ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ 1 Const B(5) have
      Lt (real c - real d) ≤ M (v c1) (v c2)
    by meson
    ultimately show ?thesis by (auto intro: dbm-entry-val-mono1[folded
less-eq])
  next
  case (Intv d)
  with Const ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ *(1,2) have u c1 = c d < u c2
by auto
  then have u c1 - u c2 < c - real d by auto
  moreover from Const ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ Intv B(5) have
    Lt (real c - d) ≤ M (v c1) (v c2)
  by meson
  ultimately show ?thesis by (auto intro: dbm-entry-val-mono1[folded
less-eq])
  next
  case Greater then show ?thesis by (auto intro: GreaterI)
  qed
next
  case (Intv c)
  show ?thesis
  proof (cases I c2, goal-cases)
    case (Const d)
    with Intv ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ *(1,2) have u c1 < c + 1 d = u c2
by auto
    then have u c1 - u c2 < c - real d + 1 by auto
    moreover from ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ Intv Const B(5) have
      Lt (real c - real d + 1) ≤ M (v c1) (v c2)
    by meson
    ultimately show ?thesis by (auto intro: dbm-entry-val-mono1[folded
less-eq])
  next
  case (2 d)
  show ?case
  proof (cases (c1,c2) ∈ r)
    case True
    note T = this

```

```

show ?thesis
proof (cases (c2,c1) ∈ r)
  case True
    with T B(5) 2 Intv ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ have
      Le (real c - real d) ≤ M (v c1) (v c2)
    by auto
  moreover from nat-intv-frac-decomp[of c u c1] nat-intv-frac-decomp[of
d u c2]
      B(1,2) ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ T True Intv 2 *(1,2)
    have u c1 - u c2 = real c - d by auto
  ultimately show ?thesis by (auto intro: dbm-entry-val-mono1 [folded
less-eq])
  next
    case False
      with T B(5) 2 Intv ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ have
        Lt (real c - real d) ≤ M (v c1) (v c2)
      by auto
    moreover from nat-intv-frac-decomp[of c u c1] nat-intv-frac-decomp[of
d u c2]
        B(1,2) ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ T False Intv 2 *(1,2)
      have u c1 - u c2 < real c - d by auto
    ultimately show ?thesis by (auto intro: dbm-entry-val-mono1 [folded
less-eq])
    qed
  next
    case False
      with B(5) 2 Intv ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ have
        Lt (real c - real d + 1) ≤ M (v c1) (v c2)
      by meson
    moreover from 2 Intv ⟨c1 ∈ X⟩ ⟨c2 ∈ X⟩ * have u c1 - u c2
< c - real d + 1 by auto
    ultimately show ?thesis by (auto intro: dbm-entry-val-mono1 [folded
less-eq])
    qed
  next
    case Greater then show ?thesis by (auto intro: GreaterI)
    qed
  next
    case Greater then show ?thesis by (auto intro: GreaterI)
    qed
  qed
next
  case 2 show ?case
  proof (safe, goal-cases)

```

**case** (1  $u$ )  
**with**  $A(4)$  **have**  $u \in V$  **unfolding** *DBM-zone-repr-def DBM-val-bounded-def*  
**by** *auto*  
**with** *region-cover* **obtain**  $R$  **where**  $R \in \mathcal{R}$   $u \in R$  **unfolding** *V-def*  
**by** *auto*  
**then obtain**  $I J r$  **where**  $R: R = \text{region } X I J r \text{ valid-region } X k I J$   
 $r$  **unfolding** *R-def* **by** *auto*  
**have**  $(\forall c \in X. (\forall d. I c = \text{Const } d \longrightarrow \text{Le } (\text{real } d) \leq M (v c) 0 \wedge \text{Le}$   
 $(- \text{real } d) \leq M 0 (v c)) \wedge$   
 $(\forall d. I c = \text{Intv } d \longrightarrow \text{Lt } (\text{real } d + 1) \leq M (v c) 0 \wedge \text{Lt } (-$   
 $\text{real } d) \leq M 0 (v c)) \wedge$   
 $(I c = \text{Greater } (k c) \longrightarrow M (v c) 0 = \infty))$   
**proof** *safe*  
**fix**  $c$  **assume**  $c \in X$   
**with**  $R \langle u \in R \rangle$  **have**  $*$ : *intv-elem*  $c u (I c)$  **by** *auto*  
**fix**  $d$  **assume**  $**$ :  $I c = \text{Const } d$   
**with**  $*$  **have**  $u c = d$  **by** *fastforce*  
**moreover from**  $**$  *clock-numbering*(3)  $\langle c \in X \rangle 1$  **have**  
 $\text{dbm-entry-val } u (\text{Some } c) \text{None } (M (v c) 0)$   
**by** *auto*  
**ultimately show**  $\text{Le } (\text{real } d) \leq M (v c) 0$   
**unfolding** *less-eq dbm-le-def* **by** (cases  $M (v c) 0$ ) *auto*  
**next**  
**fix**  $c$  **assume**  $c \in X$   
**with**  $R \langle u \in R \rangle$  **have**  $*$ : *intv-elem*  $c u (I c)$  **by** *auto*  
**fix**  $d$  **assume**  $**$ :  $I c = \text{Const } d$   
**with**  $*$  **have**  $u c = d$  **by** *fastforce*  
**moreover from**  $**$  *clock-numbering*(3)  $\langle c \in X \rangle 1$  **have**  
 $\text{dbm-entry-val } u \text{None } (\text{Some } c) (M 0 (v c))$   
**by** *auto*  
**ultimately show**  $\text{Le } (- \text{real } d) \leq M 0 (v c)$   
**unfolding** *less-eq dbm-le-def* **by** (cases  $M 0 (v c)$ ) *auto*  
**next**  
**fix**  $c$  **assume**  $c \in X$   
**with**  $R \langle u \in R \rangle$  **have**  $*$ : *intv-elem*  $c u (I c)$  **by** *auto*  
**fix**  $d$  **assume**  $**$ :  $I c = \text{Intv } d$   
**with**  $*$  **have**  $d < u c u c < d + 1$  **by** *fastforce+*  
**moreover from**  $**$  *clock-numbering*(3)  $\langle c \in X \rangle 1$  **have**  
 $\text{dbm-entry-val } u (\text{Some } c) \text{None } (M (v c) 0)$   
**by** *auto*  
**moreover have**  
 $M (v c) 0 \neq \infty \implies \text{get-const } (M (v c) 0) \in \mathbb{Z}$   
**using**  $\langle c \in X \rangle$  *clock-numbering*  $A(1)$  **by** *auto*  
**ultimately show**  $\text{Lt } (\text{real } d + 1) \leq M (v c) 0$  **unfolding** *less-eq*

```

dbm-le-def
  apply (cases M (v c) 0)
    apply auto
    apply (rename-tac x1)
    apply (subgoal-tac x1 > d)
    apply (rule dbm-lt.intros(5))
    apply (metis nat-intv-frac-gt0 frac-eq-0-iff less-irrefl linorder-not-le
of-nat-1 of-nat-add)
    apply simp
    apply (rename-tac x2)
    apply (subgoal-tac x2 > d + 1)
    apply (rule dbm-lt.intros(6))
    apply (metis of-nat-1 of-nat-add)
    apply simp
  by (metis nat-intv-not-int One-nat-def add.commute add.right-neutral
add-Suc-right le-less-trans
      less-eq-real-def linorder-neqE-linordered-idom semir-
ing-1-class.of-nat-simps(2))
next
  fix c assume c ∈ X
  with R ⟨u ∈ R⟩ have *: intv-elem c u (I c) by auto
  fix d assume **: I c = Intv d
  with * have d < u c u c < d + 1 by fastforce+
  moreover from ** clock-numbering(3) ⟨c ∈ X⟩ 1 have
    dbm-entry-val u None (Some c) (M 0 (v c))
  by auto
  moreover have M 0 (v c) ≠ ∞ ⇒ get-const (M 0 (v c)) ∈ ℤ
using ⟨c ∈ X⟩ clock-numbering A(1) by auto
  ultimately show Lt (− real d) ≤ M 0 (v c) unfolding less-eq
dbm-le-def
  proof (cases M 0 (v c), −, auto, goal-cases)
    case prems: (1 x1)
    then have u c = d + frac (u c) by (metis nat-intv-frac-decomp
⟨u c < d + 1⟩)
    with prems(5) have − x1 ≤ d + frac (u c) by auto
    with prems(1) frac-ge-0 frac-lt-1 have − x1 ≤ d
    by − (rule ints-le-add-frac2[of frac (u c) d −x1]; fastforce)
    with prems have − d ≤ x1 by auto
    then show ?case by auto
  next
    case prems: (2 x1)
    then have u c = d + frac (u c) by (metis nat-intv-frac-decomp
⟨u c < d + 1⟩)
    with prems(5) have − x1 ≤ d + frac (u c) by auto

```

```

    with prems(1) frac-ge-0 frac-lt-1 have  $-x1 \leq d$ 
    by - (rule ints-le-add-frac2[of frac (u c) d -x1]; fastforce)
    with prems(6) have  $-d < x1$  by auto
    then show ?case by auto
qed
next
fix c assume  $c \in X$ 
with R  $\langle u \in R \rangle$  have *: intv-elem c u (I c) by auto
fix d assume **:  $I c = \text{Greater } (k c)$ 
have  $M (v c) 0 \leq \text{Le } ((k o v') (v c)) \vee M (v c) 0 = \infty$ 
using A(2)  $\langle c \in X \rangle$  clock-numbering unfolding normalized'-def by
auto
with  $v-v' \langle c \in X \rangle$  have  $M (v c) 0 \leq \text{Le } (k c) \vee M (v c) 0 = \infty$  by
auto
moreover from * ** have  $k c < u c$  by fastforce
moreover from ** clock-numbering(3)  $\langle c \in X \rangle$  1 have
  dbm-entry-val u (Some c) None (M (v c) 0)
by auto
moreover have
   $M (v c) 0 \neq \infty \implies \text{get-const } (M (v c) 0) \in \mathbb{Z}$ 
using  $\langle c \in X \rangle$  clock-numbering A(1) by auto
ultimately show  $M (v c) 0 = \infty$  unfolding less-eq dbm-le-def
  apply -
  apply (rule ccontr)
  using ** apply (cases M (v c) 0)
  by auto
qed
moreover
{ fix x y assume X:  $x \in X y \in X$ 
  with R  $\langle u \in R \rangle$  have *: intv-elem x u (I x) intv-elem y u (I y) by
auto
  from X R  $\langle u \in R \rangle$  have **:
     $\text{isGreater } (I x) \vee \text{isGreater } (I y) \longrightarrow \text{intv}'\text{-elem } x y u (J x y)$ 
  by force
  have int:  $M (v x) (v y) \neq \infty \implies \text{get-const } (M (v x) (v y)) \in \mathbb{Z}$ 
using X clock-numbering A(1)
  by auto
  have int2:  $M (v y) (v x) \neq \infty \implies \text{get-const } (M (v y) (v x)) \in \mathbb{Z}$ 
using X clock-numbering A(1)
  by auto
  from 1 clock-numbering(3) X 1 have ***:
    dbm-entry-val u (Some x) (Some y) (M (v x) (v y))
    dbm-entry-val u (Some y) (Some x) (M (v y) (v x))
  by auto

```

**have**  
 $(\forall c d. I x = \text{Intv } c \wedge I y = \text{Intv } d \longrightarrow M (v x) (v y) \geq$   
 $(\text{if } (x, y) \in r \text{ then if } (y, x) \in r \text{ then } \text{Le } (c - d) \text{ else } \text{Lt } (c - d)$   
 $\text{else } \text{Lt } (c - d + 1))) \wedge$   
 $(\forall c d. I x = \text{Intv } c \wedge I y = \text{Intv } d \longrightarrow M (v y) (v x) \geq$   
 $(\text{if } (y, x) \in r \text{ then if } (x, y) \in r \text{ then } \text{Le } (d - c) \text{ else } \text{Lt } (d - c)$   
 $\text{else } \text{Lt } (d - c + 1))) \wedge$   
 $(\forall c d. I x = \text{Const } c \wedge I y = \text{Const } d \longrightarrow M (v x) (v y) \geq \text{Le } (c$   
 $- d)) \wedge$   
 $(\forall c d. I x = \text{Const } c \wedge I y = \text{Const } d \longrightarrow M (v y) (v x) \geq \text{Le } (d$   
 $- c)) \wedge$   
 $(\forall c d. I x = \text{Intv } c \wedge I y = \text{Const } d \longrightarrow M (v x) (v y) \geq \text{Lt } (c -$   
 $d + 1)) \wedge$   
 $(\forall c d. I x = \text{Intv } c \wedge I y = \text{Const } d \longrightarrow M (v y) (v x) \geq \text{Lt } (d -$   
 $c)) \wedge$   
 $(\forall c d. I x = \text{Const } c \wedge I y = \text{Intv } d \longrightarrow M (v x) (v y) \geq \text{Lt } (c -$   
 $d)) \wedge$   
 $(\forall c d. I x = \text{Const } c \wedge I y = \text{Intv } d \longrightarrow M (v y) (v x) \geq \text{Lt } (d -$   
 $c + 1)) \wedge$   
 $((\text{isGreater } (I x) \vee \text{isGreater } (I y)) \wedge J x y = \text{Greater}' (k x) \longrightarrow$   
 $M (v x) (v y) = \infty) \wedge$   
 $(\forall c. (\text{isGreater } (I x) \vee \text{isGreater } (I y)) \wedge J x y = \text{Const}' c$   
 $\longrightarrow M (v x) (v y) \geq \text{Le } c \wedge M (v y) (v x) \geq \text{Le } (- c)) \wedge$   
 $(\forall c. (\text{isGreater } (I x) \vee \text{isGreater } (I y)) \wedge J x y = \text{Intv}' c$   
 $\longrightarrow M (v x) (v y) \geq \text{Lt } (c + 1) \wedge M (v y) (v x) \geq \text{Lt } (- c))$   
**proof** (*auto*, *goal-cases*)  
**case** \*\*: (1 c d)  
**with**  $R \langle u \in R \rangle X$  **have**  $\text{frac } (u x) = \text{frac } (u y)$  **by** *auto*  
**with** \* \*\* *nat-intv-frac-decomp*[of c u x] *nat-intv-frac-decomp*[of d  
u y] **have**  
 $u x - u y = \text{real } c - d$   
**by** *auto*  
**with** \*\*\* **show** ?*case unfolding less-eq dbm-le-def* **by** (*cases M*  
(v x) (v y)) *auto*  
**next**  
**case** \*\*: (2 c d)  
**with**  $R \langle u \in R \rangle X$  **have**  $\text{frac } (u x) > \text{frac } (u y)$  **by** *auto*  
**with** \* \*\* *nat-intv-frac-decomp*[of c u x] *nat-intv-frac-decomp*[of d  
u y] **have**  
 $\text{real } c - d < u x - u y \wedge u x - u y < \text{real } c - d + 1$   
**by** *auto*  
**with** \*\*\* *int show ?case unfolding less-eq dbm-le-def*  
**by** (*cases M (v x) (v y)*, *auto*) (*fastforce intro: int-lt-Suc-le*  
*int-lt-neq-prev-lt*)+

```

next
  case **: (3 c d)
  from **  $R \langle u \in R \rangle X$  have  $\text{frac } (u x) < \text{frac } (u y)$  by auto
  with * ** nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d
u y] have
     $\text{real } c - d - 1 < u x - u y \ u x - u y < \text{real } c - d$ 
  by auto
  with *** int show ?case unfolding less-eq dbm-le-def
    by (cases M (v x) (v y), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (4 c d) with  $R(1) \langle u \in R \rangle X$  show ?case by auto
  next
    case **: (5 c d)
    with  $R \langle u \in R \rangle X$  have  $\text{frac } (u x) = \text{frac } (u y)$  by auto
    with * ** nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d
u y] have
       $u x - u y = \text{real } c - d$  by auto
    with *** show ?case unfolding less-eq dbm-le-def by (cases M
(v y) (v x)) auto
  next
    case **: (6 c d)
    from **  $R \langle u \in R \rangle X$  have  $\text{frac } (u x) < \text{frac } (u y)$  by auto
    with * ** nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d
u y] have
       $\text{real } d - c < u y - u x \ u y - u x < \text{real } d - c + 1$ 
    by auto
    with *** int2 show ?case unfolding less-eq dbm-le-def
      by (cases M (v y) (v x), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case **: (7 c d)
    from **  $R \langle u \in R \rangle X$  have  $\text{frac } (u x) > \text{frac } (u y)$  by auto
    with * ** nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d
u y] have
       $\text{real } d - c - 1 < u y - u x \ u y - u x < \text{real } d - c$ 
    by auto
    with *** int2 show ?case unfolding less-eq dbm-le-def
      by (cases M (v y) (v x), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (8 c d) with  $R(1) \langle u \in R \rangle X$  show ?case by auto
  next
    case (9 c d)

```

```

    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have
      u x - u y = real c - d by auto
    with *** show ?case unfolding less-eq dbm-le-def by (cases M
(v x) (v y)) auto
  next
    case (10 c d)
    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have
      u x - u y = real c - d
    by auto
    with *** show ?case unfolding less-eq dbm-le-def by (cases M
(v y) (v x)) auto
  next
    case (11 c d)
    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have
      real c - d < u x - u y
    by auto
    with *** int show ?case unfolding less-eq dbm-le-def
      by (cases M (v x) (v y), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (12 c d)
    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have
      real d - c - 1 < u y - u x
    by auto
    with *** int2 show ?case unfolding less-eq dbm-le-def
      by (cases M (v y) (v x), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (13 c d)
    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have
      real c - d - 1 < u x - u y
    by auto
    with *** int show ?case unfolding less-eq dbm-le-def
      by (cases M (v x) (v y), auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (14 c d)
    with * nat-intv-frac-decomp[of c u x] nat-intv-frac-decomp[of d u
y] have

```

```

      real  $d - c < u y - u x$ 
    by auto
    with *** int2 show ?case unfolding less-eq dbm-le-def
      by (cases  $M (v y) (v x)$ , auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
    next
      case (15 d)
      have  $M (v x) (v y) \leq Le ((k o v') (v x)) \vee M (v x) (v y) = \infty \vee v$ 
 $x = v y$ 
        using A(2) X clock-numbering unfolding normalized'-def by
metis
      with  $v-v' X$  have  $M (v x) (v y) \leq Le (k x) \vee M (v x) (v y) = \infty$ 
 $\vee v x = v y$  by auto
      moreover from 15 * ** have  $u x - u y > k x$  by auto
      ultimately show ?case
        unfolding less-eq dbm-le-def using ***
        by (cases  $M (v x) (v y)$ , auto) (smt X(1) X(2) of-nat-0-le-iff
 $v-v'$ )+
    next
      case (16 d)
      have  $M (v x) (v y) \leq Le ((k o v') (v x)) \vee M (v x) (v y) = \infty \vee v$ 
 $x = v y$ 
        using A(2) X clock-numbering unfolding normalized'-def by metis
      with  $v-v' X$  have  $M (v x) (v y) \leq Le (k x) \vee M (v x) (v y) = \infty$ 
 $\vee v x = v y$  by auto
      moreover from 16 * ** have  $u x - u y > k x$  by auto
      ultimately show ?case
        unfolding less-eq dbm-le-def using ***
        by (cases  $M (v x) (v y)$ , auto) (smt X(1) X(2) of-nat-0-le-iff
 $v-v'$ )+
    next
      case 17 with ** *** show ?case unfolding less-eq dbm-le-def by
(cases  $M (v x) (v y)$ , auto)
    next
      case 18 with ** *** show ?case unfolding less-eq dbm-le-def by
(cases  $M (v y) (v x)$ , auto)
    next
      case 19 with ** *** show ?case unfolding less-eq dbm-le-def by
(cases  $M (v x) (v y)$ , auto)
    next
      case 20 with ** *** show ?case unfolding less-eq dbm-le-def by
(cases  $M (v y) (v x)$ , auto)
    next
      case (21 c d)

```

```

with ** have  $c < u\ x - u\ y$  by auto
with *** int show ?case unfolding less-eq dbm-le-def
  by (cases  $M\ (v\ x)\ (v\ y)$ , auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
next
  case (22  $c\ d$ )
  with ** have  $u\ x - u\ y < c + 1$  by auto
  then have  $u\ y - u\ x > -c - 1$  by auto
  with *** int2 show ?case unfolding less-eq dbm-le-def
    by (cases  $M\ (v\ y)\ (v\ x)$ , auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
  next
    case (23  $c\ d$ )
    with ** have  $c < u\ x - u\ y$  by auto
    with *** int show ?case unfolding less-eq dbm-le-def
      by (cases  $M\ (v\ x)\ (v\ y)$ , auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
    next
      case (24  $c\ d$ )
      with ** have  $u\ x - u\ y < c + 1$  by auto
      then have  $u\ y - u\ x > -c - 1$  by auto
      with *** int2 show ?case unfolding less-eq dbm-le-def
        by (cases  $M\ (v\ y)\ (v\ x)$ , auto) (fastforce intro: int-lt-Suc-le
int-lt-neq-prev-lt)+
      qed
    }
  ultimately show ?case using  $R\ \langle u \in R \rangle\ \langle R \in \mathcal{R} \rangle$ 
  apply -
  apply standard
  apply standard
  apply rule
  apply assumption
  apply (rule  $exI[\mathbf{where}\ x = I]$ , rule  $exI[\mathbf{where}\ x = J]$ , rule  $exI[\mathbf{where}$ 
 $x = r]$ )
  by auto
  qed
qed
with  $A$  have  $S = \bigcup ?U$  by auto
moreover have  $?U \subseteq \mathcal{R}$  by blast
ultimately show ?case by blast
qed

```

**lemma** *dbm-regions'*:

$$vabstr\ S\ M \implies normalized'\ M \implies S \subseteq V \implies \exists\ U \subseteq \mathcal{R}. S = \bigcup U$$

**using** *dbm-regions* **by** (*cases*  $S = \{\}$ ) *auto*

**lemma** *dbm-regions''*:

*dbm-int*  $M\ n \implies \text{normalized}'\ M \implies [M]_{v,n} \subseteq V \implies \exists\ U \subseteq \mathcal{R}. [M]_{v,n} = \bigcup U$

**using** *dbm-regions'* **by** *auto*

**lemma** *DBM-le-subset'*:

**assumes**  $\forall i \leq n. \forall j \leq n. i \neq j \longrightarrow M\ i\ j \leq M'\ i\ j$

**and**  $\forall i \leq n. M'\ i\ i \geq Le\ 0$

**and**  $u \in [M]_{v,n}$

**shows**  $u \in [M']_{v,n}$

**proof** –

**let**  $?M = \lambda i\ j. \text{if } i = j \text{ then } Le\ 0 \text{ else } M\ i\ j$

**have**  $\forall i\ j. i \leq n \longrightarrow j \leq n \longrightarrow ?M\ i\ j \leq M'\ i\ j$  **using** *assms(1,2)* **by** *auto*

**moreover from** *DBM-set-diag* *assms(3)* **have**  $u \in [?M]_{v,n}$  **by** *auto*

**ultimately show** *?thesis* **using** *DBM-le-subset* [*folded less-eq, of n ?M M' u v*] **by** *auto*

**qed**

**lemma** *neg-diag-empty-spec*:

**assumes**  $i \leq n\ M\ i\ i < 0$

**shows**  $[M]_{v,n} = \{\}$

**using** *assms neg-diag-empty* [**where**  $v = v$  **and**  $M = M$ , *OF - assms*] *clock-numbering(2)* **by** *auto*

**lemma** *canonical-empty-zone-spec*:

**assumes** *canonical*  $M\ n$

**shows**  $[M]_{v,n} = \{\} \longleftrightarrow (\exists i \leq n. M\ i\ i < 0)$

**using** *canonical-empty-zone* [*of n v M, OF - - assms*] *clock-numbering* **by** *auto*

**lemma** *norm-set-diag*:

**assumes** *canonical*  $M\ n$   $[M]_{v,n} \neq \{\}$

**obtains**  $M'$  **where**  $[M]_{v,n} = [M']_{v,n}$   $[norm\ M\ (k\ o\ v')\ n]_{v,n} = [norm\ M'\ (k\ o\ v')\ n]_{v,n}$

$\forall i \leq n. M'\ i\ i = 0$  *canonical*  $M'\ n$

**proof** –

**from** *assms(2)* *neg-diag-empty-spec* **have**  $*$ :  $\forall i \leq n. M\ i\ i \geq Le\ 0$  **unfolding** *neutral* **by** *force*

**let**  $?M = \lambda i\ j. \text{if } i = j \text{ then } Le\ 0 \text{ else } M\ i\ j$

**let**  $?NM = norm\ M\ (k\ o\ v')\ n$

**let**  $?M2 = \lambda i\ j. \text{if } i = j \text{ then } Le\ 0 \text{ else } ?NM\ i\ j$

**from** *assms* **have**  $[?NM]_{v,n} \neq \{\}$   
**by** (*metis Collect-empty-eq norm-mono DBM-zone-repr-def clock-numbering(1)*  
*mem-Collect-eq*)  
**from** *DBM-set-diag[OF this] DBM-set-diag[OF assms(2)]* **have**  
 $[M]_{v,n} = [?M]_{v,n} [?NM]_{v,n} = [?M2]_{v,n}$   
**by** *auto*  
**moreover** **have** *norm ?M (k o v') n = ?M2 unfolding norm-def norm-diag-def*  
**by** *fastforce*  
**moreover** **have**  $\forall i \leq n. ?M i i = 0$  **unfolding** *neutral* **by** *auto*  
**moreover** **have** *canonical ?M n using assms(1) \**  
**unfolding** *neutral[symmetric] less-eq[symmetric] add[symmetric]* **by** *fast-*  
*force*  
**ultimately** **show** *?thesis* **by** (*auto intro: that*)  
**qed**

**lemma** *norm-normalizes'*:  
**notes** *any-le-inf[intro]*  
**shows** *normalized' (norm M (k o v') n)*  
**unfolding** *normalized'-def*  
**proof** (*safe, goal-cases*)  
**case** (1 *i j*)  
**show** *?case*  
**proof** (*cases M i j < Lt (- real (k (v' j)))*)  
**case** *True with 1 show ?thesis unfolding norm-def less* **by** (*auto simp:*  
*Let-def neutral*)  
**next**  
**case** *False*  
**with 1 show ?thesis unfolding norm-def** **by** (*auto simp: Let-def*)  
**qed**  
**next**  
**case** (2 *i j*)  
**have** *\*\**:  $- \text{real} ((k \text{ o } v') j) \leq (k \text{ o } v') i$  **by** *simp*  
**then** **have** *\**:  $Lt (- k (v' j)) < Le (k (v' i))$  **by** (*auto intro: Lt-lt-LeI*)  
**show** *?case*  
**proof** (*cases M i j ≤ Le (real (k (v' i)))*)  
**case** *False with 2 show ?thesis*  
**unfolding** *norm-def less-eq dbm-le-def* **by** (*auto simp: Let-def neutral*  
*split: if-split-asm*)  
**next**  
**case** *True with 2 show ?thesis unfolding norm-def* **by** (*auto simp:*  
*Let-def split: if-split-asm*)  
**qed**  
**next**  
**case** (3 *i*)

```

show ?case
proof (cases M i 0 ≤ Le (real (k (v' i))))
  case False then have Le (real (k (v' i))) < M i 0 unfolding less-eq
  dbm-le-def by auto
  with 3 show ?thesis unfolding norm-def by auto
next
  case True
  with 3 show ?thesis unfolding norm-def less-eq dbm-le-def by (auto
  simp: Let-def)
qed
next
  case (4 i)
  show ?case
  proof (cases M 0 i < Lt (- real (k (v' i))))
    case True with 4 show ?thesis unfolding norm-def less by auto
  next
    case False with 4 show ?thesis unfolding norm-def by (auto simp:
  Let-def)
  qed
qed

```

**lemma** norm-normalizes:

```

assumes  $\forall i \leq n. M i i = 0$ 
shows normalized (norm M (k o v') n)
apply (rule normalized'-normalized)
subgoal
  using assms unfolding norm-def norm-diag-def by (auto simp: DBM.neutral)
by (rule norm-normalizes')

```

**lemma** norm-int-preservation:

```

fixes M :: real DBM
assumes dbm-int M n i ≤ n j ≤ n norm M (k o v') n i j ≠ ∞
shows get-const (norm M (k o v') n i j) ∈  $\mathbb{Z}$ 
using assms unfolding norm-def by (auto simp: Let-def norm-diag-def)

```

**lemma** norm-V-preservation':

```

notes any-le-inf[intro]
assumes  $[M]_{v,n} \subseteq V$  canonical M n  $[M]_{v,n} \neq \{\}$ 
shows  $[norm M (k o v') n]_{v,n} \subseteq V$ 
proof -
  let ?M = norm M (k o v') n
  from non-empty-cycle-free[OF assms(3)] clock-numbering(2) have *: cycle-free M n by auto
  { fix c assume c ∈ X

```

**with** *clock-numbering* **have**  $c \in X \ v \ c > 0 \ v \ c \leq n$  **by** *auto*  
**with** *assms(2)* **have**  
 $M \ 0 \ (v \ c) + M \ (v \ c) \ 0 \geq M \ 0 \ 0$   
**unfolding** *add less-eq* **by** *blast*  
**moreover from** *cycle-free-diag[OF \*]* **have**  $M \ 0 \ 0 \geq Le \ 0$  **unfolding**  
*neutral* **by** *auto*  
**ultimately have** *ge-0*:  $M \ 0 \ (v \ c) + M \ (v \ c) \ 0 \geq Le \ 0$  **by** *auto*  
**have**  $M \ 0 \ (v \ c) \leq Le \ 0$   
**proof** (*cases*  $M \ 0 \ (v \ c)$ )  
**case** (*Le d*)  
**with** *ge-0* **have**  $M \ (v \ c) \ 0 \geq Le \ (-d)$   
**unfolding** *add* **by** (*cases*  $M \ (v \ c) \ 0$ ) *auto*  
**with** *Le canonical-saturated-2* [**where**  $v = v$ , *OF - -*  $\langle$ *cycle-free*  $M \ n \rangle$   
*assms(2)* *c(3)*]  
*clock-numbering(1)*  
**obtain**  $u$  **where**  $u \in [M]_{v,n} \ u \ c = -d$  **by** *auto*  
**with** *assms(1)* *c(1)* *Le* **show** *?thesis* **unfolding** *V-def* **by** *fastforce*  
**next**  
**case** (*Lt d*)  
**show** *?thesis*  
**proof** (*cases*  $d \leq 0$ )  
**case** *True*  
**then have**  $Lt \ d < Le \ 0$  **by** (*auto intro: Lt-lt-LeI*)  
**with** *Lt* **show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**then have**  $d > 0$  **by** *auto*  
**note**  $Lt' = Lt$   
**show** *?thesis*  
**proof** (*cases*  $M \ (v \ c) \ 0$ )  
**case** (*Le d'*)  
**with** *Lt ge-0* **have**  $*$ :  $d > -d'$  **unfolding** *add* **by** *auto*  
**show** *?thesis*  
**proof** (*cases*  $d' < 0$ )  
**case** *True*  
**from**  
 $*$  *clock-numbering(1)*  
*canonical-saturated-1* [**where**  $v = v$ , *OF - -*  $\langle$ *cycle-free*  $- \ - \rangle$   
*assms(2)* *c(3)*] *Lt Le*  
**obtain**  $u$  **where**  $u \in [M]_{v,n} \ u \ c = d'$   
**by** *auto*  
**with**  $\langle d' < 0 \rangle$  *assms(1)*  $\langle c \in X \rangle$  **show** *?thesis* **unfolding** *V-def*  
**by** *fastforce*  
**next**

```

      case False
      then have  $d' \geq 0$  by auto
      with  $\langle d > 0 \rangle$  have  $Le (d / 2) \leq Lt d Le (- (d / 2)) \leq Le d'$  by
auto
      with
        canonical-saturated-2[where  $v = v$ , OF - -  $\langle cycle-free - - \rangle$ 
assms(2)  $c$ (3)]
        Lt Le clock-numbering(1)
      obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - (d / 2)$ 
      by auto (metis Le-le-LtD  $\langle Le (d / 2) \leq Lt d \rangle$ )
      with  $\langle d > 0 \rangle$  assms(1)  $\langle c \in X \rangle$  show ?thesis unfolding V-def
by fastforce
      qed
    next
      case (Lt d')
      with Lt' ge-0 have  $*$ :  $d > -d'$  unfolding add by auto
      then have  $**$ :  $-d < d'$  by auto
      show ?thesis
      proof (cases  $d' \leq 0$ )
        case True
        from assms(1,3)  $c$  obtain  $u$  where  $u$ :
           $u \in V dbm-entry-val u (Some c) None (M (v c) 0)$ 
        unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
        with  $u$ (1) True Lt  $\langle c \in X \rangle$  show ?thesis unfolding V-def by
auto
      next
        case False
        with  $\langle d > 0 \rangle$  have  $Le (d / 2) \leq Lt d Le (- (d / 2)) \leq Lt d'$  by
auto
      with
        canonical-saturated-2[where  $v = v$ , OF - -  $\langle cycle-free - - \rangle$ 
assms(2)  $c$ (3)]
        Lt Lt' clock-numbering(1)
      obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - (d / 2)$ 
      by auto (metis Le-le-LtD  $\langle Le (d / 2) \leq Lt d \rangle$ )
      with  $\langle d > 0 \rangle$  assms(1)  $\langle c \in X \rangle$  show ?thesis unfolding V-def
by fastforce
      qed
    next
      case INF
      show ?thesis
      proof (cases  $d > 0$ )
        case True
        from  $\langle d > 0 \rangle$  have  $Le (d / 2) \leq Lt d$  by auto

```

```

with
  INF canonical-saturated-2[where  $v = v$ , OF - -  $\langle$ cycle-free - - $\rangle$ 
  assms(2) c(3)]
  Lt clock-numbering(1)
obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - (d / 2)$ 
  by auto (metis Le-le-LtD  $\langle$ Le (d / 2)  $\leq$  Lt d $\rangle$  any-le-inf)
  with  $\langle d > 0 \rangle$  assms(1)  $\langle c \in X \rangle$  show ?thesis unfolding V-def
by fastforce
  next
    case False
    with Lt show ?thesis by auto
  qed
  qed
  qed
next
  case INF
  obtain  $u$   $r$  where  $u \in [M]_{v,n}$   $u c = - r$   $r > 0$ 
  proof (cases M (v c) 0)
    case (Le d)
    let ?d = if  $d \leq 0$  then  $-d + 1$  else  $d$ 
    from Le INF canonical-saturated-2[where  $v = v$ , OF - -  $\langle$ cycle-free
    - - $\rangle$  assms(2) c(3), of ?d]
    clock-numbering(1)
    obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - ?d$  by (cases  $d < 0$ ) (auto
    simp: any-le-inf, smt)
    from that[OF this] show thesis by auto
  next
    case (Lt d)
    let ?d = if  $d \leq 0$  then  $-d + 1$  else  $d$ 
    from Lt INF canonical-saturated-2[where  $v = v$ , OF - -  $\langle$ cycle-free
    - - $\rangle$  assms(2) c(3), of ?d]
    clock-numbering(1)
    obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - ?d$  by (cases  $d < 0$ ) (auto
    simp: any-le-inf, smt)
    from that[OF this] show thesis by auto
  next
    case INF
    with
       $\langle M 0 (v c) = \infty \rangle$  canonical-saturated-2[where  $v = v$ , OF - -
       $\langle$ cycle-free - - $\rangle$  assms(2) c(3)]
      clock-numbering(1)
    obtain  $u$  where  $u \in [M]_{v,n}$   $u c = - 1$  by auto
    from that[OF this] show thesis by auto
  qed

```

**with** *assms*(1)  $\langle c \in X \rangle$  **show** *?thesis* **unfolding** *V-def* **by** *fastforce*  
**qed**  
**moreover then have**  $\neg Le\ 0 \prec M\ 0\ (v\ c)$  **unfolding** *less[symmetric]*  
**by** *auto*  
**ultimately have**  $*$ :  $?M\ 0\ (v\ c) \leq Le\ 0$   
**using** *assms*(3) *c* **unfolding** *norm-def* **by** (*auto simp: Let-def*)  
**fix** *u* **assume** *u*:  $u \in [?M]_{v,n}$   
**with** *c* **have** *dbm-entry-val* *u* *None* (*Some* *c*) ( $?M\ 0\ (v\ c)$ )  
**unfolding** *DBM-val-bounded-def* *DBM-zone-repr-def* **by** *auto*  
**with**  $*$  **have**  $u\ c \geq 0$  **by** (*cases*  $?M\ 0\ (v\ c)$ ) *auto*  
**}** **note** *ge-0 = this*  
**then show** *?thesis* **unfolding** *V-def* **by** *auto*  
**qed**

**lemma** *norm-V-preservation*:

**assumes**  $[M]_{v,n} \subseteq V$  *canonical* *M* *n*  
**shows**  $[norm\ M\ (k\ o\ v')\ n]_{v,n} \subseteq V$  (**is**  $[?M]_{v,n} \subseteq V$ )  
**proof** (*cases*  $[M]_{v,n} = \{\}$ )  
**case** *True*  
**obtain** *i* **where**  $i: i \leq n$   $M\ i\ i < 0$  **by** (*metis* *True* *assms*(2) *canonical-empty-zone-spec*)  
**have**  $\neg Le\ (real\ (k\ (v'\ i))) < Le\ 0$  **unfolding** *less* **by** (*cases*  $k\ (v'\ i) = 0$ , *auto*)  
**with** *i* **have**  $?M\ i\ i < 0$  **unfolding** *norm-def* **by** (*auto simp: neutral less Let-def norm-diag-def*)  
**with** *neg-diag-empty-spec*[*OF*  $\langle i \leq n \rangle$ ] **have**  $[?M]_{v,n} = \{\}$  .  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**with** *assms* **show** *?thesis*  
**apply** –  
**apply** (*rule* *norm-set-diag*[*OF* *assms*(2) *False*])  
**apply** (*rule* *norm-V-preservation'*)  
**apply** *auto*  
**done**  
**qed**

**lemma** *norm-min*:

**assumes** *normalized'* *M1*  $[M]_{v,n} \subseteq [M1]_{v,n}$   
*canonical* *M* *n*  $[M]_{v,n} \neq \{\}$   $[M]_{v,n} \subseteq V$   
**shows**  $[norm\ M\ (k\ o\ v')\ n]_{v,n} \subseteq [M1]_{v,n}$  (**is**  $[?M2]_{v,n} \subseteq [M1]_{v,n}$ )  
**proof** –  
**have** *le*:  $\bigwedge i\ j. i \leq n \implies j \leq n \implies i \neq j \implies M\ i\ j \leq M1\ i\ j$   
**using** *assms*(2,3,4) *clock-numbering*(2)

**by** (*auto intro!*: *DBM-canonical-subset-le*[*OF* - - - - - *clock-numbering*(1)])  
**from** *assms* **have**  $[M1]_{v,n} \neq \{\}$  **by** *auto*  
**with** *neg-diag-empty-spec* **have**  $*$ :  $\forall i \leq n. M1\ i\ i \geq Le\ 0$  **unfolding**  
*neutral* **by** *force*  
**from** *assms* *norm-V-preservation* **have**  $V: [?M2]_{v,n} \subseteq V$  **by** *auto*  
**have**  $u \in [M1]_{v,n}$  **if**  $u \in [?M2]_{v,n}$  **for**  $u$   
**proof** -  
**from** *that*  $V$  **have**  $V: u \in V$  **by** *fast*  
**show** *?thesis* **unfolding** *DBM-zone-repr-def* *DBM-val-bounded-def*  
**proof** (*safe*, *goal-cases*)  
**case** 1 **with**  $*$  **show** *?case* **unfolding** *less-eq* **by** *fast*  
**next**  
**case** (2  $c$ )  
**then** **have**  $c: v\ c > 0\ v\ c \leq n\ c \in X\ v'\ (v\ c) = c$  **using** *clock-numbering*  
*v-v'* **by** *metis+*  
**with**  $V$  **have** *v-bound*: *dbm-entry-val*  $u\ None\ (Some\ c)\ (Le\ 0)$  **un-**  
**folding** *V-def* **by** *auto*  
**from** *that*  $c$  **have** *bound*:  
*dbm-entry-val*  $u\ None\ (Some\ c)\ (?M2\ 0\ (v\ c))$   
**unfolding** *DBM-zone-repr-def* *DBM-val-bounded-def* **by** *auto*  
**show** *?case*  
**proof** (*cases*  $M\ 0\ (v\ c) < Lt\ (-\ k\ c)$ )  
**case** *False*  
**show** *?thesis*  
**proof** (*cases*  $Le\ 0 < M\ 0\ (v\ c)$ )  
**case** *True*  
**with**  $le\ c(1,2)$  **have**  $Le\ 0 \leq M1\ 0\ (v\ c)$  **by** *fastforce*  
**with** *dbm-entry-val-mono2*[*OF* *v-bound*, *folded less-eq*] **show** *?thesis*  
**by** *fast*  
**next**  
**case**  $F: False$   
**with** *assms*(3)  $False\ c$  **have**  $?M2\ 0\ (v\ c) = M\ 0\ (v\ c)$  **unfolding**  
*less norm-def* **by** *auto*  
**with**  $le\ c\ bound$  **show** *?thesis* **by** (*auto intro*: *dbm-entry-val-mono2*[*folded*  
*less-eq*])  
**qed**  
**next**  
**case** *True*  
**have**  $Lt\ (real-of-int\ (-\ k\ c)) < Le\ 0$  **by** *auto*  
**with**  $True\ c\ assms(3)$  **have**  $?M2\ 0\ (v\ c) = Lt\ (-\ k\ c)$  **unfolding**  
*less norm-def* **by** *auto*  
**moreover** **from** *assms*(1)  $c$  **have**  $Lt\ (-\ k\ c) \leq M1\ 0\ (v\ c)$  **unfolding**  
*normalized'-def* **by** *auto*  
**ultimately** **show** *?thesis* **using**  $le\ c\ bound$  **by** (*auto intro*: *dbm-entry-val-mono2*[*folded*

```

less-eq])
  qed
next
  case (3 c)
  then have c:  $v\ c > 0\ v\ c \leq n\ c \in X\ v'\ (v\ c) = c$  using clock-numbering
v-v' by metis+
  from that c have bound:
    dbm-entry-val u (Some c) None (?M2 (v c) 0)
    unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
  show ?case
  proof (cases M (v c) 0 ≤ Le (k c))
    case False
    with le c have  $\neg M1\ (v\ c)\ 0 \leq Le\ (k\ c)$  by fastforce
  with assms(1) c show ?thesis unfolding normalized'-def by fastforce
  next
    case True
    show ?thesis
    proof (cases M (v c) 0 < Lt 0)
      case T: True
      have  $\neg Le\ (real\ (k\ c)) \prec Lt\ 0$  by auto
      with T True c have  $?M2\ (v\ c)\ 0 = Lt\ 0$  unfolding norm-def less
by (auto simp: Let-def)
      with bound V c show ?thesis unfolding V-def by auto
    next
      case False
      with True assms(3) c have  $?M2\ (v\ c)\ 0 = M\ (v\ c)\ 0$  unfolding
less less-eq norm-def
      by (auto simp: Let-def)
      with dbm-entry-val-mono3[OF bound, folded less-eq] le c show
?thesis by auto
    qed
  qed
next
  case (4 c1 c2)
  then have c:
     $v\ c1 > 0\ v\ c1 \leq n\ c1 \in X\ v'\ (v\ c1) = c1\ v\ c2 > 0\ v\ c2 \leq n$ 
 $c2 \in X\ v'\ (v\ c2) = c2$ 
    using clock-numbering v-v' by metis+
  from that c have bound:
    dbm-entry-val u (Some c1) (Some c2) (?M2 (v c1) (v c2))
    unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
  show ?case
  proof (cases c1 = c2)
    case True

```

```

then have dbm-entry-val u (Some c1) (Some c2) (Le 0) by auto
with c True * dbm-entry-val-mono1[OF this, folded less-eq] show
?thesis by auto
next
case False
with clock-numbering(1) ⟨v c1 ≤ n⟩ ⟨v c2 ≤ n⟩ have neq: v c1 ≠ v
c2 by auto
show ?thesis
proof (cases Le (k c1) < M (v c1) (v c2))
case False
show ?thesis
proof (cases M (v c1) (v c2) < Lt (- real (k c2)))
case F: False
with c False assms(3) neq have
?M2 (v c1) (v c2) = M (v c1) (v c2)
unfolding norm-def norm-diag-def less by simp
with dbm-entry-val-mono1[OF bound, folded less-eq] le c neq
show ?thesis by auto
next
case True
with c False assms(3) neq have ?M2 (v c1) (v c2) = Lt (- k
c2)
unfolding less norm-def by simp
moreover from assms(1) c have M1 (v c1) (v c2) = ∞ ∨ M1
(v c1) (v c2) ≥ Lt (- k c2)
using neq unfolding normalized'-def by fastforce
ultimately show ?thesis using dbm-entry-val-mono1[OF bound,
folded less-eq] by auto
qed
next
case True
with le c neq have M1 (v c1) (v c2) > Le (k c1) by fastforce
moreover from True c assms(3) neq have ?M2 (v c1) (v c2) =
∞
unfolding norm-def less by simp
moreover from assms(1) c have M1 (v c1) (v c2) = ∞ ∨ M1 (v
c1) (v c2) ≤ Le (k c1)
using neq unfolding normalized'-def by fastforce
ultimately show ?thesis by auto
qed
qed
qed
qed
then show ?thesis by blast

```

qed

**lemma** *apx-norm-eq*:

**assumes** *canonical*  $M\ n\ [M]_{v,n} \subseteq V\ \text{dbm-int}\ M\ n$

**shows**  $\text{Approx}_\beta ([M]_{v,n}) = [\text{norm}\ M\ (k\ o\ v')\ n]_{v,n}$

**proof** –

**let**  $?M = \text{norm}\ M\ (k\ o\ v')\ n$

**from** *assms norm-V-preservation norm-int-preservation norm-normalizes'*

**have** \*:

$\text{vabstr} ([?M]_{v,n})\ ?M\ \text{normalized}'\ ?M\ [?M]_{v,n} \subseteq V$

**by** *auto*

**from** *dbm-regions'[OF this] obtain U where U: U ⊆ R [?M]\_{v,n} = ∪ U*

**by** *auto*

**from** *assms(3) have \*\*: [M]\_{v,n} ⊆ [?M]\_{v,n} by (simp add: norm-mono clock-numbering(1) subsetI)*

**show** *?thesis*

**proof** (*cases [M]\_{v,n} = {}*)

**case** *True*

**from** *canonical-empty-zone-spec[OF <canonical M n>] True obtain i*

**where** *i*:

$i \leq n\ M\ i\ i < 0$

**by** *auto*

**then** **have**  $?M\ i\ i < 0$

**unfolding** *norm-def norm-diag-def by (auto simp: DBM.neutral DBM.less)*

**from** *neg-diag-empty[of n v i ?M, OF - <i ≤ n> this] clock-numbering*

**have**

$[?M]_{v,n} = \{\}$

**by** (*auto intro: Lt-lt-LeI*)

**with** *apx-empty True show ?thesis by auto*

**next**

**case** *False*

**from** *apx-in[OF assms(2)] obtain U' M1 where U':*

$\text{Approx}_\beta ([M]_{v,n}) = \bigcup U'\ U' \subseteq \mathcal{R}\ [M]_{v,n} \subseteq \text{Approx}_\beta ([M]_{v,n})$

$\text{vabstr} (\text{Approx}_\beta ([M]_{v,n}))\ M1\ \text{normalized}\ M1$

**by** *auto*

**from** *norm-min[OF - - assms(1) False assms(2)] U'(3,4,5) \*(1) apx-min'[OF U(2,1) - - \*(2) \*\*]*

**show** *?thesis*

**by** (*auto dest!: normalized-normalized'*)

qed

qed

end

## 4.5 Auxiliary $\beta$ -boundedness Theorems

context *Beta-Regions'*

begin

lemma  *$\beta$ -boundedness-diag-lt*:

fixes  $m :: int$

assumes  $-k y \leq m \ m \leq k x \ x \in X \ y \in X$

shows  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x - u y < m\}$

proof –

note  $A = assms$

note  $B = A(1,2)$

let  $?U = \{R \in \mathcal{R}. \exists I J r c d (e :: int). R = region \ X \ I \ J \ r \ \wedge \ valid-region \ X \ k \ I \ J \ r \ \wedge$

$(I x = Const \ c \ \wedge \ I y = Const \ d \ \wedge \ real \ c - d < m \ \vee$

$I x = Const \ c \ \wedge \ I y = Intv \ d \ \wedge \ real \ c - d \leq m \ \vee$

$I x = Intv \ c \ \wedge \ I y = Const \ d \ \wedge \ real \ c + 1 - d \leq m \ \vee$

$I x = Intv \ c \ \wedge \ I y = Intv \ d \ \wedge \ real \ c - d \leq m \ \wedge \ (x,y) \in r \ \wedge \ (y,x) \notin r \ \vee$

$I x = Intv \ c \ \wedge \ I y = Intv \ d \ \wedge \ real \ c - d < m \ \wedge \ (y,x) \in r \ \vee$

$(I x = Greater \ (k \ x) \ \vee \ I y = Greater \ (k \ y)) \ \wedge \ J \ x \ y = Smaller' \ (-k \ y) \ \vee$

$(I x = Greater \ (k \ x) \ \vee \ I y = Greater \ (k \ y)) \ \wedge \ J \ x \ y = Intv' \ e \ \wedge \ e <$

$m \ \vee$

$(I x = Greater \ (k \ x) \ \vee \ I y = Greater \ (k \ y)) \ \wedge \ J \ x \ y = Const' \ e \ \wedge \ e <$

$m$

$)\}$

$\{ \text{fix } u \ I \ J \ r \ \text{assume } u \in region \ X \ I \ J \ r \ I x = Greater \ (k \ x) \ \vee \ I y = Greater \ (k \ y)$

$\text{with } A(3,4) \ \text{have } intv'\text{-elem } x \ y \ u \ (J \ x \ y) \ \text{by force}$

$\} \ \text{note } * = this$

$\{ \text{fix } u \ I \ J \ r \ \text{assume } u \in region \ X \ I \ J \ r$

$\text{with } A(3,4) \ \text{have } intv\text{-elem } x \ u \ (I \ x) \ intv\text{-elem } y \ u \ (I \ y) \ \text{by force+}$

$\} \ \text{note } ** = this$

$\text{have } \bigcup ?U = \{u \in V. u x - u y < m\}$

proof (*safe, goal-cases*)

case (2  $u$ ) with  $**[OF \ this(1)]$  show *?case* by auto

next

case (4  $u$ ) with  $**[OF \ this(1)]$  show *?case* by auto

next

case (6  $u$ ) with  $**[OF \ this(1)]$  show *?case* by auto

next

case (8  $u \ X \ I \ J \ r \ c \ d$ )

from *this*  $A(3,4)$  have *intv-elem*  $x \ u \ (I \ x) \ intv\text{-elem } y \ u \ (I \ y) \ frac \ (u$

```

x) < frac (u y) by force+
  with nat-intv-frac-decomp 8(4,5) have
    u x = c + frac (u x) u y = d + frac (u y) frac (u x) < frac (u y)
  by force+
  with 8(6) show ?case by linarith
next
case (10 u X I J r c d)
with **[OF this(1)] 10(4,5) have u x < c + 1 d < u y by auto
then have u x - u y < real (c + 1) - real d by linarith
moreover from 10(6) have real c + 1 - d ≤ m
proof -
  have int c - int d < m
    using 10(6) by linarith
  then show ?thesis
    by simp
qed
ultimately show ?case by linarith
next
case 12 with *[OF this(1)] B show ?case by auto
next
case 14 with *[OF this(1)] B show ?case by auto
next
case (23 u)
from region-cover-V[OF this(1)] obtain R where R: R ∈ ℛ u ∈ R by
auto
then obtain I J r where R': R = region X I J r valid-region X k I J r
unfolding ℛ-def by auto
with R' R(2) A have C:
  intv-elem x u (I x) intv-elem y u (I y) valid-intv (k x) (I x) valid-intv
(k y) (I y)
  by auto
{ assume A: I x = Greater (k x) ∨ I y = Greater (k y)
  obtain intv and d :: int where intv:
    valid-intv' (k y) (k x) intv intv'-elem x y u intv
    intv = Smaller' (- k y) ∨ intv = Intv' d ∧ d < m ∨ intv = Const'
d ∧ d < m
  proof (cases u x - u y < - int (k y))
    case True
      have valid-intv' (k y) (k x) (Smaller' (- k y)) ..
      moreover with True have intv'-elem x y u (Smaller' (- k y)) by
auto
      ultimately show thesis by (auto intro: that)
    case False

```

```

show thesis
proof (cases  $\exists (c :: \text{int}). u x - u y = c$ )
  case True
  then obtain  $c :: \text{int}$  where  $c: u x - u y = c$  by auto
  have  $\text{valid-intv}' (k y) (k x) (\text{Const}' c)$  using  $\text{False } B(2) \ 23(2) \ c$ 
by fastforce
  moreover with  $c$  have  $\text{intv}'\text{-elem } x y u (\text{Const}' c)$  by auto
  moreover have  $c < m$  using  $c \ 23(2)$  by auto
  ultimately show thesis by (auto intro: that)
next
  case False
  then obtain  $c :: \text{real}$  where  $c: u x - u y = c \ c \notin \mathbb{Z}$  by (metis
Ints-cases)
  have  $\text{valid-intv}' (k y) (k x) (\text{Intv}' (\text{floor } c))$ 
proof
  show  $-\text{int } (k y) \leq \lfloor c \rfloor$  using  $\langle \neg - < \rightarrow \rangle c$  by linarith
  show  $\lfloor c \rfloor < \text{int } (k x)$  using  $B(2) \ 23(2) \ c$  by linarith
qed
  moreover have  $\text{intv}'\text{-elem } x y u (\text{Intv}' (\text{floor } c))$ 
proof
  from  $c(1,2)$  show  $\lfloor c \rfloor < u x - u y$  by (meson  $\text{False eq-iff not-le}$ 
of-int-floor-le)
  from  $c(1,2)$  show  $u x - u y < \lfloor c \rfloor + 1$  by simp
qed
  moreover have  $\lfloor c \rfloor < m$  using  $c \ 23(2)$  by linarith
  ultimately show thesis using that by auto
qed
qed
let  $?J = \lambda a b. \text{if } x = a \wedge y = b \text{ then intv else } J a b$ 
let  $?R = \text{region } X I ?J r$ 
let  $?X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}$ 
have  $u \in ?R$ 
proof (standard, goal-cases)
  case 1 from  $R R'$  show  $?case$  by auto
next
  case 2 from  $R R'$  show  $?case$  by auto
next
  case 3 show  $?X_0 = ?X_0$  by auto
next
  case 4 from  $R R'$  show  $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow \text{frac } (u$ 
 $x) \leq \text{frac } (u y)$  by auto
next
  case 5
  show  $?case$ 

```

```

proof (clarify, goal-cases)
  case (1 a b)
  show ?case
  proof (cases  $x = a \wedge y = b$ )
    case True with intv show ?thesis by auto
  next
    case False
    with  $R(2)$   $R'(1)$  1 show ?thesis by force
  qed
qed
qed
have valid-region  $X$   $k$   $I$  ? $J$   $r$ 
proof
  show ? $X_0 = ?X_0$  ..
  show  $r \subseteq ?X_0 \times ?X_0$  using  $R'$  by auto
  show refl-on ? $X_0$   $r$  using  $R'$  by auto
  show trans  $r$  using  $R'$  by auto
  show total-on ? $X_0$   $r$  using  $R'$  by auto
  show  $\forall x \in X. \text{valid-intv } (k\ x) (I\ x)$  using  $R'$  by auto
  show  $\forall xa \in X. \forall ya \in X. \text{isGreater } (I\ xa) \vee \text{isGreater } (I\ ya)$ 
     $\rightarrow \text{valid-intv}' (\text{int } (k\ ya)) (\text{int } (k\ xa))$  (if  $x = xa \wedge y = ya$  then
intv else  $J\ xa\ ya$ )
  proof (clarify, goal-cases)
    case (1 a b)
    show ?case
    proof (cases  $x = a \wedge y = b$ )
      case True
      with  $B$  intv show ?thesis by auto
    next
      case False
      with  $R'(2)$  1 show ?thesis by force
    qed
  qed
moreover then have ? $R \in \mathcal{R}$  unfolding  $\mathcal{R}$ -def by auto
ultimately have ? $R \in ?U$  using intv
  apply clarify
  apply (rule  $\text{exI}[\text{where } x = I]$ , rule  $\text{exI}[\text{where } x = ?J]$ , rule
exI}[\text{where } x = r])
  using  $A$  by fastforce
with  $\langle u \in \text{region} \dots \rangle$  have ?case by (intro Complete-Lattices.UnionI)
blast+
} note * = this
show ?case

```

```

proof (cases I x)
  case (Const c)
  show ?thesis
  proof (cases I y, goal-cases)
    case (1 d)
    with C(1,2) Const A(2,3) 23(2) have real c - real d < m by auto
    with Const 1 R R' show ?thesis by blast
  next
  case (Intv d)
  with C(1,2) Const A(2,3) 23(2) have real c - (d + 1) < m by
auto
  then have c < 1 + (d + m) by linarith
  then have real c - d ≤ m by simp
  with Const Intv R R' show ?thesis by blast
  next
  case (Greater d) with * C(4) show ?thesis by auto
  qed
next
  case (Intv c)
  show ?thesis
  proof (cases I y, goal-cases)
    case (Const d)
    with C(1,2) Intv A(2,3) 23(2) have real c - d < m by auto
    then have real c < m + d by linarith
    then have c < m + d by linarith
    then have real c + 1 - d ≤ m by simp
    with Const Intv R R' show ?thesis by blast
  next
  case (2 d)
  show ?thesis
  proof (cases (y, x) ∈ r)
    case True
    with C(1,2) R R' Intv 2 A(3,4) have
      c < u x u x < c + 1 d < u y u y < d + 1 frac (u x) ≥ frac (u y)
    by force+
    with 23(2) nat-intv-frac-decomp have c + frac (u x) - (d + frac
(u y)) < m by auto
    with ⟨frac - ≥ -⟩ have real c - real d < m by linarith
    with Intv 2 True R R' show ?thesis by blast
  next
  case False
  with R R' A(3,4) Intv 2 have (x,y) ∈ r by fastforce
  with C(1,2) R R' Intv 2 have c < u x u y < d + 1 by force+
  with 23(2) have c < 1 + d + m by auto

```

```

    then have real  $c - d \leq m$  by simp
    with Intv 2 False <- ∈ r> R R' show ?thesis by blast
  qed
next
  case (Greater d) with * C(4) show ?thesis by auto
  qed
next
  case (Greater d) with * C(3) show ?thesis by auto
  qed
qed (auto intro: A simp: V-def, (fastforce dest!: *)+)
moreover have  $?U \subseteq \mathcal{R}$  by fastforce
ultimately show ?thesis by blast
qed

```

lemma  $\beta$ -boundedness-diag-eq:

```

  fixes m :: int
  assumes - k y ≤ m m ≤ k x x ∈ X y ∈ X
  shows ∃ U ⊆ ℛ. ⋃ U = {u ∈ V. u x - u y = m}
proof -
  note A = assms
  note B = A(1,2)
  let ?U = {R ∈ ℛ. ∃ I J r c d (e :: int). R = region X I J r ∧ valid-region
X k I J r ∧
  (I x = Const c ∧ I y = Const d ∧ real c - d = m ∨
  I x = Intv c ∧ I y = Intv d ∧ real c - d = m ∧ (x, y) ∈ r ∧ (y, x)
∈ r ∨
  (I x = Greater (k x) ∨ I y = Greater (k y)) ∧ J x y = Const' e ∧ e =
m
  )}
  { fix u I J r assume u ∈ region X I J r I x = Greater (k x) ∨ I y =
Greater (k y)
  with A(3,4) have intv'-elem x y u (J x y) by force
  } note * = this
  { fix u I J r assume u ∈ region X I J r
  with A(3,4) have intv-elem x u (I x) intv-elem y u (I y) by force+
  } note ** = this
  have ⋃ ?U = {u ∈ V. u x - u y = m}
proof (safe, goal-cases)
  case (2 u) with **[OF this(1)] show ?case by auto
next
  case (4 u X I J r c d)
  from this A(3,4) have intv-elem x u (I x) intv-elem y u (I y) frac (u
x) = frac (u y) by force+
  with nat-intv-frac-decomp 4(4,5) have

```

```

    u x = c + frac (u x) u y = d + frac (u y) frac (u x) = frac (u y)
  by force+
  with 4(6) show ?case by linarith
next
  case (9 u)
  from region-cover-V[OF this(1)] obtain R where R: R ∈ ℛ u ∈ R by
auto
  then obtain I J r where R': R = region X I J r valid-region X k I J r
unfolding ℛ-def by auto
  with R' R(2) A have C:
    intv-elem x u (I x) intv-elem y u (I y) valid-intv (k x) (I x) valid-intv
(k y) (I y)
  by auto
  { assume A: I x = Greater (k x) ∨ I y = Greater (k y)
  obtain intv where intv:
    valid-intv' (k y) (k x) intv intv'-elem x y u intv intv = Const' m
  proof (cases u x - u y < - int (k y))
    case True
      with 9 B show ?thesis by auto
    next
      case False
      show thesis
      proof (cases ∃ (c :: int). u x - u y = c)
        case True
          then obtain c :: int where c: u x - u y = c by auto
          have valid-intv' (k y) (k x) (Const' c) using False B(2) 9(2) c by
fastforce
          moreover with c have intv'-elem x y u (Const' c) by auto
          moreover have c = m using c 9(2) by auto
          ultimately show thesis by (auto intro: that)
        next
          case False
          then have u x - u y ∉ ℤ by (metis Ints-cases)
          with 9 show ?thesis by auto
        qed
      qed
    let ?J = λ a b. if x = a ∧ y = b then intv else J a b
    let ?R = region X I ?J r
    let ?X0 = {x ∈ X. ∃ d. I x = Intv d}
    have u ∈ ?R
    proof (standard, goal-cases)
      case 1 from R R' show ?case by auto
    next
      case 2 from R R' show ?case by auto
  }

```

```

next
  case 3 show  $?X_0 = ?X_0$  by auto
next
  case 4 from  $R R'$  show  $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow \text{frac}(u$ 
 $x) \leq \text{frac}(u y)$  by auto
next
  case 5
  show ?case
  proof (clarify, goal-cases)
    case (1 a b)
    show ?case
    proof (cases  $x = a \wedge y = b$ )
      case True with intv show ?thesis by auto
    next
      case False with  $R(2) R'(1) 1$  show ?thesis by force
    qed
  qed
qed
have valid-region  $X k I ?J r$ 
proof (rule valid-region.intros)
  show  $?X_0 = ?X_0$  ..
  show  $r \subseteq ?X_0 \times ?X_0$  using  $R'$  by auto
  show refl-on  $?X_0 r$  using  $R'$  by auto
  show trans  $r$  using  $R'$  by auto
  show total-on  $?X_0 r$  using  $R'$  by auto
  show  $\forall x \in X. \text{valid-intv}(k x) (I x)$  using  $R'$  by auto
next
  show  $\forall xa \in X. \forall ya \in X. \text{isGreater}(I xa) \vee \text{isGreater}(I ya) \longrightarrow$ 
   $\text{valid-intv}'(\text{int}(k ya))(\text{int}(k xa))$  (if  $x = xa \wedge y = ya$  then intv
else  $J xa ya$ )
  proof (clarify, goal-cases)
    case (1 a b)
    show ?case
    proof (cases  $x = a \wedge y = b$ )
      case True with  $B$  intv show ?thesis by auto
    next
      case False with  $R'(2) 1$  show ?thesis by force
    qed
  qed
qed
moreover then have  $?R \in \mathcal{R}$  unfolding  $\mathcal{R}$ -def by auto
ultimately have  $?R \in ?U$  using intv
apply clarify
  apply (rule exI[where  $x = I$ ], rule exI[where  $x = ?J$ ], rule

```

```

exI[where x = r]
  using A by fastforce
  with ⟨u ∈ region - - -⟩ have ?case by (intro Complete-Lattices.UnionI)
blast+
} note * = this
show ?case
proof (cases I x)
  case (Const c)
  show ?thesis
  proof (cases I y, goal-cases)
    case (1 d)
    with C(1,2) Const A(2,3) 9(2) have real c - d = m by auto
    with Const 1 R R' show ?thesis by blast
  next
  case (Intv d)
  from Intv Const C(1,2) have range: d < u y u y < d + 1 and eq:
u x = c by auto
  from eq have u x ∈ ℤ by auto
  with nat-intv-not-int[OF range] have u x - u y ∉ ℤ using Ints-diff
by fastforce
  with 9 show ?thesis by auto
  next
  case Greater with C * show ?thesis by auto
qed
next
case (Intv c)
show ?thesis
proof (cases I y, goal-cases)
  case (Const d)
  from Intv Const C(1,2) have range: c < u x u x < c + 1 and eq:
u y = d by auto
  from eq have u y ∈ ℤ by auto
  with nat-intv-not-int[OF range] have u x - u y ∉ ℤ using Ints-add
by fastforce
  with 9 show ?thesis by auto
  next
  case (2 d)
  with Intv C have range: c < u x u x < c + 1 d < u y u y < d + 1
by auto
  show ?thesis
  proof (cases (x, y) ∈ r)
    case True
    note T = this
    show ?thesis

```

```

proof (cases (y, x) ∈ r)
  case True
    with Intv 2 T R' ⟨u ∈ R⟩ A(3,4) have frac (u x) = frac (u y)
by force
  with nat-intv-frac-decomp[OF range(1,2)] nat-intv-frac-decomp[OF
range(3,4)] have
    u x - u y = real c - real d
  by algebra
  with 9 have real c - d = m by auto
  with T True Intv 2 R R' show ?thesis by force
next
  case False
    with Intv 2 T R' ⟨u ∈ R⟩ A(3,4) have frac (u x) < frac (u y)
by force
  then have
    frac (u x - u y) ≠ 0
  by (metis add.left-neutral diff-add-cancel frac-add frac-unique-iff
less-irrefl)
  then have u x - u y ∉ ℤ by (metis frac-eq-0-iff)
  with 9 show ?thesis by auto
qed
next
  case False
  note F = this
  show ?thesis
  proof (cases x = y)
    case True
      with R'(2) Intv ⟨x ∈ X⟩ have (x, y) ∈ r (y, x) ∈ r by (auto
simp: refl-on-def)
      with Intv True R' R 9(2) show ?thesis by force
    next
      case False
      with F R'(2) Intv 2 ⟨x ∈ X⟩ ⟨y ∈ X⟩ have (y, x) ∈ r by (fastforce
simp: total-on-def)
      with F Intv 2 R' ⟨u ∈ R⟩ A(3,4) have frac (u x) > frac (u y)
by force
    then have
      frac (u x - u y) ≠ 0
    by (metis add.left-neutral diff-add-cancel frac-add frac-unique-iff
less-irrefl)
    then have u x - u y ∉ ℤ by (metis frac-eq-0-iff)
    with 9 show ?thesis by auto
  qed
qed

```

```

next
  case Greater with * C show ?thesis by force
qed
next
  case Greater with * C show ?thesis by force
qed
qed (auto intro: A simp: V-def dest: *)
moreover have ?U  $\subseteq$   $\mathcal{R}$  by fastforce
ultimately show ?thesis by blast
qed

```

**lemma**  $\beta$ -boundedness-1t:

```

fixes m :: int
assumes m  $\leq$  k x x  $\in$  X
shows  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x < m\}$ 
proof -
  note A = assms
  let ?U = {R  $\in$   $\mathcal{R}$ .  $\exists I J r c. R = \text{region } X I J r \wedge \text{valid-region } X k I J r$ 
 $\wedge$ 
  (I x = Const c  $\wedge$  c < m  $\vee$  I x = Intv c  $\wedge$  c < m)}
  { fix u I J r assume u  $\in$  region X I J r
    with A have intv-elem x u (I x) by force+
  } note ** = this
  have  $\bigcup ?U = \{u \in V. u x < m\}$ 
  proof (safe, goal-cases)
    case (2 u) with **[OF this(1)] show ?case by auto
  next
    case (4 u) with **[OF this(1)] show ?case by auto
  next
    case (5 u)
    from region-cover-V[OF this(1)] obtain R where R: R  $\in$   $\mathcal{R}$  u  $\in$  R by
  auto
    then obtain I J r where R': R = region X I J r valid-region X k I J r
  unfolding  $\mathcal{R}$ -def by auto
    with R' R(2) A have C:
      intv-elem x u (I x) valid-intv (k x) (I x)
    by auto
    show ?case
  proof (cases I x)
    case (Const c)
      with 5 C(1) have c < m by auto
      with R R' Const show ?thesis by blast
  next
    case (Intv c)

```

```

    with 5 C(1) have c < m by auto
    with R R' Intv show ?thesis by blast
  next
    case (Greater c) with 5 C A Greater show ?thesis by auto
  qed
qed (auto intro: A simp: V-def)
moreover have ?U ⊆ R by fastforce
ultimately show ?thesis by blast
qed

```

**lemma**  $\beta$ -boundedness-gt:

```

  fixes m :: int
  assumes m ≤ k x x ∈ X
  shows ∃ U ⊆ R. ∪ U = {u ∈ V. u x > m}
proof –
  note A = assms
  let ?U = {R ∈ R. ∃ I J r c. R = region X I J r ∧ valid-region X k I J r
  ∧
  (I x = Const c ∧ c > m ∨ I x = Intv c ∧ c ≥ m ∨ I x = Greater (k
  x))}
  { fix u I J r assume u ∈ region X I J r
    with A have intv-elem x u (I x) by force+
  } note ** = this
  have ∪ ?U = {u ∈ V. u x > m}
proof (safe, goal-cases)
  case (2 u) with **[OF this(1)] show ?case by auto
  next
  case (4 u) with **[OF this(1)] show ?case by auto
  next
  case (6 u) with A **[OF this(1)] show ?case by auto
  next
  case (7 u)
  from region-cover-V[OF this(1)] obtain R where R: R ∈ R u ∈ R by
  auto
  then obtain I J r where R': R = region X I J r valid-region X k I J r
unfolding R-def by auto
  with R' R(2) A have C:
    intv-elem x u (I x) valid-intv (k x) (I x)
  by auto
  show ?case
proof (cases I x)
  case (Const c)
  with 7 C(1) have c > m by auto
  with R R' Const show ?thesis by blast

```

```

next
  case (Intv c)
  with  $\gamma C(1)$  have  $c \geq m$  by auto
  with  $R R' Intv$  show ?thesis by blast
next
  case (Greater c)
  with  $C$  have  $k x = c$  by auto
  with  $R R' Greater$  show ?thesis by blast
qed
qed (auto intro: A simp: V-def)
moreover have  $?U \subseteq \mathcal{R}$  by fastforce
ultimately show ?thesis by blast
qed

```

lemma  $\beta$ -boundedness-eq:

```

fixes  $m :: int$ 
assumes  $m \leq k x x \in X$ 
shows  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x = m\}$ 
proof -
  note  $A = assms$ 
  let  $?U = \{R \in \mathcal{R}. \exists I J r c. R = region\ X\ I\ J\ r \wedge valid-region\ X\ k\ I\ J\ r$ 
 $\wedge I x = Const\ c \wedge c = m\}$ 
  have  $\bigcup ?U = \{u \in V. u x = m\}$ 
  proof (safe, goal-cases)
    case (2 u) with  $A$  show ?case by force
  next
    case (3 u)
    from region-cover-V[OF this(1)] obtain  $R$  where  $R: R \in \mathcal{R} u \in R$  by
    auto
    then obtain  $I J r$  where  $R': R = region\ X\ I\ J\ r\ valid-region\ X\ k\ I\ J\ r$ 
  unfolding  $\mathcal{R}$ -def by auto
    with  $R' R(2) A$  have  $C: intv\ elem\ x\ u\ (I\ x)\ valid\ intv\ (k\ x)\ (I\ x)$  by
    auto
    show ?case
  proof (cases  $I x$ )
    case (Const c)
    with  $\exists C(1)$  have  $c = m$  by auto
    with  $R R' Const$  show ?thesis by blast
  next
    case (Intv c)
    with  $C$  have  $c < u x u x < c + 1$  by auto
    from nat-intv-not-int[OF this]  $\exists$  show ?thesis by auto
  next
    case (Greater c)

```

**with**  $C \exists A$  **show** *?thesis* **by** *auto*  
**qed**  
**qed** (*force intro: A simp: V-def*)  
**moreover have**  $?U \subseteq \mathcal{R}$  **by** *fastforce*  
**ultimately show** *?thesis* **by** *blast*  
**qed**

**lemma**  $\beta$ -boundedness-diag-le:

**fixes**  $m :: int$   
**assumes**  $- k y \leq m \ m \leq k x \ x \in X \ y \in X$   
**shows**  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x - u y \leq m\}$   
**proof** –  
**from**  $\beta$ -boundedness-diag-eq[*OF assms*]  $\beta$ -boundedness-diag-lt[*OF assms*]  
**obtain**  $U1 \ U2$  **where**  $A$ :  
 $U1 \subseteq \mathcal{R} \bigcup U1 = \{u \in V. u x - u y < m\} \ U2 \subseteq \mathcal{R} \bigcup U2 = \{u \in V. u x - u y = m\}$   
**by** *blast*  
**then have**  $\{u \in V. u x - u y \leq m\} = \bigcup (U1 \cup U2) \ U1 \cup U2 \subseteq \mathcal{R}$  **by** *auto*  
**then show** *?thesis* **by** *blast*  
**qed**

**lemma**  $\beta$ -boundedness-le:

**fixes**  $m :: int$   
**assumes**  $m \leq k x \ x \in X$   
**shows**  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x \leq m\}$   
**proof** –  
**from**  $\beta$ -boundedness-lt[*OF assms*]  $\beta$ -boundedness-eq[*OF assms*] **obtain**  
 $U1 \ U2$  **where**  $A$ :  
 $U1 \subseteq \mathcal{R} \bigcup U1 = \{u \in V. u x < m\} \ U2 \subseteq \mathcal{R} \bigcup U2 = \{u \in V. u x = m\}$   
**by** *blast*  
**then have**  $\{u \in V. u x \leq m\} = \bigcup (U1 \cup U2) \ U1 \cup U2 \subseteq \mathcal{R}$  **by** *auto*  
**then show** *?thesis* **by** *blast*  
**qed**

**lemma**  $\beta$ -boundedness-ge:

**fixes**  $m :: int$   
**assumes**  $m \leq k x \ x \in X$   
**shows**  $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u x \geq m\}$   
**proof** –  
**from**  $\beta$ -boundedness-gt[*OF assms*]  $\beta$ -boundedness-eq[*OF assms*] **obtain**  
 $U1 \ U2$  **where**  $A$ :  
 $U1 \subseteq \mathcal{R} \bigcup U1 = \{u \in V. u x > m\} \ U2 \subseteq \mathcal{R} \bigcup U2 = \{u \in V. u x = m\}$

$= m\}$   
**by** *blast*  
**then have**  $\{u \in V. u x \geq m\} = \bigcup (U1 \cup U2) U1 \cup U2 \subseteq \mathcal{R}$  **by** *auto*  
**then show** *?thesis* **by** *blast*  
**qed**

**lemma**  *$\beta$ -boundedness-diag-lt'*:

**fixes**  $m :: int$

**shows**

$- k y \leq (m :: int) \implies m \leq k x \implies x \in X \implies y \in X \implies Z \subseteq \{u \in V. u x - u y < m\}$   
 $\implies Approx_\beta Z \subseteq \{u \in V. u x - u y < m\}$

**proof** (*goal-cases*)

**case** 1

**note**  $A = this$

**from**  *$\beta$ -boundedness-diag-lt*[*OF*  $A(1-4)$ ] **obtain**  $U$  **where**  $U$ :

$U \subseteq \mathcal{R} \{u \in V. u x - u y < m\} = \bigcup U$

**by** *auto*

**from** 1 *clock-numbering* **have**  $*$ :  $v x > 0 \vee y > 0 \vee x \leq n \vee y \leq n$  **by** *auto*

**have**  $**$ :  $\bigwedge c. v c = 0 \implies False$

**proof**  $-$

**fix**  $c$  **assume**  $v c = 0$

**moreover from** *clock-numbering*(1) **have**  $v c > 0$  **by** *auto*

**ultimately show** *False* **by** *auto*

**qed**

**let**  $?M = \lambda i j. if (i = v x \wedge j = v y) then Lt (real-of-int m) else if i = j \vee i = 0 then Le 0 else \infty$

**have**  $\{u \in V. u x - u y < m\} = [?M]_{v,n}$  **unfolding** *DBM-zone-repr-def* *DBM-val-bounded-def*

**using**  $**$  **proof** (*auto*, *goal-cases*)

**case** (1  $u c$ )

**with** *clock-numbering* **have**  $c \in X$  **by** *metis*

**with** 1 **show** *?case* **unfolding** *V-def* **by** *auto*

**next**

**case** (2  $u c1 c2$ )

**with** *clock-numbering*(1) **have**  $x = c1 \ y = c2$  **by** *auto*

**with** 2(5) **show** *?case* **by** *auto*

**next**

**case** (3  $u c1 c2$ )

**with** *clock-numbering*(1) **have**  $c1 = c2$  **by** *auto*

**then show** *?case* **by** *auto*

**next**

**case** (4  $u c1 c2$ )

```

    with clock-numbering(1) have c1 = c2 by auto
    then show ?case by auto
next
  case (5 u c1 c2)
  with clock-numbering(1) have x = c1 y = c2 by auto
  with 5(6) show ?case by auto
next
  case (6 u)
  show ?case unfolding V-def
  proof safe
    fix c assume c ∈ X
    with clock-numbering have v c > 0 v c ≤ n by auto
    with 6(6) show u c ≥ 0 by auto
  qed
next
  case (7 u)
  then have dbm-entry-val u (Some x) (Some y) (Lt (real-of-int m)) by
metis
    then show ?case by auto
  qed
  then have vabstr {u ∈ V. u x - u y < m} ?M by auto
  moreover have normalized ?M unfolding normalized less-eq dbm-le-def
using A v-v' by auto
  ultimately show ?thesis using apx-min[OF U(2,1)] A(5) by blast
qed

```

**lemma** *β-boundedness-diag-le'*:

```

  fixes m :: int
  shows
    - k y ≤ (m :: int) ⇒ m ≤ k x ⇒ x ∈ X ⇒ y ∈ X ⇒ Z ⊆ {u ∈ V.
u x - u y ≤ m}
    ⇒ Approxβ Z ⊆ {u ∈ V. u x - u y ≤ m}
  proof (goal-cases)
    case 1
    note A = this
    from β-boundedness-diag-le[OF A(1-4)] obtain U where U:
      U ⊆ ℛ {u ∈ V. u x - u y ≤ m} = ⋃ U
    by auto
    from 1 clock-numbering have *: v x > 0 v y > 0 v x ≤ n v y ≤ n by
auto
    have **: ⋀ c. v c = 0 ⇒ False
  proof -
    fix c assume v c = 0
    moreover from clock-numbering(1) have v c > 0 by auto
  
```

```

    ultimately show False by auto
  qed
  let ?M =  $\lambda i j.$  if  $(i = v x \wedge j = v y)$  then Le (real-of-int m) else if  $i = j \vee i = 0$  then Le 0 else  $\infty$ 
  have  $\{u \in V. u x - u y \leq m\} = [?M]_{v,n}$  unfolding DBM-zone-repr-def
  DBM-val-bounded-def
  using * **
  proof (auto, goal-cases)
    case (1 u c)
    with clock-numbering have  $c \in X$  by metis
    with 1 show ?case unfolding V-def by auto
  next
    case (2 u c1 c2)
    with clock-numbering(1) have  $x = c1 \ y = c2$  by auto
    with 2(5) show ?case by auto
  next
    case (3 u c1 c2)
    with clock-numbering(1) have  $c1 = c2$  by auto
    then show ?case by auto
  next
    case (4 u c1 c2)
    with clock-numbering(1) have  $c1 = c2$  by auto
    then show ?case by auto
  next
    case (5 u c1 c2)
    with clock-numbering(1) have  $x = c1 \ y = c2$  by auto
    with 5(6) show ?case by auto
  next
    case (6 u)
    show ?case unfolding V-def
    proof safe
      fix c assume  $c \in X$ 
      with clock-numbering have  $v c > 0 \ v c \leq n$  by auto
      with 6(6) show  $u c \geq 0$  by auto
    qed
  next
    case (7 u)
    then have dbm-entry-val u (Some x) (Some y) (Le (real-of-int m)) by
  metis
    then show ?case by auto
  qed
  then have vabstr  $\{u \in V. u x - u y \leq m\}$  ?M by auto
  moreover have normalized ?M unfolding normalized less-eq dbm-le-def
  using A v-v' by auto

```

**ultimately show** *?thesis* **using** *apx-min[OF U(2,1)] A(5)* **by** *blast*  
**qed**

**lemma**  *$\beta$ -boundedness-lt'*:

**fixes** *m :: int*

**shows**

$m \leq k \ x \implies x \in X \implies Z \subseteq \{u \in V. u \ x < m\} \implies \text{Approx}_\beta Z \subseteq \{u \in V. u \ x < m\}$

**proof** (*goal-cases*)

**case** *1*

**note** *A = this*

**from**  *$\beta$ -boundedness-lt[OF A(1,2)]* **obtain** *U* **where** *U: U  $\subseteq$   $\mathcal{R}$  {u  $\in$  V. u x < m} =  $\bigcup$  U* **by** *auto*

**from** *1 clock-numbering* **have** *\*: v x > 0 v x  $\leq$  n* **by** *auto*

**have** *\*\*:*  $\bigwedge c. v \ c = 0 \implies \text{False}$

**proof** *-*

**fix** *c* **assume** *v c = 0*

**moreover from** *clock-numbering(1)* **have** *v c > 0* **by** *auto*

**ultimately show** *False* **by** *auto*

**qed**

**let** *?M =  $\lambda i \ j. \text{if } (i = v \ x \wedge j = 0) \text{ then } \text{Lt } (\text{real-of-int } m) \text{ else if } i = j \vee i = 0 \text{ then } \text{Le } 0 \text{ else } \infty$*

**have**  $\{u \in V. u \ x < m\} = [?M]_{v,n}$  **unfolding** *DBM-zone-repr-def DBM-val-bounded-def*  
**using** *\*\**

**proof** (*auto, goal-cases*)

**case** (*1 u c*)

**with** *clock-numbering* **have** *c  $\in$  X* **by** *metis*

**with** *1* **show** *?case* **unfolding** *V-def* **by** *auto*

**next**

**case** (*2 u c1*)

**with** *clock-numbering(1)* **have** *x = c1* **by** *auto*

**with** *2(4)* **show** *?case* **by** *auto*

**next**

**case** (*3 u c*)

**with** *clock-numbering* **have** *c  $\in$  X* **by** *metis*

**with** *3* **show** *?case* **unfolding** *V-def* **by** *auto*

**next**

**case** (*4 u c1 c2*)

**with** *clock-numbering(1)* **have** *c1 = c2* **by** *auto*

**then show** *?case* **by** *auto*

**next**

**case** (*5 u*)

**show** *?case* **unfolding** *V-def*

**proof** *safe*

```

    fix c assume c ∈ X
    with clock-numbering have v c > 0 v c ≤ n by auto
    with 5(4) show u c ≥ 0 by auto
  qed
qed
then have vabstr {u ∈ V. u x < m} ?M by auto
moreover have normalized ?M unfolding normalized less-eq dbm-le-def
using A v-v' by auto
ultimately show ?thesis using apx-min[OF U(2,1)] A(3) by blast
qed

```

**lemma** *β-boundedness-gt'*:

```

  fixes m :: int
  shows
    m ≤ k x ⇒ x ∈ X ⇒ Z ⊆ {u ∈ V. u x > m} ⇒ Approxβ Z ⊆ {u ∈
    V. u x > m}
  proof goal-cases
    case 1
    from β-boundedness-gt[OF this(1,2)] obtain U where U: U ⊆ R {u ∈
    V. u x > m} = ⋃ U by auto
    from 1 clock-numbering have *: v x > 0 v x ≤ n by auto
    have **: ⋀ c. v c = 0 ⇒ False
    proof –
      fix c assume v c = 0
      moreover from clock-numbering(1) have v c > 0 by auto
      ultimately show False by auto
    qed
    obtain M where vabstr {u ∈ V. u x > m} M normalized M
    proof (cases m ≥ 0)
      case True
      let ?M = λ i j. if (i = 0 ∧ j = v x) then Lt (−real-of-int m) else if i
      = j ∨ i = 0 then Le 0 else ∞
      have {u ∈ V. u x > m} = [?M]v,n unfolding DBM-zone-repr-def
      DBM-val-bounded-def
      using * **
      proof (auto, goal-cases)
        case (1 u c)
        with clock-numbering(1) have x = c by auto
        with 1(5) show ?case by auto
      next
        case (2 u c)
        with clock-numbering have c ∈ X bymetis
        with 2 show ?case unfolding V-def by auto
      next
    end
  end

```

```

    case (3 u c1 c2)
    with clock-numbering(1) have c1 = c2 by auto
    then show ?case by auto
next
    case (4 u c1 c2)
    with clock-numbering(1) have c1 = c2 by auto
    then show ?case by auto
next
    case (5 u)
    show ?case unfolding V-def
    proof safe
      fix c assume c ∈ X
      with clock-numbering have c: v c > 0 v c ≤ n by auto
      show u c ≥ 0
      proof (cases v c = v x)
        case False
        with 5(4) c show ?thesis by auto
      next
        case True
        with 5(4) c have - u c < - m by auto
        with ⟨m ≥ 0⟩ show ?thesis by auto
      qed
    qed
  qed
  moreover have normalized ?M unfolding normalized using 1 v-v' by
  auto
  ultimately show ?thesis by (intro that[of ?M]) auto
next
  case False
  then have {u ∈ V. u x > m} = V unfolding V-def using ⟨x ∈ X⟩
  by auto
  with R-union all-dbm that show ?thesis by auto
  qed
  with apx-min[OF U(2,1)] 1(3) show ?thesis by blast
  qed

```

lemma obtains-dbm-le:

```

  fixes m :: int
  assumes x ∈ X m ≤ k x
  obtains M where vabstr {u ∈ V. u x ≤ m} M normalized M
proof -
  from assms clock-numbering have *: v x > 0 v x ≤ n by auto
  have **: ∧ c. v c = 0 ⇒ False
  proof -

```

```

fix  $c$  assume  $v\ c = 0$ 
  moreover from clock-numbering(1) have  $v\ c > 0$  by auto
  ultimately show False by auto
qed
let  $?M = \lambda\ i\ j.$  if ( $i = v\ x \wedge j = 0$ ) then Le (real-of-int  $m$ ) else if  $i = j$ 
 $\vee\ i = 0$  then Le 0 else  $\infty$ 
have  $\{u \in V. u\ x \leq m\} = [?M]_{v,n}$  unfolding DBM-zone-repr-def DBM-val-bounded-def
using * **
proof (auto, goal-cases)
  case (1  $u\ c$ )
    with clock-numbering have  $c \in X$  by metis
    with 1 show  $?case$  unfolding V-def by auto
  next
    case (2  $u\ c1$ )
      with clock-numbering(1) have  $x = c1$  by auto
      with 2(4) show  $?case$  by auto
    next
      case (3  $u\ c$ )
        with clock-numbering have  $c \in X$  by metis
        with 3 show  $?case$  unfolding V-def by auto
      next
        case (4  $u\ c1\ c2$ )
          with clock-numbering(1) have  $c1 = c2$  by auto
          then show  $?case$  by auto
        next
          case (5  $u$ )
            show  $?case$  unfolding V-def
            proof safe
              fix  $c$  assume  $c \in X$ 
              with clock-numbering have  $v\ c > 0\ v\ c \leq n$  by auto
              with 5(4) show  $u\ c \geq 0$  by auto
            qed
          qed
        then have vabstr  $\{u \in V. u\ x \leq m\}\ ?M$  by auto
        moreover have normalized  $?M$  unfolding normalized using assms  $v\ v'$ 
by auto
        ultimately show  $?thesis$  ..
      qed

```

**lemma**  *$\beta$ -boundedness-le'*:

**fixes**  $m :: int$

**shows**

$m \leq k\ x \implies x \in X \implies Z \subseteq \{u \in V. u\ x \leq m\} \implies \text{Approx}_\beta\ Z \subseteq \{u \in$

$V. u x \leq m$   
**proof** (*goal-cases*)  
**case** 1  
**from**  $\beta$ -boundedness-le[OF this(1,2)] **obtain**  $U$  **where**  $U: U \subseteq \mathcal{R} \{u \in V. u x \leq m\} = \bigcup U$  **by** *auto*  
**from** obtains-dbm-le 1 **obtain**  $M$  **where**  $vabstr \{u \in V. u x \leq m\} M$  *normalized*  $M$  **by** *auto*  
**with** *apx-min*[OF  $U(2,1)$ ] 1(3) **show** *?thesis* **by** *blast*  
**qed**

**lemma** *obtains-dbm-ge*:

**fixes**  $m :: int$   
**assumes**  $x \in X \ m \leq k \ x$   
**obtains**  $M$  **where**  $vabstr \{u \in V. u x \geq m\} M$  *normalized*  $M$   
**proof** –  
**from** *assms clock-numbering* **have**  $*: v x > 0 \ v x \leq n$  **by** *auto*  
**have**  $** : \bigwedge c. v c = 0 \implies False$   
**proof** –  
**fix**  $c$  **assume**  $v c = 0$   
**moreover from** *clock-numbering(1)* **have**  $v c > 0$  **by** *auto*  
**ultimately show** *False* **by** *auto*  
**qed**  
**obtain**  $M$  **where**  $vabstr \{u \in V. u x \geq m\} M$  *normalized*  $M$   
**proof** (*cases*  $m \geq 0$ )  
**case** *True*  
**let**  $?M = \lambda i j. \text{if } (i = 0 \wedge j = v x) \text{ then } Le \ (-real-of-int \ m) \text{ else if } i = j \vee i = 0 \text{ then } Le \ 0 \text{ else } \infty$   
**have**  $\{u \in V. u x \geq m\} = [?M]_{v,n}$  **unfolding** *DBM-zone-repr-def*  
*DBM-val-bounded-def*  
**using**  $* **$   
**proof** (*auto, goal-cases*)  
**case** (1  $u \ c$ )  
**with** *clock-numbering(1)* **have**  $x = c$  **by** *auto*  
**with** 1(5) **show** *?case* **by** *auto*  
**next**  
**case** (2  $u \ c$ )  
**with** *clock-numbering* **have**  $c \in X$  **by** *metis*  
**with** 2 **show** *?case* **unfolding** *V-def* **by** *auto*  
**next**  
**case** (3  $u \ c1 \ c2$ )  
**with** *clock-numbering(1)* **have**  $c1 = c2$  **by** *auto*  
**then show** *?case* **by** *auto*  
**next**  
**case** (4  $u \ c1 \ c2$ )

```

    with clock-numbering(1) have  $c1 = c2$  by auto
    then show ?case by auto
next
  case (5 u)
  show ?case unfolding V-def
  proof safe
    fix c assume  $c \in X$ 
    with clock-numbering have  $c: v\ c > 0\ v\ c \leq n$  by auto
    show  $u\ c \geq 0$ 
    proof (cases  $v\ c = v\ x$ )
      case False
        with 5(4) c show ?thesis by auto
      next
        case True
          with 5(4) c have  $-u\ c \leq -m$  by auto
          with  $\langle m \geq 0 \rangle$  show ?thesis by auto
    qed
  qed
  moreover have normalized ?M unfolding normalized using assms
  v-v' by auto
  ultimately show ?thesis by (intro that[of ?M]) auto
next
  case False
  then have  $\{u \in V. u\ x \geq m\} = V$  unfolding V-def using  $\langle x \in X \rangle$ 
  by auto
  with R-union all-dbm that show ?thesis by auto
  qed
  then show ?thesis ..
qed

```

**lemma**  *$\beta$ -boundedness-ge'*:

```

  fixes  $m :: int$ 
  shows  $m \leq k\ x \implies x \in X \implies Z \subseteq \{u \in V. u\ x \geq m\} \implies Approx_\beta Z$ 
   $\subseteq \{u \in V. u\ x \geq m\}$ 
  proof (goal-cases)
    case 1
    from  $\beta$ -boundedness-ge[OF this(1,2)] obtain U where  $U: U \subseteq \mathcal{R}\ \{u \in$ 
   $V. u\ x \geq m\} = \bigcup U$  by auto
    from obtains-dbm-ge 1 obtain M where vabstr  $\{u \in V. u\ x \geq m\}\ M$ 
  normalized M by auto
    with apx-min[OF U(2,1)] 1(3) show ?thesis by blast
  qed

```

end

end

## 5 The Classic Construction for Decidability

```
theory Regions
imports Timed-Automata TA-Misc
begin
```

The following is a formalization of regions in the correct version of Patricia Bouyer et al.

### 5.1 Definition of Regions

```
type-synonym 'c ceiling = ('c  $\Rightarrow$  nat)
```

```
datatype intv =
  Const nat |
  Intv nat |
  Greater nat
```

```
type-synonym t = real
```

```
inductive valid-intv :: nat  $\Rightarrow$  intv  $\Rightarrow$  bool
```

```
where
```

```
   $0 \leq d \Longrightarrow d \leq c \Longrightarrow \text{valid-intv } c \text{ (Const } d)$  |
   $0 \leq d \Longrightarrow d < c \Longrightarrow \text{valid-intv } c \text{ (Intv } d)$  |
  valid-intv c (Greater c)
```

```
inductive intv-elem :: 'c  $\Rightarrow$  ('c,t) cval  $\Rightarrow$  intv  $\Rightarrow$  bool
```

```
where
```

```
   $u x = d \Longrightarrow \text{intv-elem } x u \text{ (Const } d)$  |
   $d < u x \Longrightarrow u x < d + 1 \Longrightarrow \text{intv-elem } x u \text{ (Intv } d)$  |
   $c < u x \Longrightarrow \text{intv-elem } x u \text{ (Greater } c)$ 
```

```
abbreviation total-preorder r  $\equiv$  refl r  $\wedge$  trans r
```

```
inductive valid-region :: 'c set  $\Rightarrow$  ('c  $\Rightarrow$  nat)  $\Rightarrow$  ('c  $\Rightarrow$  intv)  $\Rightarrow$  'c rel  $\Rightarrow$ 
bool
```

```
where
```

```
   $\llbracket X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}; r \subseteq X_0 \times X_0; \text{refl-on } X_0 r; \text{trans } r;$ 
  total-on X0 r;
```

$\forall x \in X. \text{valid-intv } (k \ x) \ (I \ x)]$   
 $\implies \text{valid-region } X \ k \ I \ r$

**inductive-set** *region* for  $X \ I \ r$

**where**

$\forall x \in X. u \ x \geq 0 \implies \forall x \in X. \text{intv-elem } x \ u \ (I \ x) \implies X_0 = \{x \in X.$   
 $\exists d. I \ x = \text{Intv } d\} \implies$   
 $\forall x \in X_0. \forall y \in X_0. (x, y) \in r \longleftrightarrow \text{frac } (u \ x) \leq \text{frac } (u \ y)$   
 $\implies u \in \text{region } X \ I \ r$

Defining the unique element of a partition that contains a valuation

**definition** *part* ( $\langle[-]\rangle$  [61,61] 61) **where** *part*  $v \ \mathcal{R} \equiv \text{THE } R. R \in \mathcal{R} \wedge v \in R$

**inductive-set** *Succ* for  $\mathcal{R} \ R$  **where**

$u \in R \implies R \in \mathcal{R} \implies R' \in \mathcal{R} \implies t \geq 0 \implies R' = [u \oplus t]_{\mathcal{R}} \implies R' \in \text{Succ } \mathcal{R} \ R$

First we need to show that the set of regions is a partition of the set of all clock assignments. This property is only claimed by P. Bouyer.

**inductive-cases**[*elim!*]: *intv-elem*  $x \ u \ (\text{Const } d)$   
**inductive-cases**[*elim!*]: *intv-elem*  $x \ u \ (\text{Intv } d)$   
**inductive-cases**[*elim!*]: *intv-elem*  $x \ u \ (\text{Greater } d)$   
**inductive-cases**[*elim!*]: *valid-intv*  $c \ (\text{Greater } d)$   
**inductive-cases**[*elim!*]: *valid-intv*  $c \ (\text{Const } d)$   
**inductive-cases**[*elim!*]: *valid-intv*  $c \ (\text{Intv } d)$

**declare** *valid-intv.intros*[*intro*]

**declare** *intv-elem.intros*[*intro*]

**declare** *Succ.intros*[*intro*]

**declare** *Succ.cases*[*elim*]

**declare** *region.cases*[*elim*]

**declare** *valid-region.cases*[*elim*]

## 5.2 Basic Properties

First we show that all valid intervals are distinct.

**lemma** *valid-intv-distinct*:

$\text{valid-intv } c \ I \implies \text{valid-intv } c \ I' \implies \text{intv-elem } x \ u \ I \implies \text{intv-elem } x \ u \ I' \implies I = I'$

**by** (*cases*  $I$ ; *cases*  $I'$ ; *auto*)

From this we show that all valid regions are distinct.

**lemma** *valid-regions-distinct*:

$valid\text{-}region\ X\ k\ I\ r \implies valid\text{-}region\ X\ k\ I'\ r' \implies v \in region\ X\ I\ r \implies v \in region\ X\ I'\ r'$   
 $\implies region\ X\ I\ r = region\ X\ I'\ r'$

**proof** *goal-cases*

**case** *A: 1*

{ **fix**  $x$  **assume**  $x: x \in X$   
**with**  $A(1)$  **have**  $valid\text{-}intv\ (k\ x)\ (I\ x)$  **by** *auto*  
**moreover from**  $A(2)$   $x$  **have**  $valid\text{-}intv\ (k\ x)\ (I'\ x)$  **by** *auto*  
**moreover from**  $A(3)$   $x$  **have**  $intv\text{-}elem\ x\ v\ (I\ x)$  **by** *auto*  
**moreover from**  $A(4)$   $x$  **have**  $intv\text{-}elem\ x\ v\ (I'\ x)$  **by** *auto*  
**ultimately have**  $I\ x = I'\ x$  **using** *valid-intv-distinct* **by** *fastforce*  
**}** **note**  $*$  = *this*

**from**  $A$  **show** *?thesis*

**proof** (*safe, goal-cases*)

**case**  $A: (1\ u)$

**have**  $intv\text{-}elem\ x\ u\ (I'\ x)$  **if**  $x \in X$  **for**  $x$  **using**  $A(5)$   $*$  **that** **by** *auto*  
**then have**  $B: \forall x \in X. intv\text{-}elem\ x\ u\ (I'\ x)$  **by** *auto*

**let**  $?X_0 = \{x \in X. \exists d. I'\ x = Intv\ d\}$

{ **fix**  $x\ y$  **assume**  $x: x \in ?X_0$  **and**  $y: y \in ?X_0$   
**have**  $(x, y) \in r' \iff frac\ (u\ x) \leq frac\ (u\ y)$

**proof**

**assume**  $frac\ (u\ x) \leq frac\ (u\ y)$   
**with**  $A(5)$   $x\ y\ *$  **have**  $(x, y) \in r$  **by** *auto*  
**with**  $A(3)$   $x\ y\ *$  **have**  $frac\ (v\ x) \leq frac\ (v\ y)$  **by** *auto*  
**with**  $A(4)$   $x\ y$  **show**  $(x, y) \in r'$  **by** *auto*

**next**

**assume**  $(x, y) \in r'$   
**with**  $A(4)$   $x\ y$  **have**  $frac\ (v\ x) \leq frac\ (v\ y)$  **by** *auto*  
**with**  $A(3)$   $x\ y\ *$  **have**  $(x, y) \in r$  **by** *auto*  
**with**  $A(5)$   $x\ y\ *$  **show**  $frac\ (u\ x) \leq frac\ (u\ y)$  **by** *auto*

**qed**

**}**

**then have**  $*$ :  $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r' \iff frac\ (u\ x) \leq frac\ (u\ y)$  **by** *auto*

**from**  $A(5)$  **have**  $\forall x \in X. 0 \leq u\ x$  **by** *auto*

**from**  $region.intros[OF\ this\ B\ -\ *]$  **show** *?case* **by** *auto*

**next**

**case**  $A: (2\ u)$

**have**  $intv\text{-}elem\ x\ u\ (I\ x)$  **if**  $x \in X$  **for**  $x$  **using**  $*$   $A(5)$  **that** **by** *auto*

**then have**  $B: \forall x \in X. intv\text{-}elem\ x\ u\ (I\ x)$  **by** *auto*

**let**  $?X_0 = \{x \in X. \exists d. I\ x = Intv\ d\}$

```

{ fix x y assume x: x ∈ ?X0 and y: y ∈ ?X0
  have (x, y) ∈ r ↔ frac (u x) ≤ frac (u y)
  proof
    assume frac (u x) ≤ frac (u y)
    with A(5) x y * have (x,y) ∈ r' by auto
    with A(4) x y * have frac (v x) ≤ frac (v y) by auto
    with A(3) x y show (x,y) ∈ r by auto
  next
    assume (x,y) ∈ r
    with A(3) x y have frac (v x) ≤ frac (v y) by auto
    with A(4) x y * have (x,y) ∈ r' by auto
    with A(5) x y * show frac (u x) ≤ frac (u y) by auto
  qed
}
then have *: ∀ x ∈ ?X0. ∀ y ∈ ?X0. (x, y) ∈ r ↔ frac (u x) ≤ frac
(u y) by auto
from A(5) have ∀ x ∈ X. 0 ≤ u x by auto
from region.intros[OF this B - *] show ?case by auto
qed
qed

```

**lemma**  *$\mathcal{R}$ -regions-distinct:*

$\llbracket \mathcal{R} = \{ \text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r \}; R \in \mathcal{R}; v \in R; R' \in \mathcal{R}; R \neq R' \rrbracket \implies v \notin R'$

**using** *valid-regions-distinct* **by** *blast*

Secondly, we also need to show that every valuations belongs to a region which is part of the partition.

**definition** *intv-of* ::  $\text{nat} \Rightarrow t \Rightarrow \text{intv}$  **where**

```

intv-of k c ≡
  if (c > k) then Greater k
  else if (∃ x :: nat. x = c) then (Const (nat (floor c)))
  else (Intv (nat (floor c)))

```

**lemma** *region-cover:*

$\forall x \in X. u \ x \geq 0 \implies \exists R. R \in \{ \text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r \} \wedge u \in R$

**proof** (*standard, standard*)

```

assume assm: ∀ x ∈ X. 0 ≤ u x
let ?I = λ x. intv-of (k x) (u x)
let ?X0 = {x ∈ X. ∃ d. ?I x = Intv d}
let ?r = {(x,y). x ∈ ?X0 ∧ y ∈ ?X0 ∧ frac (u x) ≤ frac (u y)}
show u ∈ region X ?I ?r
proof (standard, auto simp: assm, goal-cases)

```

**case** (1  $x$ )  
**thus** ?case **unfolding** *intv-of-def*  
**proof** (*auto, goal-cases*)  
**case** A: (1  $a$ )  
**from** A(2) **have**  $\lfloor u x \rfloor = u x$  **by** (*metis of-int-floor-cancel of-int-of-nat-eq*)  
**with** *assm* A(1) **have**  $u x = \text{real}(\text{nat } \lfloor u x \rfloor)$  **by** *auto*  
**then show** ?case **by** *auto*  
**next**  
**case** A: 2  
**from** A(1,2) **have**  $\text{real}(\text{nat } \lfloor u x \rfloor) < u x$   
**by** (*metis assm floor-less-iff int-nat-eq less-eq-real-def less-irrefl not-less of-int-of-nat-eq of-nat-0*)  
**moreover from** *assm* **have**  $u x < \text{real}(\text{nat } (\lfloor u x \rfloor + 1))$  **by** *linarith*  
**ultimately show** ?case **by** *auto*  
**qed**  
**qed**  
**have** *valid-intv* ( $k x$ ) (*intv-of* ( $k x$ ) ( $u x$ )) **if**  $x \in X$  **for**  $x$  **using** *that*  
**proof** (*auto simp: intv-of-def, goal-cases*)  
**case** 1 **then show** ?case **by** (*intro valid-intv.intros(1)*) (*auto, linarith*)  
**next**  
**case** 2  
**then show** ?case **using** *assm floor-less-iff nat-less-iff*  
**by** (*intro valid-intv.intros(2)*) *fastforce+*  
**qed**  
**then have** *valid-region*  $X k ?I ?r$   
**by** (*intro valid-region.intros*) (*auto simp: refl-on-def trans-def total-on-def*)  
**then show** *region*  $X ?I ?r \in \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$  **by**  
*auto*  
**qed**

**lemma** *intv-not-empty*:  
**obtains**  $d$  **where** *intv-elem*  $x (v(x := d)) (I x)$   
**proof** (*cases I x, goal-cases*)  
**case** (1  $d$ )  
**then have** *intv-elem*  $x (v(x := d)) (I x)$  **by** *auto*  
**with** 1 **show** ?case **by** *auto*  
**next**  
**case** (2  $d$ )  
**then have** *intv-elem*  $x (v(x := d + 0.5)) (I x)$  **by** *auto*  
**with** 2 **show** ?case **by** *auto*  
**next**  
**case** (3  $d$ )  
**then have** *intv-elem*  $x (v(x := d + 0.5)) (I x)$  **by** *auto*  
**with** 3 **show** ?case **by** *auto*

qed

**fun** *get-intv-val* :: *intv*  $\Rightarrow$  *real*  $\Rightarrow$  *real*

**where**

*get-intv-val* (*Const* *d*) - = *d* |  
*get-intv-val* (*Intv* *d*) *f* = *d* + *f* |  
*get-intv-val* (*Greater* *d*) - = *d* + 1

**lemma** *region-not-empty-aux*:

**assumes**  $0 < f$   $f < 1$   $0 < g$   $g < 1$

**shows**  $\text{frac } (\text{get-intv-val } (\text{Intv } d) f) \leq \text{frac } (\text{get-intv-val } (\text{Intv } d') g) \longleftrightarrow f \leq g$

**using** *assms* **by** (*simp*, *metis frac-eq frac-nat-add-id less-eq-real-def*)

**lemma** *region-not-empty*:

**assumes** *finite* *X* *valid-region* *X* *k* *I* *r*

**shows**  $\exists u. u \in \text{region } X I r$

**proof** -

**let**  $?X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}$

**obtain** *f* :: '*a*  $\Rightarrow$  *nat* **where** *f*:

$\forall x \in ?X_0. \forall y \in ?X_0. f x \leq f y \longleftrightarrow (x, y) \in r$

**apply** (*rule finite-total-preorder-enumeration*)

**apply** (*subgoal-tac finite ?X<sub>0</sub>*)

**apply** *assumption*

**using** *assms* **by** *auto*

**let**  $?M = \text{if } ?X_0 \neq \{\} \text{ then } \text{Max } \{f x \mid x. x \in ?X_0\} \text{ else } 1$

**let**  $?f = \lambda x. (f x + 1) / (?M + 2)$

**let**  $?v = \lambda x. \text{get-intv-val } (I x) (\text{if } x \in ?X_0 \text{ then } ?f x \text{ else } 1)$

**have** *frac-intv*:  $\forall x \in ?X_0. 0 < ?f x \wedge ?f x < 1$

**proof** (*standard*, *goal-cases*)

**case** (*1* *x*)

**then** **have** \*:  $?X_0 \neq \{\}$  **by** *auto*

**have**  $f x \leq \text{Max } \{f x \mid x. x \in ?X_0\}$  **apply** (*rule Max-ge*) **using**  $\langle \text{finite } X \rangle$  **1** **by** *auto*

**with** *1* **show** *?case* **by** *auto*

qed

**with** *region-not-empty-aux* **have** \*:

$\forall x \in ?X_0. \forall y \in ?X_0. \text{frac } (?v x) \leq \text{frac } (?v y) \longleftrightarrow ?f x \leq ?f y$

**by** *force*

**have**  $\forall x \in ?X_0. \forall y \in ?X_0. ?f x \leq ?f y \longleftrightarrow f x \leq f y$  **by** (*simp add: divide-le-cancel*)**+**

**with** *f* **have**  $\forall x \in ?X_0. \forall y \in ?X_0. ?f x \leq ?f y \longleftrightarrow (x, y) \in r$  **by** *auto*

**with** \* **have** *frac-order*:  $\forall x \in ?X_0. \forall y \in ?X_0. \text{frac } (?v x) \leq \text{frac } (?v y) \longleftrightarrow (x, y) \in r$  **by** *auto*

```

have ?v ∈ region X I r
proof standard
  show ∀ x ∈ X. intv-elem x ?v (I x)
  proof (standard, case-tac I x, goal-cases)
    case (2 x d)
    then have *: x ∈ ?X0 by auto
    with frac-intv have 0 < ?f x ?f x < 1 by auto
    moreover from 2 have ?v x = d + ?f x by auto
    ultimately have ?v x < d + 1 ∧ d < ?v x by linarith
    then show intv-elem x ?v (I x) by (subst 2(2)) (intro intv-elem.intros(2),
auto)
  qed auto
next
  show ∀ x ∈ X. 0 ≤ get-intv-val (I x) (if x ∈ ?X0 then ?f x else 1)
  by (standard, case-tac I x) auto
next
  show {x ∈ X. ∃ d. I x = Intv d} = {x ∈ X. ∃ d. I x = Intv d} ..
next
  from frac-order show ∀ x ∈ ?X0. ∀ y ∈ ?X0. ((x, y) ∈ r) = (frac (?v x) ≤
frac (?v y)) by blast
  qed
  then show ?thesis by auto
qed

```

Now we can show that there is always exactly one region a valid valuation belongs to.

**lemma** *regions-partition*:

$\mathcal{R} = \{\text{region } X I r \mid I r. \text{ valid-region } X k I r\} \implies \forall x \in X. 0 \leq u x \implies \exists! R \in \mathcal{R}. u \in R$

**proof** (goal-cases)

**case** 1

**note** A = this

**with** region-cover[OF A(2)] **obtain** R **where** R: R ∈  $\mathcal{R}$  ∧ u ∈ R **by** fastforce

**moreover have** R' = R **if** R' ∈  $\mathcal{R}$  ∧ u ∈ R' **for** R'

**using** that R valid-regions-distinct **unfolding** A(1) **by** blast

**ultimately show** ?thesis **by** auto

**qed**

**lemma** *region-unique*:

$\mathcal{R} = \{\text{region } X I r \mid I r. \text{ valid-region } X k I r\} \implies u \in R \implies R \in \mathcal{R} \implies [u]_{\mathcal{R}} = R$

**proof** (goal-cases)

**case** 1

**note**  $A = \text{this}$   
**from**  $A$  **obtain**  $I r$  **where**  $*$ : *valid-region*  $X k I r R = \text{region } X I r u \in \text{region } X I r$  **by auto**  
**from**  $\text{this}(3)$  **have**  $\forall x \in X. 0 \leq u x$  **by auto**  
**from**  $\text{theI}'[\text{OF regions-partition}[\text{OF } A(1) \text{ this}]] A(1)$  **obtain**  $I' r'$  **where**  
 $v$ : *valid-region*  $X k I' r' [u]_{\mathcal{R}} = \text{region } X I' r' u \in \text{region } X I' r'$   
**unfolding part-def by auto**  
**from** *valid-regions-distinct* $[\text{OF } *(1) v(1) *(3) v(3)] v(2) *(2)$  **show**  $?case$   
**by auto**  
**qed**

**lemma** *regions-partition'*:

$\mathcal{R} = \{\text{region } X I r \mid I r. \text{valid-region } X k I r\} \implies \forall x \in X. 0 \leq v x \implies \forall x \in X. 0 \leq v' x \implies v' \in [v]_{\mathcal{R}}$   
 $\implies [v']_{\mathcal{R}} = [v]_{\mathcal{R}}$

**proof** (*goal-cases*)

**case** 1

**note**  $A = \text{this}$

**from**  $\text{theI}'[\text{OF regions-partition}[\text{OF } A(1,2)]] A(1,4)$  **obtain**  $I r$  **where**

$v$ : *valid-region*  $X k I r [v]_{\mathcal{R}} = \text{region } X I r v' \in \text{region } X I r$

**unfolding part-def by auto**

**from**  $\text{theI}'[\text{OF regions-partition}[\text{OF } A(1,3)]] A(1)$  **obtain**  $I' r'$  **where**

$v'$ : *valid-region*  $X k I' r' [v']_{\mathcal{R}} = \text{region } X I' r' v' \in \text{region } X I' r'$

**unfolding part-def by auto**

**from** *valid-regions-distinct* $[\text{OF } v'(1) v(1) v'(3) v(3)] v(2) v'(2)$  **show**

$?case$  **by simp**

**qed**

**lemma** *regions-closed*:

$\mathcal{R} = \{\text{region } X I r \mid I r. \text{valid-region } X k I r\} \implies R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies [v \oplus t]_{\mathcal{R}} \in \mathcal{R}$

**proof** *goal-cases*

**case**  $A$ : 1

**then obtain**  $I r$  **where**  $v \in \text{region } X I r$  **by auto**

**from**  $\text{this}(1)$  **have**  $\forall x \in X. v x \geq 0$  **by auto**

**with**  $A(4)$  **have**  $\forall x \in X. (v \oplus t) x \geq 0$  **unfolding cval-add-def by simp**

**from** *regions-partition* $[\text{OF } A(1) \text{ this}]$  **obtain**  $R'$  **where**  $R' \in \mathcal{R} (v \oplus t) \in R'$  **by auto**

**with** *region-unique* $[\text{OF } A(1) \text{ this}(2,1)]$  **show**  $?case$  **by auto**

**qed**

**lemma** *regions-closed'*:

$\mathcal{R} = \{\text{region } X I r \mid I r. \text{valid-region } X k I r\} \implies R \in \mathcal{R} \implies v \in R \implies$

$t \geq 0 \implies (v \oplus t) \in [v \oplus t]_{\mathcal{R}}$

**proof** *goal-cases*

**case** *A: 1*

**then obtain**  $I r$  **where**  $v \in \text{region } X I r$  **by** *auto*

**from**  $\text{this}(1)$  **have**  $\forall x \in X. v x \geq 0$  **by** *auto*

**with**  $A(4)$  **have**  $\forall x \in X. (v \oplus t) x \geq 0$  **unfolding** *cval-add-def* **by** *simp*

**from** *regions-partition*[*OF*  $A(1)$   $\text{this}$ ] **obtain**  $R'$  **where**  $R' \in \mathcal{R} (v \oplus t) \in R'$  **by** *auto*

**with** *region-unique*[*OF*  $A(1)$   $\text{this}(2,1)$ ] **show** *?case* **by** *auto*

**qed**

**lemma** *valid-regions-I-cong*:

$\text{valid-region } X k I r \implies \forall x \in X. I x = I' x \implies \text{region } X I r = \text{region } X I' r \wedge \text{valid-region } X k I' r$

**proof** (*safe, goal-cases*)

**case** (*1 v*)

**note**  $A = \text{this}$

**then have** [*simp*]: $\bigwedge x. x \in X \implies I' x = I x$  **by** *metis*

**show** *?case*

**proof** (*standard, goal-cases*)

**case** *1*

**from**  $A(3)$  **show** *?case* **by** *auto*

**next**

**case** *2*

**from**  $A(3)$  **show** *?case* **by** *auto*

**next**

**case** *3*

**show**  $\{x \in X. \exists d. I x = \text{Intv } d\} = \{x \in X. \exists d. I' x = \text{Intv } d\}$  **by** *auto*

**next**

**case** *4*

**let**  $?X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}$

**from**  $A(3)$  **show**  $\forall x \in ?X_0. \forall y \in ?X_0. ((x, y) \in r) = (\text{frac } (v x) \leq \text{frac } (v y))$  **by** *auto*

**qed**

**next**

**case** (*2 v*)

**note**  $A = \text{this}$

**then have** [*simp*]: $\bigwedge x. x \in X \implies I' x = I x$  **by** *metis*

**show** *?case*

**proof** (*standard, goal-cases*)

**case** *1*

**from**  $A(3)$  **show** *?case* **by** *auto*

**next**

```

    case 2
    from A(3) show ?case by auto
next
    case 3
    show {x ∈ X. ∃ d. I' x = Intv d} = {x ∈ X. ∃ d. I x = Intv d} by auto
next
    case 4
    let ?X0 = {x ∈ X. ∃ d. I' x = Intv d}
    from A(3) show ∀ x ∈ ?X0. ∀ y ∈ ?X0. ((x, y) ∈ r) = (frac (v x) ≤
frac (v y)) by auto
    qed
next
    case 3
    note A = this
    then have [simp]: ∧ x. x ∈ X ⇒ I' x = I x by metis
    show ?case
    apply rule
    apply (subgoal-tac {x ∈ X. ∃ d. I x = Intv d} = {x ∈ X. ∃ d. I' x =
Intv d})
    apply assumption
    using A by auto
    qed

fun intv-const :: intv ⇒ nat
where
  intv-const (Const d) = d |
  intv-const (Intv d) = d |
  intv-const (Greater d) = d

lemma finite-ℛ:
  notes [[simp proc add: finite-Collect]] finite-subset[intro]
  fixes X k
  defines ℛ ≡ {region X I r | I r. valid-region X k I r}
  assumes finite X
  shows finite ℛ
proof -
  { fix I r assume A: valid-region X k I r
    let ?X0 = {x ∈ X. ∃ d. I x = Intv d}
    from A have r ⊆ X × X by auto
    then have r ∈ Pow (X × X) by auto
  }
  then have {r. ∃ I. valid-region X k I r} ⊆ Pow (X × X) by auto
  with ⟨finite X⟩ have fin: finite {r. ∃ I. valid-region X k I r} by auto
  let ?m = Max {k x | x. x ∈ X}

```

```

let ?I = {intv. intv-const intv ≤ ?m}
let ?fin-map = λ I. ∀ x. (x ∈ X → I x ∈ ?I) ∧ (x ∉ X → I x = Const
0)
let ?R = {region X I r | I r. valid-region X k I r ∧ ?fin-map I}
have ?I = (Const ‘ {d. d ≤ ?m}) ∪ (Intv ‘ {d. d ≤ ?m}) ∪ (Greater ‘
{d. d ≤ ?m})
by auto (case-tac x, auto)
then have finite ?I by auto
from finite-set-of-finite-funs[OF ‹finite X› this] have finite {I. ?fin-map
I} .
with fin have finite {(I, r). valid-region X k I r ∧ ?fin-map I}
by (fastforce intro: pairwise-finiteI finite-ex-and1 frac-add-le-preservation
del: finite-subset)
then have finite ?R by fastforce
moreover have R ⊆ ?R
proof
fix R assume R: R ∈ R
then obtain I r where I: R = region X I r valid-region X k I r
unfolding R-def by auto
let ?I = λ x. if x ∈ X then I x else Const 0
let ?R = region X ?I r
from valid-regions-I-cong[OF I(2)] I have R = ?R valid-region X k ?I
r by auto
moreover have ∀ x. x ∉ X → ?I x = Const 0 by auto
moreover have ∀ x. x ∈ X → intv-const (I x) ≤ ?m
proof auto
fix x assume x: x ∈ X
with I(2) have valid-intv (k x) (I x) by auto
moreover from ‹finite X› x have k x ≤ ?m by (auto intro: Max-ge)
ultimately show intv-const (I x) ≤ Max {k x | x. x ∈ X} by (cases
I x) auto
qed
ultimately show R ∈ ?R by force
qed
ultimately show finite R by blast
qed

```

**lemma** SuccI2:

```

R = {region X I r | I r. valid-region X k I r} ⇒ v ∈ R ⇒ R ∈ R ⇒
t ≥ 0 ⇒ R' = [v ⊕ t]R
⇒ R' ∈ Succ R R

```

**proof** goal-cases

**case** A: 1

```

from Succ.intros[OF A(2) A(3) regions-closed[OF A(1,3,2,4)] A(4)]

```

$A(5)$  **show** ?case **by** auto  
**qed**

### 5.3 Set of Regions

The first property Bouyer shows is that these regions form a 'set of regions'.

For the unbounded region in the upper right corner, the set of successors only contains itself.

**lemma** *Succ-refl*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies \text{finite } X \implies R \in \mathcal{R} \implies R \in \text{Succ } \mathcal{R} \ R$

**proof** *goal-cases*

**case**  $A: 1$

**then obtain**  $I \ r$  **where**  $R: \text{ valid-region } X \ k \ I \ r \ R = \text{region } X \ I \ r$  **by** auto  
**with**  $A$  **region-not-empty** **obtain**  $v$  **where**  $v: v \in \text{region } X \ I \ r$  **by**metis  
**with**  $R$  **have**  $*$ :  $(v \oplus 0) \in R$  **unfolding** *cval-add-def* **by** auto

**from** *regions-closed'*[*OF*  $A(1,3-)$ ]  $v \ R$  **have**  $(v \oplus 0) \in [v \oplus 0]_{\mathcal{R}}$  **by** auto  
**from** *region-unique*[*OF*  $A(1) * A(3)$ ]  $A(3)$   $v$ [*unfolded*  $R(2)$ ][*symmetric*]

**show** ?case **by** auto

**qed**

**lemma** *Succ-refl'*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies \text{finite } X \implies \forall x \in X.$

$\exists c. I \ x = \text{Greater } c$

$\implies \text{region } X \ I \ r \in \mathcal{R} \implies \text{Succ } \mathcal{R} (\text{region } X \ I \ r) = \{\text{region } X \ I \ r\}$

**proof** *goal-cases*

**case**  $A: 1$

**have**  $*$ :  $(v \oplus t) \in \text{region } X \ I \ r$  **if**  $v: v \in \text{region } X \ I \ r$  **and**  $t: t \geq 0$  **for**  $v$   
**and**  $t :: t$

**proof** ((*rule region.intros*), auto, *goal-cases*)

**case** 1

**with**  $v \ t$  **show** ?case **unfolding** *cval-add-def* **by** auto

**next**

**case** (2  $x$ )

**with**  $A$  **obtain**  $c$  **where**  $c: I \ x = \text{Greater } c$  **by** auto

**with**  $v \ 2$  **have**  $v \ x > c$  **by** *fastforce*

**with**  $t$  **have**  $v \ x + t > c$  **by** auto

**then have**  $(v \oplus t) \ x > c$  **by** (*simp add: cval-add-def*)

**from** *intv-elem.intros*(3)[*of*  $c \ v \oplus t$ , *OF* *this*]  $c$  **show** ?case **by** auto

**next**

**case** (3  $x$ )

**from** *this*(1)  $A$  **obtain**  $c$  **where**  $I \ x = \text{Greater } c$  **by** auto

**with** 3(2) **show** ?case **by** auto

```

next
  case ( $\not\leq x$ )
  from  $this(1)$   $A$  obtain  $c$  where  $I x = Greater\ c$  by auto
  with  $\not\leq(2)$  show ?case by auto
qed
show ?case
proof (standard, standard)
  fix  $R$  assume  $R: R \in Succ\ \mathcal{R}$  (region X I r)
  then obtain  $v\ t$  where  $v$ :
     $v \in region\ X\ I\ r\ R = [v \oplus t]_{\mathcal{R}}\ R \in \mathcal{R}\ t \geq 0$ 
  by (cases rule: Succ.cases) auto
  from  $v(1)$  have  $**$ :  $\forall x \in X. 0 \leq v\ x$  by auto
  with  $v(4)$  have  $\forall x \in X. 0 \leq (v \oplus t)\ x$  unfolding cval-add-def by
auto
  from  $*[OF\ v(1,4)]\ regions-partition'[OF\ A(1)\ **\ this]\ region-unique[OF\ A(1)\ v(1)\ A(4)]\ v(2)$ 
  show  $R \in \{region\ X\ I\ r\}$  by auto
next
  from  $A(4)$  obtain  $I'\ r'$  where  $R': region\ X\ I\ r = region\ X\ I'\ r'$ 
valid-region X k I' r'
  unfolding  $A(1)$  by auto
  with region-not-empty[OF A(2) this(2)] obtain  $v$  where  $v: v \in region\ X\ I\ r$  by auto
  from region-unique[OF A(1) this A(4)] have  $*$ :  $[v \oplus 0]_{\mathcal{R}} = region\ X\ I\ r$ 
r
  unfolding cval-add-def by auto
  with  $v\ A(4)$  have  $[v \oplus 0]_{\mathcal{R}} \in Succ\ \mathcal{R}$  (region X I r) by (intro Succ.intros;
auto)
  with  $*$  show  $\{region\ X\ I\ r\} \subseteq Succ\ \mathcal{R}$  (region X I r) by auto
qed
qed

```

Defining the closest successor of a region. Only exists if at least one interval is upper-bounded.

**definition**

```

succ  $\mathcal{R}\ R =$ 
  (SOME  $R'. R' \in Succ\ \mathcal{R}\ R \wedge (\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t'))$ )

```

**inductive** *isConst* :: *intv*  $\Rightarrow$  *bool*

**where**

```

isConst (Const -)

```

**inductive** *isIntv* :: *intv*  $\Rightarrow$  *bool*

**where**

*isIntv* (*Intv* -)

**inductive** *isGreater* :: *intv*  $\Rightarrow$  *bool*

**where**

*isGreater* (*Greater* -)

**declare** *isIntv.intros*[*intro!*] *isConst.intros*[*intro!*] *isGreater.intros*[*intro!*]

**declare** *isIntv.cases*[*elim!*] *isConst.cases*[*elim!*] *isGreater.cases*[*elim!*]

What Bouyer states at the end. However, we have to be a bit more precise than in her statement.

**lemma** *closest-prestable-1*:

**fixes** *I X k r*

**defines**  $\mathcal{R} \equiv \{ \text{region } X \ I \ r \mid I \ r. \ \text{valid-region } X \ k \ I \ r \}$

**defines**  $R \equiv \text{region } X \ I \ r$

**defines**  $Z \equiv \{ x \in X . \exists c. I \ x = \text{Const } c \}$

**assumes**  $Z \neq \{ \}$

**defines**  $I' \equiv \lambda x. \text{if } x \notin Z \text{ then } I \ x \text{ else if } \text{intv-const } (I \ x) = k \ x \text{ then } \text{Greater } (k \ x) \text{ else } \text{Intv } (\text{intv-const } (I \ x))$

**defines**  $r' \equiv r \cup \{ (x,y) . x \in Z \wedge y \in X \wedge \text{intv-const } (I \ x) < k \ x \wedge \text{isIntv } (I' \ y) \}$

**assumes** *finite* *X*

**assumes** *valid-region* *X k I r*

**shows**  $\forall v \in R. \forall t > 0. \exists t' \leq t. (v \oplus t') \in \text{region } X \ I' \ r' \wedge t' \geq 0$

**and**  $\forall v \in \text{region } X \ I' \ r'. \forall t \geq 0. (v \oplus t) \notin R$

**and**  $\forall x \in X. \neg \text{isConst } (I' \ x)$

**and**  $\forall v \in R. \forall t < 1. \forall t' \geq 0. (v \oplus t') \in \text{region } X \ I' \ r'$

$\longrightarrow \{ x. x \in X \wedge (\exists c. I \ x = \text{Intv } c \wedge v \ x + t \geq c + 1) \}$

$= \{ x. x \in X \wedge (\exists c. I' \ x = \text{Intv } c \wedge (v \oplus t') \ x + (t - t') \geq$

$c + 1) \}$

**proof** (*safe, goal-cases*)

**fix** *v* **assume** *v*:  $v \in R$  **fix** *t* :: *t* **assume** *t*:  $0 < t$

**have** *elem*: *intv-elem* *x v* (*I* *x*) **if** *x*:  $x \in X$  **for** *x* **using** *v x* **unfolding** *R-def* **by** *auto*

**have** \*:  $(v \oplus t) \in \text{region } X \ I' \ r'$  **if** *A*:  $\forall x \in X. \neg \text{isIntv } (I \ x)$  **and** *t*:  $t > 0 \ t < 1$  **for** *t*

**proof** (*standard, goal-cases*)

**case** *1*

**from** *v* **have**  $\forall x \in X. v \ x \geq 0$  **unfolding** *R-def* **by** *auto*

**with** *t* **show** ?*case* **unfolding** *cval-add-def* **by** *auto*

**next**

**case** *2*

```

show ?case
proof (standard, case-tac I x, goal-cases)
  case (1 x c)
  with elem[OF ⟨x ∈ X⟩] have v x = c by auto
  show ?case
proof (cases intv-const (I x) = k x, auto simp: 1 I'-def Z-def, goal-cases)
  case 1
  with ⟨v x = c⟩ have v x = k x by auto
  with t show ?case by (auto simp: cval-add-def)
next
  case 2
  from assms(8) 1 have c ≤ k x by (cases rule: valid-region.cases)
auto
  with 2 have c < k x by linarith
  from t ⟨v x = c⟩ show ?case by (auto simp: cval-add-def)
qed
next
  case (2 x c)
  with A show ?case by auto
next
  case (3 x c)
  then have I' x = Greater c unfolding I'-def Z-def by auto
  with t 3 elem[OF ⟨x ∈ X⟩] show ?case by (auto simp: cval-add-def)
qed
next
  case 3 show {x ∈ X. ∃ d. I' x = Intv d} = {x ∈ X. ∃ d. I' x = Intv
d} ..
next
  case 4
  let ?X0' = {x ∈ X. ∃ d. I' x = Intv d}
  show ∀ x ∈ ?X0'. ∀ y ∈ ?X0'. ((x, y) ∈ r') = (frac ((v ⊕ t) x) ≤ frac ((v
⊕ t) y))
proof (safe, goal-cases)
  case (1 x y d d')
  note B = this
  have x ∈ Z apply (rule ccontr) using A B by (auto simp: I'-def)
  with elem[OF B(1)] have frac (v x) = 0 unfolding Z-def by auto
  with frac-distr[of t v x] t have *: frac (v x + t) = t by auto
  have y ∈ Z apply (rule ccontr) using A B by (auto simp: I'-def)
  with elem[OF B(3)] have frac (v y) = 0 unfolding Z-def by auto
  with frac-distr[of t v y] t have frac (v y + t) = t by auto
  with * show ?case unfolding cval-add-def by auto
next
  case B: (2 x)

```

```

    have  $x \in Z$  apply (rule ccontr) using  $A B$  by (auto simp:  $I'$ -def)
    with  $B$  have  $\text{intv-const } (I x) \neq k x$  unfolding  $I'$ -def by auto
    with  $B(1)$  assms(8) have  $\text{intv-const } (I x) < k x$  by (fastforce elim!:
valid-intv.cases)
    with  $B \langle x \in Z \rangle$  show ?case unfolding  $r'$ -def by auto
qed
qed
let ?S = {1 - frac (v x) | x. x ∈ X ∧ isIntv (I x)}
let ?t = Min ?S
{ assume A: ∃ x ∈ X. isIntv (I x)
  from ⟨finite X⟩ have finite ?S by auto
  from A have ?S ≠ {} by auto
  from Min-in[OF ⟨finite ?S⟩ this] obtain x where
    x: x ∈ X isIntv (I x) ?t = 1 - frac (v x)
  by force
  have frac (v x) < 1 by (simp add: frac-lt-1)
  then have ?t > 0 by (simp add: x(3))
  then have ?t / 2 > 0 by auto
  from x(2) obtain c where I x = Intv c by (auto)
  with elem[OF x(1)] have v-x: c < v x v x < c + 1 by auto
  from nat-intv-frac-gt0[OF this] have frac (v x) > 0 .
  with x(3) have ?t < 1 by auto
  { fix t :: t assume t: 0 < t t ≤ ?t / 2
    { fix y assume y ∈ X isIntv (I y)
      then have 1 - frac (v y) ∈ ?S by auto
      from Min-le[OF ⟨finite ?S⟩ this] ⟨?t > 0⟩ t have t < 1 - frac (v
y) by linarith
    } note frac-bound = this
  have (v ⊕ t) ∈ region X I' r'
  proof (standard, goal-cases)
    case 1
    from v have ∀ x ∈ X. v x ≥ 0 unfolding R-def by auto
    with ⟨?t > 0⟩ t show ?case unfolding eval-add-def by auto
  next
    case 2
    show ?case
    proof (standard, case-tac I x, goal-cases)
      case A: (1 x c)
      with elem[OF ⟨x ∈ X⟩] have v x = c by auto
      show ?case
      proof (cases intv-const (I x) = k x, auto simp: A I'-def Z-def,
goal-cases)
        case 1
        with ⟨v x = c⟩ have v x = k x by auto

```

```

    with  $\langle ?t > 0 \rangle t$  show ?case by (auto simp: cval-add-def)
  next
    case 2
    from assms(8) A have  $c \leq k x$  by (cases rule: valid-region.cases)
  auto
    with 2 have  $c < k x$  by linarith
    from  $\langle v x = c \rangle \langle ?t < 1 \rangle t$  show ?case
    by (auto simp: cval-add-def)
  qed
next
  case (2 x c)
  with elem[OF  $\langle x \in X \rangle$ ] have  $v: c < v x \ v x < c + 1$  by auto
  with  $\langle ?t > 0 \rangle$  have  $c < v x + (?t / 2)$  by auto
  from 2 have  $I' x = I x$  unfolding I'-def Z-def by auto
  from frac-bound[OF 2(1)] 2(2) have  $t < 1 - \text{frac } (v x)$  by auto
  from frac-add-le-preservation[OF v(2) this] t v(1) 2 show ?case
  unfolding cval-add-def  $\langle I' x = I x \rangle$  by auto
next
  case (3 x c)
  then have  $I' x = \text{Greater } c$  unfolding I'-def Z-def by auto
  with 3 elem[OF  $\langle x \in X \rangle$ ] t show ?case
  by (auto simp: cval-add-def)
  qed
next
  case 3 show  $\{x \in X. \exists d. I' x = \text{Intv } d\} = \{x \in X. \exists d. I' x = \text{Intv } d\}$  ..
next
  case 4
  let ?X0 =  $\{x \in X. \exists d. I x = \text{Intv } d\}$ 
  let ?X'0 =  $\{x \in X. \exists d. I' x = \text{Intv } d\}$ 
  show  $\forall x \in ?X'_0. \forall y \in ?X_0. ((x, y) \in r') = (\text{frac } ((v \oplus t) x) \leq \text{frac } ((v \oplus t) y))$ 
  proof (safe, goal-cases)
    case (1 x y d d')
    note B = this
    show ?case
    proof (cases  $x \in Z$ )
      case False
      note F = this
      show ?thesis
    proof (cases  $y \in Z$ )
      case False
      with F B have *:  $x \in ?X_0 \ y \in ?X_0$  unfolding I'-def by auto
      from B(5) show ?thesis unfolding r'-def

```

```

proof (safe, goal-cases)
  case 1
    with  $v * \text{have } le: \text{frac } (v x) \leq \text{frac } (v y)$  unfolding  $R\text{-def}$ 
by auto
    from  $\text{frac-bound } * \text{have } t < 1 - \text{frac } (v x) \ t < 1 - \text{frac } (v$ 
 $y)$  by fastforce+
    with  $\text{frac-distr } t$  have
 $\text{frac } (v x) + t = \text{frac } (v x + t) \ \text{frac } (v y) + t = \text{frac } (v y + t)$ 
by simp+
    with  $le$  show  $?case$  unfolding  $\text{cval-add-def}$  by fastforce
next
  case 2
    from  $\text{this}(1) \ \text{elem}$  have  $** : \text{frac } (v x) = 0$  unfolding  $Z\text{-def}$ 
by force
    from  $2(4)$  obtain  $c$  where  $I' y = \text{Intv } c$  by (auto )
    then have  $y \in Z \vee I y = \text{Intv } c$  unfolding  $I'\text{-def}$  by  $\text{presburger}$ 
    then show  $?case$ 
    proof
      assume  $y \in Z$ 
      with  $\text{elem}[OF \ 2(2)]$  have  $*** : \text{frac } (v y) = 0$  unfolding
 $Z\text{-def}$  by force
      show  $?thesis$  by ( $\text{simp add: } ** \ *** \ \text{frac-add} \ \text{cval-add-def}$ )
    next
      assume  $A : I y = \text{Intv } c$ 
      have  $le : \text{frac } (v x) \leq \text{frac } (v y)$  by ( $\text{simp add: } **$ )
      from  $\text{frac-bound } * \text{have } t < 1 - \text{frac } (v x) \ t < 1 - \text{frac } (v$ 
 $y)$  by fastforce+
      with  $2 \ t$  have
 $\text{frac } (v x) + t = \text{frac } (v x + t) \ \text{frac } (v y) + t = \text{frac } (v y$ 
 $+ t)$ 
      using  $F$  by blast+
      with  $le$  show  $?case$  unfolding  $\text{cval-add-def}$  by fastforce
    qed
  qed
next
  case  $True$ 
  then obtain  $d'$  where  $d' : I y = \text{Const } d'$  unfolding  $Z\text{-def}$  by
 $\text{auto}$ 
  from  $B(5)$  show  $?thesis$  unfolding  $r'\text{-def}$ 
proof (safe, goal-cases)
  case 1
    from  $d'$  have  $y \notin ?X_0$  by auto
    with  $\text{assms}(8)$  show  $?case$  using  $1$  by auto
  next

```

```

      case 2
      with F show ?case by simp
    qed
  qed
next
  case True
  with elem have **: frac (v x) = 0 unfolding Z-def by force
  from B(4) have y ∈ Z ∨ I y = Intv d' unfolding I'-def by
presburger
  then show ?thesis
  proof
    assume y ∈ Z
    with elem[OF B(3)] have ***: frac (v y) = 0 unfolding Z-def
by force
    show ?thesis by (simp add: ** *** frac-add cval-add-def)
  next
    assume A: I y = Intv d'
    with B(3) have y ∈ ?X0 by auto
    with frac-bound have t < 1 - frac (v y) by fastforce+
    moreover from ** ⟨?t < 1⟩ have ?t / 2 < 1 - frac (v x) by
linarith
    ultimately have
      frac (v x) + t = frac (v x + t) frac (v y) + t = frac (v y + t)
    using frac-distr t by simp+
    moreover have frac (v x) <= frac (v y) by (simp add: **)
    ultimately show ?thesis unfolding cval-add-def by fastforce
  qed
  qed
next
  case B: (2 x y d d')
  show ?case
  proof (cases x ∈ Z, goal-cases)
    case True
    with B(1,2) have intv-const (I x) ≠ k x unfolding I'-def by
auto
    with B(1) assms(8) have intv-const (I x) < k x by (fastforce
elim!: valid-intv.cases)
    with B True show ?thesis unfolding r'-def by auto
  next
    case (False)
    with B(1,2) have x-intv: isIntv (I x) unfolding Z-def I'-def by
auto
    show ?thesis
    proof (cases y ∈ Z)

```

```

      case False
      with B(3,4) have y-intv: isIntv (I y) unfolding Z-def I'-def
by auto
      with frac-bound x-intv B(1,3) have t < 1 - frac (v x) t < 1
- frac (v y) by auto
      from frac-add-leD[OF - this] B(5) t have
      frac (v x) ≤ frac (v y)
      by (auto simp: cval-add-def)
      with v assms(2) B(1,3) x-intv y-intv have (x, y) ∈ r by (auto
)

      then show ?thesis by (simp add: r'-def)
next
      case True
      from frac-bound x-intv B(1) have b: t < 1 - frac (v x) by auto
      from x-intv obtain c where I x = Intv c by auto
      with elem[OF ⟨x ∈ X⟩] have v: c < v x v x < c + 1 by auto
      from True elem[OF ⟨y ∈ X⟩] have *: frac (v y) = 0 unfolding
Z-def by auto
      with t ⟨?t < 1⟩ floor-frac-add-preservation'[of t v y] have
      floor (v y + t) = floor (v y)
      by auto
      then have frac (v y + t) = t
      by (metis * add-diff-cancel-left' diff-add-cancel diff-self frac-def)
      moreover from nat-intv-frac-gt0[OF v] have 0 < frac (v x) .
      moreover from frac-distr[OF - b] t have frac (v x + t) = frac
(v x) + t by auto
      ultimately show ?thesis using B(5) unfolding cval-add-def
by auto
      qed
      qed
      qed
      qed
    }
    with ⟨?t/2 > 0⟩ have 0 < ?t/2 ∧ (∀ t. 0 < t ∧ t ≤ ?t/2 → (v ⊕
t) ∈ region X I' r') by auto
  } note ** = this
  show ∃ t' ≤ t. (v ⊕ t') ∈ region X I' r' ∧ 0 ≤ t'
  proof (cases ∃ x ∈ X. isIntv (I x))
  case True
  note T = this
  show ?thesis
  proof (cases t ≤ ?t/2)
  case True with T t ** show ?thesis by auto
  next

```

```

    case False
    then have  $?t/2 \leq t$  by auto
    moreover from  $T$  ** have  $(v \oplus ?t/2) \in \text{region } X I' r' ?t/2 > 0$  by
auto
    ultimately show ?thesis by (fastforce del: region.cases)
  qed
next
  case False
  note  $F = \text{this}$ 
  show ?thesis
  proof (cases  $t < 1$ )
    case True with  $F t$  * show ?thesis by auto
  next
    case False
    then have  $0.5 \leq t$  by auto
    moreover from  $F$  * have  $(v \oplus 0.5) \in \text{region } X I' r'$  by auto
    ultimately show ?thesis by (fastforce del: region.cases)
  qed
qed
next
  fix  $v t$  assume  $A: v \in \text{region } X I' r' 0 \leq t (v \oplus t) \in R$ 
  from assms(3,4) obtain  $x c$  where  $x: I x = \text{Const } c x \in Z x \in X$  by
auto
  with  $A(1)$  have intv-elem  $x v (I' x)$  by auto
  with  $x$  have  $v x > c$  unfolding I'-def
  apply (auto elim: intv-elem.cases)
  apply (cases  $c = k x$ )
  by auto
  moreover from  $A(3)$   $x(1,3)$  have  $v x + t = c$ 
  by (fastforce elim!: intv-elem.cases simp: cval-add-def R-def)
  ultimately show False using  $A(2)$  by auto
next
  fix  $x c$  assume  $x \in X I' x = \text{Const } c$ 
  then show False
  apply (auto simp: I'-def Z-def)
  apply (cases  $\forall c. I x \neq \text{Const } c$ )
  apply auto
  apply (rename-tac c')
  apply (case-tac c' = k x)
  by auto
next
  case ( $\lambda v t t' x c$ )
  note  $A = \text{this}$ 
  then have  $I' x = \text{Intv } c$  unfolding I'-def Z-def by auto

```

**moreover from A have**  $real (c + 1) \leq (v \oplus t') x + (t - t')$  **unfolding**  
*cval-add-def* **by** *auto*  
**ultimately show** *?case* **by** *blast*  
**next**  
**case** *A*:  $(5 v t t' x c)$   
**show** *?case*  
**proof**  $(cases x \in Z)$   
**case** *False*  
**with** *A* **have**  $I x = Intv c$  **unfolding** *I'-def* **by** *auto*  
**with** *A* **show** *?thesis* **unfolding** *cval-add-def* **by** *auto*  
**next**  
**case** *True*  
**with** *A(6)* **have**  $I x = Const c$   
**apply**  $(auto simp: I'-def)$   
**apply**  $(cases intv-const (I x) = k x)$   
**by**  $(auto simp: Z-def)$   
**with** *A(1,5)* *R-def* **have**  $v x = c$  **by** *fastforce*  
**with** *A(2,7)* **show** *?thesis* **by**  $(auto simp: cval-add-def)$   
**qed**  
**qed**

**lemma** *closest-valid-1*:

**fixes**  $I X k r$   
**defines**  $\mathcal{R} \equiv \{region X I r \mid I r. valid-region X k I r\}$   
**defines**  $R \equiv region X I r$   
**defines**  $Z \equiv \{x \in X . \exists c. I x = Const c\}$   
**assumes**  $Z \neq \{\}$   
**defines**  $I' \equiv \lambda x. if x \notin Z then I x else if intv-const (I x) = k x then$   
*Greater (k x) else Intv (intv-const (I x))*  
**defines**  $r' \equiv r \cup \{(x,y) . x \in Z \wedge y \in X \wedge intv-const (I x) < k x \wedge isIntv$   
 $(I' y)\}$   
**assumes** *finite X*  
**assumes** *valid-region X k I r*  
**shows** *valid-region X k I' r'*  
**proof**  
**let**  $?X_0 = \{x \in X. \exists d. I x = Intv d\}$   
**let**  $?X_0' = \{x \in X. \exists d. I' x = Intv d\}$   
**let**  $?S = \{(x, y). x \in Z \wedge y \in X \wedge intv-const (I x) < k x \wedge isIntv (I' y)\}$   
**show**  $?X_0' = ?X_0' ..$   
**have** *r-subset*:  $r \subseteq ?X_0 \times ?X_0$  **and** *refl*: *refl-on ?X\_0 r* **and** *total*: *total-on*  
 $?X_0 r$  **and**  
*trans*: *trans r* **and** *valid*:  $\bigwedge x. x \in X \implies valid-intv (k x) (I x)$   
**using** *assms(8)* **by** *auto*  
**then have**  $r \subseteq ?X_0' \times ?X_0'$  **unfolding** *I'-def Z-def* **by** *auto*

```

moreover have  $?S \subseteq ?X_0' \times ?X_0'$ 
  apply (auto)
  apply (auto simp: Z-def)[]
  apply (auto simp: I'-def)[]
done
ultimately show  $r' \subseteq ?X_0' \times ?X_0'$  unfolding r'-def by auto

show refl-on  $?X_0'$  r' unfolding refl-on-def
proof auto
  fix x d assume A:  $x \in X$   $I' x = \text{Intv } d$ 
  show  $(x, x) \in r'$ 
  proof (cases  $x \in Z$ )
    case True
      with A have intv-const  $(I x) \neq k x$  unfolding I'-def by auto
      with assms(8) A(1) have intv-const  $(I x) < k x$  by (fastforce elim!:
valid-intv.cases)
      with True A show  $(x, x) \in r'$  by (auto simp: r'-def)
    next
      case False
      with A refl show  $(x, x) \in r'$  by (auto simp: I'-def refl-on-def r'-def)
  qed
qed
show total-on  $?X_0'$  r' unfolding total-on-def
proof (standard, standard, standard)
  fix x y assume  $x \in ?X_0'$   $y \in ?X_0'$   $x \neq y$ 
  then obtain d d' where A:  $x \in X$   $y \in X$   $I' x = (\text{Intv } d)$   $I' y = (\text{Intv } d')$   $x$ 
 $\neq y$  by auto
  let ?thesis =  $(x, y) \in r' \vee (y, x) \in r'$ 
  show ?thesis
  proof (cases  $x \in Z$ )
    case True
      with A have intv-const  $(I x) \neq k x$  unfolding I'-def by auto
      with assms(8) A(1) have intv-const  $(I x) < k x$  by (fastforce elim!:
valid-intv.cases)
      with True A show ?thesis by (auto simp: r'-def)
    next
      case F: False
      show ?thesis
      proof (cases  $y \in Z$ )
        case True
          with A have intv-const  $(I y) \neq k y$  unfolding I'-def by auto
          with assms(8) A(2) have intv-const  $(I y) < k y$  by (fastforce elim!:
valid-intv.cases)
          with True A show ?thesis by (auto simp: r'-def)
        case False
          show ?thesis
      qed
  qed

```

```

next
  case False
  with A F have  $I x = \text{Intv } d \ I y = \text{Intv } d'$  by (auto simp: I'-def)
  with  $A(1,2,5)$  total show ?thesis unfolding total-on-def r'-def by
auto
  qed
  qed
  qed
  show trans r' unfolding trans-def
  proof safe
    fix  $x y z$  assume  $A: (x, y) \in r' (y, z) \in r'$ 
    show  $(x, z) \in r'$ 
    proof (cases (x,y) \in r)
      case True
      then have  $y \notin Z$  using r-subset unfolding Z-def by auto
      then have  $(y, z) \in r$  using A unfolding r'-def by auto
      with trans True show ?thesis unfolding trans-def r'-def by blast
    next
      case False
      with  $A(1)$  have  $F: x \in Z \ \text{intv-const } (I x) < k x$  unfolding r'-def by
auto
      moreover from  $A(2)$  r-subset have  $z \in X \ \text{isIntv } (I' z)$ 
      by (auto simp: r'-def) (auto simp: I'-def Z-def)
      ultimately show ?thesis unfolding r'-def by auto
    qed
  qed
  show  $\forall x \in X. \ \text{valid-intv } (k x) (I' x)$ 
  proof (auto simp: I'-def intro: valid, goal-cases)
    case  $(1 x)$ 
    with assms(8) have  $\text{intv-const } (I x) < k x$  by (fastforce elim!: valid-intv.cases)
    then show ?case by auto
  qed
  qed

```

**lemma** *closest-prestable-2:*

```

fixes  $I X k r$ 
defines  $\mathcal{R} \equiv \{\text{region } X \ I r \mid I r. \ \text{valid-region } X \ k \ I r\}$ 
defines  $R \equiv \text{region } X \ I r$ 
assumes  $\forall x \in X. \ \neg \text{isConst } (I x)$ 
defines  $X_0 \equiv \{x \in X. \ \text{isIntv } (I x)\}$ 
defines  $M \equiv \{x \in X_0. \ \forall y \in X_0. (x, y) \in r \longrightarrow (y, x) \in r\}$ 
defines  $I' \equiv \lambda x. \ \text{if } x \notin M \ \text{then } I x \ \text{else } \text{Const } (\text{intv-const } (I x) + 1)$ 
defines  $r' \equiv \{(x, y) \in r. \ x \notin M \wedge y \notin M\}$ 
assumes finite X

```

**assumes** *valid-region*  $X$   $k$   $I$   $r$   
**assumes**  $M \neq \{\}$   
**shows**  $\forall v \in R. \forall t \geq 0. (v \oplus t) \notin R \longrightarrow (\exists t' \leq t. (v \oplus t') \in \text{region } X$   
 $I' r' \wedge t' \geq 0)$   
**and**  $\forall v \in \text{region } X I' r'. \forall t \geq 0. (v \oplus t) \notin R$   
**and**  $\forall v \in R. \forall t'. \{x. x \in X \wedge (\exists c. I' x = \text{Intv } c \wedge (v \oplus t') x + (t$   
 $- t') \geq \text{real } (c + 1))\}$   
 $= \{x. x \in X \wedge (\exists c. I x = \text{Intv } c \wedge v x + t \geq \text{real } (c +$   
 $1))\} - M$   
**and**  $\exists x \in X. \text{isConst } (I' x)$   
**proof** (*safe, goal-cases*)  
**fix**  $v$  **assume**  $v: v \in R$  **fix**  $t :: t$  **assume**  $t: t \geq 0$   $(v \oplus t) \notin R$   
**note**  $M = \text{assms}(10)$   
**then obtain**  $x$   $c$  **where**  $x: x \in M$   $I x = \text{Intv } c$   $x \in X$   $x \in X_0$  **unfolding**  
 $M\text{-def}$   $X_0\text{-def}$  **by force**  
**let**  $?t = 1 - \text{frac } (v x)$   
**let**  $?v = v \oplus ?t$   
**have**  $\text{elem}: \text{intv-elem } x$   $v$   $(I x)$  **if**  $x \in X$  **for**  $x$  **using** *that*  $v$  **unfolding**  
 $R\text{-def}$  **by auto**  
**from**  $\text{assms}(9)$  **have**  $*$ : *trans*  $r$  *total-on*  $\{x \in X. \exists d. I x = \text{Intv } d\}$   $r$  **by**  
*auto*  
**then have**  $\text{trans}[\text{intro}]$ :  $\bigwedge x y z. (x, y) \in r \implies (y, z) \in r \implies (x, z) \in r$   
**unfolding**  $\text{trans-def}$   
**by blast**  
**have**  $\{x \in X. \exists d. I x = \text{Intv } d\} = X_0$  **unfolding**  $X_0\text{-def}$  **by auto**  
**with**  $*(2)$  **have** *total: total-on*  $X_0$   $r$  **by auto**  
**{ fix**  $y$  **assume**  $y: y \notin M$   $y \in X_0$   
**have**  $\neg (x, y) \in r$  **using**  $x$   $y$  **unfolding**  $M\text{-def}$  **by auto**  
**moreover with** *total*  $x$   $y$  **have**  $(y, x) \in r$  **unfolding** *total-on-def* **by**  
*auto*  
**ultimately have**  $\neg (x, y) \in r \wedge (y, x) \in r ..$   
**}** **note**  $M\text{-max} = \text{this}$   
**{ fix**  $y$  **assume**  $T1: y \in M$   $x \neq y$   
**then have**  $T2: y \in X_0$  **unfolding**  $M\text{-def}$  **by auto**  
**with** *total*  $x$   $T1$  **have**  $(x, y) \in r \vee (y, x) \in r$  **by** (*auto simp: total-on-def*)  
**with**  $T1(1)$   $x(1)$  **have**  $(x, y) \in r$   $(y, x) \in r$  **unfolding**  $M\text{-def}$  **by auto**  
**}** **note**  $M\text{-eq} = \text{this}$   
**{ fix**  $y$  **assume**  $y: y \notin M$   $y \in X_0$   
**with**  $M\text{-max}$  **have**  $\neg (x, y) \in r$   $(y, x) \in r$  **by auto**  
**with**  $v[\text{unfolded } R\text{-def}]$   $X_0\text{-def}$   $x(4)$   $y(2)$  **have**  $\text{frac } (v y) < \text{frac } (v x)$   
**by auto**  
**then have**  $?t < 1 - \text{frac } (v y)$  **by auto**  
**}** **note**  $t\text{-bound}' = \text{this}$   
**{ fix**  $y$  **assume**  $y: y \in X_0$

```

have ?t ≤ 1 - frac (v y)
proof (cases x = y)
  case True thus ?thesis by simp
next
  case False
  have (y, x) ∈ r
  proof (cases y ∈ M)
    case False with M-max y show ?thesis by auto
  next
    case True with False M-eq y show ?thesis by auto
  qed
with v[unfolded R-def] X0-def x(4) y have frac (v y) ≤ frac (v x) by
auto
  then show ?t ≤ 1 - frac (v y) by auto
  qed
} note t-bound''' = this
have frac (v x) < 1 by (simp add: frac-lt-1)
then have ?t > 0 by (simp add: x(3))
{ fix c y fix t :: t assume y: y ∉ M I y = Intv c y ∈ X and t: t ≥ 0 t
≤ ?t
  then have y ∈ X0 unfolding X0-def by auto
  with t-bound' y have ?t < 1 - frac (v y) by auto
  with t have t < 1 - frac (v y) by auto
  moreover from elem[OF ⟨y ∈ X⟩] y have c < v y v y < c + 1 by
auto
  ultimately have (v y + t) < c + 1 using frac-add-le-preservation by
blast
  with ⟨c < v y⟩ t have intv-elem y (v ⊕ t) (I y) by (auto simp:
cval-add-def y)
} note t-bound = this
from elem[OF x(3)] x(2) have v-x: c < v x v x < c + 1 by auto
then have floor (v x) = c by linarith
then have shift: v x + ?t = c + 1 unfolding frac-def by auto
have v x + t ≥ c + 1
proof (rule ccontr, goal-cases)
  case 1
  then have AA: v x + t < c + 1 by simp
  with shift have lt: t < ?t by auto
  let ?v = v ⊕ t
  have ?v ∈ region X I r
  proof (standard, goal-cases)
    case 1
    from v have ∀ x ∈ X. v x ≥ 0 unfolding R-def by auto
    with t show ?case unfolding cval-add-def by auto
  
```

```

next
  case 2
  show ?case
  proof (safe, goal-cases)
    case (1 y)
    note A = this
    with elem have e: intv-elem y v (I y) by auto
    show ?case
    proof (cases y ∈ M)
      case False
      then have [simp]: I' y = I y by (auto simp: I'-def)
      show ?thesis
      proof (cases I y, goal-cases)
        case 1 with assms(3) A show ?case by auto
      next
        case (2 c)
        from t-bound[OF False this A t(1)] lt show ?case by (auto simp:
cval-add-def 2)
      next
        case (3 c)
        with e have v y > c by auto
        with 3 t(1) show ?case by (auto simp: cval-add-def)
      qed
    next
      case True
      then have y ∈ X0 by (auto simp: M-def)
      note T = this True
      show ?thesis
      proof (cases x = y)
        case False
        with M-eq T have (x, y) ∈ r (y, x) ∈ r by presburger+
        with v[unfolded R-def] X0-def x(4) T(1) have *: frac (v y) =
frac (v x) by auto
        from T(1) obtain c where c: I y = Intv c by (auto simp:
X0-def)
        with elem T(1) have c < v y v y < c + 1 by (fastforce simp:
X0-def)+
        then have floor (v y) = c by linarith
        with * lt have (v y + t) < c + 1 unfolding frac-def by auto
        with ⟨c < v y⟩ t show ?thesis by (auto simp: c cval-add-def)
      next
        case True with ⟨c < v x⟩ t AA x show ?thesis by (auto simp:
cval-add-def)
      qed
    qed
  qed

```

```

    qed
  qed
next
  show  $X_0 = \{x \in X. \exists d. I x = Intv d\}$  by (auto simp add:  $X_0$ -def)
next
  have  $t > 0$ 
  proof (rule ccontr, goal-cases)
    case 1 with  $t v$  show False unfolding cval-add-def by auto
  qed
  show  $\forall y \in X_0. \forall z \in X_0. ((y, z) \in r) = (\text{frac}((v \oplus t)y) \leq \text{frac}((v \oplus t)z))$ 
  proof (auto simp:  $X_0$ -def, goal-cases)
    case (1  $y z d d'$ )
    note  $A = this$ 
    from  $A$  have [simp]:  $y \in X_0 z \in X_0$  unfolding  $X_0$ -def  $I'$ -def by
  auto
    from  $A$  v[unfolded  $R$ -def] have  $le: \text{frac}(v y) \leq \text{frac}(v z)$  by (auto
  simp:  $r'$ -def)
    from  $t$ -bound''' have  $?t \leq 1 - \text{frac}(v y) ?t \leq 1 - \text{frac}(v z)$  by
  auto
    with  $lt$  have  $t < 1 - \text{frac}(v y) t < 1 - \text{frac}(v z)$  by auto
    with frac-distr[OF  $\langle t > 0 \rangle$ ] have
       $\text{frac}(v y) + t = \text{frac}(v y + t) \text{frac}(v z) + t = \text{frac}(v z + t)$ 
    by auto
    with  $le$  show ?case by (auto simp: cval-add-def)
  next
    case (2  $y z d d'$ )
    note  $A = this$ 
    from  $A$  have [simp]:  $y \in X_0 z \in X_0$  unfolding  $X_0$ -def by auto
    from  $t$ -bound''' have  $?t \leq 1 - \text{frac}(v y) ?t \leq 1 - \text{frac}(v z)$  by
  auto
    with  $lt$  have  $t < 1 - \text{frac}(v y) t < 1 - \text{frac}(v z)$  by auto
    from frac-add-leD[OF  $\langle t > 0 \rangle$  this]  $A(5)$  have
       $\text{frac}(v y) \leq \text{frac}(v z)$ 
    by (auto simp: cval-add-def)
    with v[unfolded  $R$ -def]  $A$  show ?case by auto
  qed
  qed
  with  $t$   $R$ -def show False by simp
  qed
  with shift have  $t \geq ?t$  by simp
  let ? $R = \text{region } X I' r'$ 
  let ? $X_0 = \{x \in X. \exists d. I' x = Intv d\}$ 
  have  $(v \oplus ?t) \in ?R$ 

```

```

proof (standard, goal-cases)
  case 1
  from  $v$  have  $\forall x \in X. v x \geq 0$  unfolding R-def by auto
  with  $\langle ?t > 0 \rangle t$  show ?case unfolding cval-add-def by auto
next
  case 2
  show ?case
  proof (safe, goal-cases)
    case (1  $y$ )
    note  $A = \text{this}$ 
    with elem have  $e: \text{intv-elm } y v (I y)$  by auto
    show ?case
    proof (cases  $y \in M$ )
      case False
      then have [simp]:  $I' y = I y$  by (auto simp: I'-def)
      show ?thesis
      proof (cases  $I y$ , goal-cases)
        case 1 with assms(3)  $A$  show ?case by auto
      next
        case (2  $c$ )
        from t-bound[OF False this A]  $\langle ?t > 0 \rangle$  show ?case by (auto simp:
cval-add-def 2)
      next
        case (3  $c$ )
        with  $e$  have  $v y > c$  by auto
        with 3  $\langle ?t > 0 \rangle$  show ?case by (auto simp: cval-add-def)
    qed
  next
  case True
  then have  $y \in X_0$  by (auto simp: M-def)
  note  $T = \text{this True}$ 
  show ?thesis
  proof (cases  $x = y$ )
    case False
    with M-eq  $T(2)$  have  $(x, y) \in r (y, x) \in r$  by auto
    with  $v$ [unfolded R-def]  $X_0$ -def  $x(4)$   $T(1)$  have  $*$ :  $\text{frac } (v y) = \text{frac}$ 
( $v x$ ) by auto
    from  $T(1)$  obtain  $c$  where  $c: I y = \text{Intv } c$  by (auto simp: X0-def)
    with elem  $T(1)$  have  $c < v y v y < c + 1$  by (fastforce simp:
X0-def)
    then have  $\text{floor } (v y) = c$  by linarith
    with  $*$  have  $(v y + ?t) = c + 1$  unfolding frac-def by auto
    with  $T(2)$  show ?thesis by (auto simp: c cval-add-def I'-def)
  next

```

```

      case True with shift x show ?thesis by (auto simp: cval-add-def
I'-def)
    qed
  qed
  qed
  next
  show ?X0 = ?X0 ..
  next
  show  $\forall y \in ?X_0. \forall z \in ?X_0. ((y, z) \in r') = (\text{frac}((v \oplus 1 - \text{frac}(v x))y) \leq \text{frac}((v \oplus 1 - \text{frac}(v x)) z))$ 
  proof (safe, goal-cases)
    case (1 y z d d')
    note A = this
    then have  $y \notin M z \notin M$  unfolding I'-def by auto
    with A have [simp]:  $I' y = I y I' z = I z y \in X_0 z \in X_0$  unfolding
X0-def I'-def by auto
    from A v[unfolded R-def] have le:  $\text{frac}(v y) \leq \text{frac}(v z)$  by (auto
simp: r'-def)
    from t-bound'  $\langle y \notin M \rangle \langle z \notin M \rangle$  have  $?t < 1 - \text{frac}(v y) ?t < 1 -$ 
frac(v z) by auto
    with frac-distr[OF  $\langle ?t > 0 \rangle$ ] have
       $\text{frac}(v y) + ?t = \text{frac}(v y + ?t) \text{frac}(v z) + ?t = \text{frac}(v z + ?t)$ 
    by auto
    with le show ?case by (auto simp: cval-add-def)
  next
  case (2 y z d d')
  note A = this
  then have  $M: y \notin M z \notin M$  unfolding I'-def by auto
  with A have [simp]:  $I' y = I y I' z = I z y \in X_0 z \in X_0$  unfolding
X0-def I'-def by auto
  from t-bound'  $\langle y \notin M \rangle \langle z \notin M \rangle$  have  $?t < 1 - \text{frac}(v y) ?t < 1 -$ 
frac(v z) by auto
  from frac-add-leD[OF  $\langle ?t > 0 \rangle$  this] A(5) have
     $\text{frac}(v y) \leq \text{frac}(v z)$ 
  by (auto simp: cval-add-def)
  with v[unfolded R-def] A M show ?case by (auto simp: r'-def)
  qed
  qed
  with  $\langle ?t > 0 \rangle \langle ?t \leq t \rangle$  show  $\exists t' \leq t. (v \oplus t') \in \text{region } X I' r' \wedge 0 \leq t'$ 
by auto
  next
  fix v t assume A:  $v \in \text{region } X I' r' 0 \leq t (v \oplus t) \in R$ 
  from assms(10) obtain x c where x:
     $x \in X_0 I x = \text{Intv } c x \in X x \in M$ 

```

**unfolding**  $M$ -def  $X_0$ -def **by force**  
**with**  $A(1)$  **have**  $\text{intv-elem } x \ v \ (I' \ x)$  **by auto**  
**with**  $x$  **have**  $v \ x = c + 1$  **unfolding**  $I'$ -def **by auto**  
**moreover from**  $A(3)$   $x(2,3)$  **have**  $v \ x + t < c + 1$  **by** (*fastforce simp: cval-add-def R-def*)  
**ultimately show**  $\text{False}$  **using**  $A(2)$  **by auto**  
**next**  
**case**  $A: (\exists \ v \ t' \ x \ c)$   
**from**  $A(3)$  **have**  $I \ x = \text{Intv } c$  **by** (*auto simp: I'-def*) (*cases*  $x \in M$ , *auto*)  
**with**  $A(4)$  **show**  $?case$  **by** (*auto simp: cval-add-def*)  
**next**  
**case**  $\not 4$   
**then show**  $?case$  **unfolding**  $I'$ -def **by auto**  
**next**  
**case**  $A: (\exists \ v \ t' \ x \ c)$   
**then have**  $I' \ x = \text{Intv } c$  **unfolding**  $I'$ -def **by auto**  
**moreover from**  $A$  **have**  $\text{real } (c + 1) \leq (v \oplus t') \ x + (t - t')$  **by** (*auto simp: cval-add-def*)  
**ultimately show**  $?case$  **by blast**  
**next**  
**from** *assms(5,10)* **obtain**  $x$  **where**  $x: x \in M$  **by blast**  
**then have**  $\text{isConst } (I' \ x)$  **by** (*auto simp: I'-def*)  
**with**  $x$  **show**  $\exists x \in X. \text{isConst } (I' \ x)$  **unfolding**  $M$ -def  $X_0$ -def **by force**  
**qed**

**lemma** *closest-valid-2:*

**fixes**  $I \ X \ k \ r$   
**defines**  $\mathcal{R} \equiv \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$   
**defines**  $R \equiv \text{region } X \ I \ r$   
**assumes**  $\forall x \in X. \neg \text{isConst } (I \ x)$   
**defines**  $X_0 \equiv \{x \in X. \text{isIntv } (I \ x)\}$   
**defines**  $M \equiv \{x \in X_0. \forall y \in X_0. (x, y) \in r \longrightarrow (y, x) \in r\}$   
**defines**  $I' \equiv \lambda x. \text{if } x \notin M \text{ then } I \ x \text{ else } \text{Const } (\text{intv-const } (I \ x) + 1)$   
**defines**  $r' \equiv \{(x, y) \in r. x \notin M \wedge y \notin M\}$   
**assumes** *finite*  $X$   
**assumes** *valid-region*  $X \ k \ I \ r$   
**assumes**  $M \neq \{\}$   
**shows** *valid-region*  $X \ k \ I' \ r'$

**proof**

**let**  $?X_0 = \{x \in X. \exists d. I \ x = \text{Intv } d\}$   
**let**  $?X_0' = \{x \in X. \exists d. I' \ x = \text{Intv } d\}$   
**show**  $?X_0' = ?X_0' ..$   
**have**  $r$ -subset:  $r \subseteq ?X_0 \times ?X_0$  **and** *refl*: *refl-on*  $?X_0 \ r$  **and** *total*: *total-on*  $?X_0 \ r$  **and**

```

    trans: trans r and valid:  $\bigwedge x. x \in X \implies \text{valid-intv } (k\ x) (I\ x)$ 
    using assms(9) by auto
have subs:  $r' \subseteq r$  unfolding r'-def by auto
show  $r' \subseteq ?X_0' \times ?X_0'$  using r-subset unfolding r'-def I'-def by auto

show refl-on  $?X_0'$  r' unfolding refl-on-def
proof auto
  fix x d assume A:  $x \in X\ I'\ x = \text{Intv } d$ 
  then have  $x \notin M$  by (force simp: I'-def)
  with A have  $I\ x = \text{Intv } d$  by (force simp: I'-def)
  with A refl have  $(x, x) \in r$  by (auto simp: refl-on-def)
  then show  $(x, x) \in r'$  by (auto simp: r'-def  $\langle x \notin M \rangle$ )
qed
show total-on  $?X_0'$  r' unfolding total-on-def
proof (safe, goal-cases)
  case (1 x y d d')
  note A = this
  then have  $*: x \notin M\ y \notin M$  by (force simp: I'-def)+
  with A have  $I\ x = \text{Intv } d\ I\ y = \text{Intv } d'$  by (force simp: I'-def)+
  with A total have  $(x, y) \in r \vee (y, x) \in r$  by (auto simp: total-on-def)
  with A(6) * show ?case unfolding r'-def by auto
qed
show trans r' unfolding trans-def
proof safe
  fix x y z assume A:  $(x, y) \in r'\ (y, z) \in r'$ 
  from trans have [intro]:
     $\bigwedge x\ y\ z. (x, y) \in r \implies (y, z) \in r \implies (x, z) \in r$ 
  unfolding trans-def by blast
  from A show  $(x, z) \in r'$  by (auto simp: r'-def)
qed
show  $\forall x \in X. \text{valid-intv } (k\ x) (I'\ x)$ 
using valid unfolding I'-def
proof (auto simp: I'-def intro: valid, goal-cases)
  case (1 x)
  with assms(9) have intv-const  $(I\ x) < k\ x$  by (fastforce simp: M-def
X0-def)
  then show ?case by auto
qed
qed

```

### 5.3.1 Putting the Proof for the 'Set of Regions' Property Together

Misc lemma *total-finite-trans-max*:

$X \neq \{\} \implies \text{finite } X \implies \text{total-on } X \ r \implies \text{trans } r \implies \exists x \in X. \forall y \in X. x \neq y \longrightarrow (y, x) \in r$

**proof** (*induction card X arbitrary: X*)

**case** 0

**then show** ?case **by auto**

**next**

**case** (Suc n)

**then obtain** x **where** x: x ∈ X **by blast**

**show** ?case

**proof** (cases n = 0)

**case** True

**with** Suc.hyps(2) ⟨finite X⟩ x **have** X = {x} **by** (metis card-Suc-eq empty-iff insertE)

**then show** ?thesis **by auto**

**next**

**case** False

**show** ?thesis

**proof** (cases ∀y∈X. x ≠ y ⟶ (y, x) ∈ r)

**case** True **with** x **show** ?thesis **by auto**

**next**

**case** False

**then obtain** y **where** y: y ∈ X x ≠ y ∧ (y, x) ∈ r **by auto**

**with** x Suc.prem(3) **have** (x, y) ∈ r **unfolding** total-on-def **by blast**

**let** ?X = X - {x}

**have** tot: total-on ?X r **using** ⟨total-on X r⟩ **unfolding** total-on-def **by auto**

**from** x Suc.hyps(2) ⟨finite X⟩ **have** card: n = card ?X **by auto**

**with** ⟨finite X⟩ ⟨n ≠ 0⟩ **have** ?X ≠ {} **by auto**

**from** Suc.hyps(1)[OF card this - tot ⟨trans r⟩] ⟨finite X⟩ **obtain** x'

**where**

IH: x' ∈ ?X ∀ y ∈ ?X. x' ≠ y ⟶ (y, x') ∈ r

**by auto**

**have** (x', x) ∉ r

**proof** (rule ccontr, auto)

**assume** A: (x', x) ∈ r

**with** y(3) **have** x' ≠ y **by auto**

**with** y IH **have** (y, x') ∈ r **by auto**

**with** ⟨trans r⟩ A **have** (y, x) ∈ r **unfolding** trans-def **by blast**

**with** y **show** False **by auto**

**qed**

**with** ⟨x ∈ X⟩ ⟨x' ∈ ?X⟩ ⟨total-on X r⟩ **have** (x, x') ∈ r **unfolding** total-on-def **by auto**

**with** IH **show** ?thesis **by auto**

**qed**

qed  
qed

**lemma** *card-mono-strict-subset*:

*finite A  $\implies$  finite B  $\implies$  finite C  $\implies$   $A \cap B \neq \{\}$   $\implies$   $C = A - B \implies$   
*card C < card A**

**by** (*metis Diff-disjoint Diff-subset inf-commute less-le psubset-card-mono*)

**Proof** First we show that a shift by a non-negative integer constant means that any two valuations from the same region are being shifted to the same region.

**lemma** *int-shift-equiv*:

**fixes**  $X k$  **fixes**  $t :: int$

**defines**  $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$

**assumes**  $v \in R\ v' \in R\ R \in \mathcal{R}\ t \geq 0$

**shows**  $(v' \oplus t) \in [v \oplus t]_{\mathcal{R}}$  **using** *assms*

**proof** –

**from** *assms* **obtain**  $I\ r$  **where**  $A: R = region\ X\ I\ r\ valid-region\ X\ k\ I\ r$   
**by** *auto*

**from** *regions-closed*[*OF - assms*(4,2), *of X k t*] *assms*(1,5) **obtain**  $I' r'$   
**where**  $RR:$

$[v \oplus t]_{\mathcal{R}} = region\ X\ I' r' valid-region\ X\ k\ I' r'$

**by** *auto*

**from** *regions-closed'*[*OF - assms*(4,2), *of X k t*] *assms*(1,5) **have**  $RR': (v \oplus t) \in [v \oplus t]_{\mathcal{R}}$  **by** *auto*

**show** *?thesis*

**proof** (*simp add: RR(1), rule, goal-cases*)

**case** 1

**from**  $\langle v' \in R \rangle A(1)$  **have**  $\forall x \in X. 0 \leq v' x$  **by** *auto*

**with**  $\langle t \geq 0 \rangle$  **show** *?case unfolding eval-add-def by auto*

**next**

**case** 2

**show** *?case*

**proof** *safe*

**fix**  $x$  **assume**  $x: x \in X$

**with**  $\langle v' \in R \rangle \langle v \in R \rangle A(1)$  **have**  $I: intv-elem\ x\ v\ (I\ x)\ intv-elem\ x\ v'$   
 $(I\ x)$  **by** *auto*

**from**  $x\ RR\ RR'$  **have**  $I': intv-elem\ x\ (v \oplus t)\ (I' x)$  **by** *auto*

**show**  $intv-elem\ x\ (v' \oplus t)\ (I' x)$

**proof** (*cases I' x*)

**case** (*Const c*)

**from** *Const I'* **have**  $v\ x + t = c$  **unfolding** *eval-add-def by auto*

**with**  $x\ A(1)\ \langle v \in R \rangle \langle t \geq 0 \rangle$  **have**  $*$ :  $v\ x = c - nat\ t\ t \leq c$  **by**

```

fastforce+
  have  $I x = \text{Const } (c - \text{nat } t)$ 
  proof (cases  $I x$ )
    case (Greater  $c'$ )
      with  $RR(2) \text{ Const } \langle x \in X \rangle$  have  $c \leq k x$  by fastforce
      with  $* \langle t \geq 0 \rangle$  have  $*: v x \leq k x$  by auto
      from Greater  $A(2) \langle x \in X \rangle$  have  $c' = k x$  by fastforce
      moreover from  $I(1) \text{ Greater}$  have  $v x > c'$  by auto
      ultimately show ?thesis
        using  $\langle c \leq k x \rangle *$  by auto
    qed (use  $I$  in  $\langle \text{auto simp: } * \rangle$ )
  with  $I \langle t \geq 0 \rangle *(2)$  have  $v' x + t = c$  by auto
  with Const show ?thesis unfolding cval-add-def by auto
next
case (Intv  $c$ )
with  $I'$  have  $c < v x + t$   $v x + t < c + 1$  unfolding cval-add-def
by auto
with  $x A(1) \langle v \in R \rangle \langle t \geq 0 \rangle$ 
have  $*: c - \text{nat } t < v x$   $v x < c - \text{nat } t + 1$   $t \leq c$ 
  by fastforce+
have  $I x = \text{Intv } (c - \text{nat } t)$ 
proof (cases  $I x$ )
  case (Greater  $c'$ )
    with  $RR(2) \text{ Intv } \langle x \in X \rangle$  have  $c \leq k x$  by fastforce
    with  $*$  have  $*: v x \leq k x$  using Intv  $RR(2) x$  by fastforce
    from Greater  $A(2) \langle x \in X \rangle$  have  $c' = k x$  by fastforce
    moreover from  $I(1) \text{ Greater}$  have  $v x > c'$  by auto
    ultimately show ?thesis
      using  $\langle c \leq k x \rangle *$  by auto
  qed (use  $I *$  in  $\langle \text{auto simp del: of-nat-diff} \rangle$ )
  with  $I \langle t \leq c \rangle$  have  $c < v' x + \text{nat } t$   $v' x + t < c + 1$  by auto
  with Intv  $\langle t \geq 0 \rangle$  show ?thesis unfolding cval-add-def by auto
next
case (Greater  $c$ )
with  $I'$  have  $*: c < v x + t$  unfolding cval-add-def by auto
show ?thesis
proof (cases  $I x$ )
  case (Const  $c'$ )
    with  $x A(1) I(2) \langle v \in R \rangle \langle v' \in R \rangle$  have  $v x = v' x$  by fastforce
    with Greater  $*$  show ?thesis unfolding cval-add-def by auto
  next
  case (Intv  $c'$ )
    with  $x A(1) I(2) \langle v \in R \rangle \langle v' \in R \rangle$  have  $**: c' < v x$   $v x < c' +$ 
1  $c' < v' x$ 

```

```

    by fastforce+
    then have  $c' + t < v x + t$   $v x + t < c' + t + 1$  by auto
    with * have  $c \leq c' + t$  by auto
    with  $**(3)$  have  $v' x + t > c$  by auto
    with Greater * show ?thesis unfolding cval-add-def by auto
  next
    fix  $c'$  assume  $c': I x = \text{Greater } c'$ 
    with  $x A(1) I(2) \langle v \in R \rangle \langle v' \in R \rangle$  have  $** : c' < v x$   $c' < v' x$  by
fastforce+
    from Greater RR(2)  $c' A(2) \langle x \in X \rangle$  have  $c' = k x$   $c = k x$  by
fastforce+
    with  $\langle t \geq 0 \rangle ** (2)$  Greater show intv-elem  $x (v' \oplus \text{real-of-int } t)$ 
( $I' x$ )
    unfolding cval-add-def by auto
  qed
qed
qed
next
show  $\{x \in X. \exists d. I' x = \text{Intv } d\} = \{x \in X. \exists d. I' x = \text{Intv } d\} ..$ 
next
let  $?X_0 = \{x \in X. \exists d. I' x = \text{Intv } d\}$ 
{ fix  $x y :: \text{real}$ 
  have  $\text{frac } (x + t) \leq \text{frac } (y + t) \iff \text{frac } x \leq \text{frac } y$  by (simp add:
frac-def)
} note frac-equiv = this
{ fix  $x y$ 
  have  $\text{frac } ((v \oplus t) x) \leq \text{frac } ((v \oplus t) y) \iff \text{frac } (v x) \leq \text{frac } (v y)$ 
  unfolding cval-add-def using frac-equiv by auto
} note frac-equiv' = this
{ fix  $x y$ 
  have  $\text{frac } ((v' \oplus t) x) \leq \text{frac } ((v' \oplus t) y) \iff \text{frac } (v' x) \leq \text{frac } (v' y)$ 
  unfolding cval-add-def using frac-equiv by auto
} note frac-equiv'' = this
{ fix  $x y$  assume  $x: x \in X$  and  $y: y \in X$  and  $B: \neg \text{isGreater}(I x) \neg$ 
isGreater( $I y$ )
  have  $\text{frac } (v x) \leq \text{frac } (v y) \iff \text{frac } (v' x) \leq \text{frac } (v' y)$ 
  proof (cases  $I x$ )
    case (Const  $c$ )
      with  $x \langle v \in R \rangle \langle v' \in R \rangle A(1)$  have  $v x = c$   $v' x = c$  by fastforce+
      then have  $\text{frac } (v x) \leq \text{frac } (v y)$   $\text{frac } (v' x) \leq \text{frac } (v' y)$  unfolding
frac-def by simp+
      then show ?thesis by auto
  next
    case (Intv  $c$ )

```

```

with  $x \langle v \in R \rangle A(1)$  have  $v: c < v x \ v x < c + 1$  by fastforce+
from  $\text{Intv } x \langle v' \in R \rangle A(1)$  have  $v': c < v' x \ v' x < c + 1$  by
fastforce+
show ?thesis
proof (cases I y, goal-cases)
  case (Const c')
    with  $y \langle v \in R \rangle \langle v' \in R \rangle A(1)$  have  $v y = c' \ v' y = c'$  by fastforce+
    then have  $\text{frac } (v y) = 0 \ \text{frac } (v' y) = 0$  by auto
    with  $\text{nat-intv-frac-gt0}[OF \ v] \ \text{nat-intv-frac-gt0}[OF \ v']$ 
      have  $\neg \text{frac } (v x) \leq \text{frac } (v y) \ \neg \text{frac } (v' x) \leq \text{frac } (v' y)$  by
linarith+
    then show ?thesis by auto
  next
    case 2: (Intv c')
    with  $x \ y \ \text{Intv } \langle v \in R \rangle \langle v' \in R \rangle A(1)$  have
       $(x, y) \in r \iff \text{frac } (v x) \leq \text{frac } (v y)$ 
       $(x, y) \in r \iff \text{frac } (v' x) \leq \text{frac } (v' y)$ 
    by auto
    then show ?thesis by auto
  next
    case Greater
    with  $B$  show ?thesis by auto
  qed
next
  case Greater with  $B$  show ?thesis by auto
qed
} note frac-cong = this
have not-greater:  $\neg \text{isGreater } (I x)$  if  $x: x \in X \ \neg \text{isGreater } (I' x)$  for  $x$ 
proof (rule ccontr, auto, goal-cases)
  case (1 c)
    with  $x \langle v \in R \rangle A(1,2)$  have  $c < v x$  by fastforce+
    moreover from  $x \ A(2) \ 1$  have  $c = k x$  by fastforce+
    ultimately have  $*$ :  $k x < v x + t$  using  $\langle t \geq 0 \rangle$  by simp
    from  $RR(1,2) \ RR' x$  have  $I': \text{intv-elim } x \ (v \oplus t) \ (I' x) \ \text{valid-intv } (k$ 
x) \ (I' x) by auto
    from  $x$  show False
    proof (cases I' x, auto)
      case (Const c')
        with  $I' *$  show False by (auto simp: cval-add-def)
      next
        case (Intv c')
          with  $I' *$  show False by (auto simp: cval-add-def)
    qed
  qed

```

**show**  $\forall x \in ?X_0. \forall y \in ?X_0. ((x, y) \in r^\wedge) = (\text{frac}((v' \oplus t) x) \leq \text{frac}((v' \oplus t) y))$   
**proof** (*standard, standard*)  
**fix**  $x y$  **assume**  $x: x \in ?X_0$  **and**  $y: y \in ?X_0$   
**then have**  $B: \neg \text{isGreater}(I' x) \neg \text{isGreater}(I' y)$  **by auto**  
**with**  $x y$  **not-greater** **have**  $\neg \text{isGreater}(I x) \neg \text{isGreater}(I y)$  **by auto**  
**with**  $x y$  **frac-cong** **have**  $\text{frac}(v x) \leq \text{frac}(v y) \longleftrightarrow \text{frac}(v' x) \leq \text{frac}(v' y)$  **by auto**  
**moreover from**  $x y$   $RR(1) RR'$  **have**  $(x, y) \in r' \longleftrightarrow \text{frac}((v \oplus t) x) \leq \text{frac}((v \oplus t) y)$   
**by fastforce**  
**ultimately show**  $(x, y) \in r' \longleftrightarrow \text{frac}((v' \oplus t) x) \leq \text{frac}((v' \oplus t) y)$   
**using** *frac-equiv' frac-equiv''* **by blast**  
**qed**  
**qed**  
**qed**

Now, we can use the 'immediate' induction proposed by P. Bouyer for shifts smaller than one. The induction principle is not at all obvious: the induction is over the set of clocks for which the valuation is shifted beyond the current interval boundaries. Using the two successor operations, we can see that either the set of these clocks remains the same ( $Z =$ ) or strictly decreases ( $Z =$ ).

**lemma** *set-of-regions-lt-1*:

**fixes**  $X k I r t v$   
**defines**  $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$   
**defines**  $C \equiv \{x. x \in X \wedge (\exists c. I x = \text{Intv } c \wedge v x + t \geq c + 1)\}$   
**assumes** *valid-region*  $X k I r v \in \text{region } X I r v' \in \text{region } X I r$  *finite*  $X$   
 $0 \leq t < 1$   
**shows**  $\exists t' \geq 0. (v' \oplus t') \in [v \oplus t]_{\mathcal{R}}$  **using** *assms*  
**proof** (*induction card C arbitrary: C I r v v' t rule: less-induct*)  
**case** *less*  
**let**  $?R = \text{region } X I r$   
**let**  $?C = \{x \in X. \exists c. I x = \text{Intv } c \wedge \text{real}(c + 1) \leq v x + t\}$   
**from** *less* **have**  $R: ?R \in \mathcal{R}$  **by auto**  
**{ fix**  $v I k r$  **fix**  $t :: t$   
**assume** *no-consts*:  $\forall x \in X. \neg \text{isConst}(I x)$   
**assume**  $v: v \in \text{region } X I r$   
**assume**  $t: t \geq 0$   
**let**  $?C = \{x \in X. \exists c. I x = \text{Intv } c \wedge \text{real}(c + 1) \leq v x + t\}$   
**assume**  $C: ?C = \{\}$   
**let**  $?R = \text{region } X I r$   
**have**  $(v \oplus t) \in ?R$   
**proof** (*rule, goal-cases*)

```

case 1
with  $\langle t \geq 0 \rangle \langle v \in ?R \rangle$  show  $?case$  by (auto simp: cval-add-def)
next
case 2
show  $?case$ 
proof (standard, case-tac I x, goal-cases)
  case (1  $x c$ )
  with no-consts show  $?case$  by auto
next
  case (2  $x c$ )
  with  $\langle v \in ?R \rangle$  have  $c < v x$  by fastforce
  with  $\langle t \geq 0 \rangle$  have  $c < v x + t$  by auto
  moreover from 2  $C$  have  $v x + t < c + 1$  by fastforce
  ultimately show  $?case$  by (auto simp: 2 cval-add-def)
next
  case (3  $x c$ )
  with  $\langle v \in ?R \rangle$  have  $c < v x$  by fastforce
  with  $\langle t \geq 0 \rangle$  have  $c < v x + t$  by auto
  then show  $?case$  by (auto simp: 3 cval-add-def)
qed
next
  show  $\{x \in X. \exists d. I x = Intv d\} = \{x \in X. \exists d. I x = Intv d\} ..$ 
next
  let  $?X_0 = \{x \in X. \exists d. I x = Intv d\}$ 
  { fix  $x d :: real$  fix  $c :: nat$  assume  $A: c < x x + d < c + 1 d \geq 0$ 
    then have  $d < 1 - frac\ x$  unfolding frac-def using floor-eq3
    of-nat-Suc by fastforce
  } note intv-frac = this
  { fix  $x$  assume  $x: x \in ?X_0$ 
    then obtain  $c$  where  $x \in X I x = Intv c$  by auto
    with  $\langle v \in ?R \rangle$  have  $*$ :  $c < v x$  by fastforce
    with  $\langle t \geq 0 \rangle$  have  $c < v x + t$  by auto
    from  $x C$  have  $v x + t < c + 1$  by auto
    from intv-frac[OF * this  $\langle t \geq 0 \rangle$ ] have  $t < 1 - frac\ (v x)$  by auto
  } note intv-frac = this
  { fix  $x y$  assume  $x: x \in ?X_0$  and  $y: y \in ?X_0$ 
    from frac-add-leIFF[OF  $\langle t \geq 0 \rangle$  intv-frac[OF  $x$ ] intv-frac[OF  $y$ ]]
    have  $frac\ (v x) \leq frac\ (v y) \longleftrightarrow frac\ ((v \oplus t) x) \leq frac\ ((v \oplus t) y)$ 
    by (auto simp: cval-add-def)
  } note frac-cong = this
  show  $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow frac\ ((v \oplus t) x) \leq frac\ ((v \oplus t) y)$ 
  proof (standard, standard, goal-cases)
    case (1  $x y$ )

```

**with**  $\langle v \in ?R \rangle$  **have**  $(x, y) \in r \iff \text{frac}(v x) \leq \text{frac}(v y)$  **by** *auto*  
**with** *frac-cong*[*OF 1*] **show** *?case* **by** *simp*  
**qed**  
**qed**  
**}** **note** *critical-empty-intro* = *this*  
**{** **assume** *const*:  $\exists x \in X. \text{isConst}(I x)$   
**assume** *t*:  $t > 0$   
**from** *const* **have**  $\{x \in X. \exists c. I x = \text{Const } c\} \neq \{\}$  **by** *auto*  
**from** *closest-prestable-1*[*OF this less.prem*(4) *less*(3)] *R* *closest-valid-1*[*OF this less.prem*(4) *less*(3)]  
**obtain**  $I'' r''$   
**where** *stability*:  $\forall v \in ?R. \forall t > 0. \exists t' \leq t. (v \oplus t') \in \text{region } X I'' r''$   
 $\wedge t' \geq 0$   
**and** *succ-not-refl*:  $\forall v \in \text{region } X I'' r''. \forall t \geq 0. (v \oplus t) \notin ?R$   
**and** *no-consts*:  $\forall x \in X. \neg \text{isConst}(I'' x)$   
**and** *crit-mono*:  $\forall v \in ?R. \forall t < 1. \forall t' \geq 0. (v \oplus t') \in \text{region } X I'' r''$   
 $\longrightarrow \{x. x \in X \wedge (\exists c. I x = \text{Intv } c \wedge v x + t \geq c + 1)\}$   
 $= \{x. x \in X \wedge (\exists c. I'' x = \text{Intv } c \wedge (v \oplus t') x + (t - t') \geq c + 1)\}$   
**and** *succ-valid*: *valid-region*  $X k I'' r''$   
**by** *auto*  
**let**  $?R'' = \text{region } X I'' r''$   
**from** *stability less*(4)  $\langle t > 0 \rangle$  **obtain** *t1* **where** *t1*:  $t1 \geq 0 \ t1 \leq t \ (v \oplus t1) \in ?R''$  **by** *auto*  
**from** *stability less*(5)  $\langle t > 0 \rangle$  **obtain** *t2* **where** *t2*:  $t2 \geq 0 \ t2 \leq t \ (v' \oplus t2) \in ?R''$  **by** *auto*  
**let**  $?v = v \oplus t1$   
**let**  $?t = t - t1$   
**let**  $?C' = \{x \in X. \exists c. I'' x = \text{Intv } c \wedge \text{real}(c + 1) \leq ?v x + ?t\}$   
**from** *t1*  $\langle t < 1 \rangle$  **have** *tt*:  $0 \leq ?t \ ?t < 1$  **by** *auto*  
**from** *crit-mono*  $\langle t < 1 \rangle$  *t1*(1,3)  $\langle v \in ?R \rangle$  **have** *crit*:  
 $?C = ?C'$   
**by** *auto*  
**with** *t1 t2 succ-valid no-consts* **have**  
 $\exists t1 \geq 0. \exists t2 \geq 0. \exists I' r'. t1 \leq t \wedge (v \oplus t1) \in \text{region } X I' r'$   
 $\wedge t2 \leq t \wedge (v' \oplus t2) \in \text{region } X I' r'$   
 $\wedge \text{valid-region } X k I' r'$   
 $\wedge (\forall x \in X. \neg \text{isConst}(I' x))$   
 $\wedge ?C = \{x \in X. \exists c. I' x = \text{Intv } c \wedge \text{real}(c + 1) \leq (v \oplus t1) x + (t - t1)\}$   
**by** *blast*  
**}** **note** *const-dest* = *this*

**{ fix  $t :: \text{real fix } v \text{ I r x c v'}$**   
**let  $?R = \text{region } X \text{ I r}$**   
**assume  $v: v \in ?R$**   
**assume  $v': v' \in ?R$**   
**assume  $\text{valid}: \text{valid-region } X \text{ k I r}$**   
**assume  $t: t > 0 \ t < 1$**   
**let  $?C = \{x \in X. \exists c. I x = \text{Intv } c \wedge \text{real } (c + 1) \leq v x + t\}$**   
**assume  $C: ?C = \{\}$**   
**assume  $\text{const}: \exists x \in X. \text{isConst } (I x)$**   
**then have  $\{x \in X. \exists c. I x = \text{Const } c\} \neq \{\}$  by auto**  
**from  $\text{closest-prestable-1}[OF \text{ this less.prem}(4) \text{ valid}] R \text{ closest-valid-1}[OF$**   
 **$\text{this less.prem}(4) \text{ valid}]$**   
**obtain  $I'' r''$**   
**where  $\text{stability}: \forall v \in ?R. \forall t > 0. \exists t' \leq t. (v \oplus t') \in \text{region } X \text{ I'' } r''$**   
 $\wedge t' \geq 0$   
**and  $\text{succ-not-reft}: \forall v \in \text{region } X \text{ I'' } r''. \forall t \geq 0. (v \oplus t) \notin ?R$**   
**and  $\text{no-consts}: \forall x \in X. \neg \text{isConst } (I'' x)$**   
**and  $\text{crit-mono}: \forall v \in ?R. \forall t < 1. \forall t' \geq 0. (v \oplus t') \in \text{region } X$**   
 $I'' r''$   
 $\longrightarrow \{x. x \in X \wedge (\exists c. I x = \text{Intv } c \wedge v x + t \geq c +$   
 $1)\}$   
 $= \{x. x \in X \wedge (\exists c. I'' x = \text{Intv } c \wedge (v \oplus t') x + (t$   
 $- t') \geq c + 1)\}$   
**and  $\text{succ-valid}: \text{valid-region } X \text{ k I'' } r''$**   
**by auto**  
**let  $?R'' = \text{region } X \text{ I'' } r''$**   
**from  $\text{stability } v \langle t > 0 \rangle$  obtain  $t1$  where  $t1: t1 \geq 0 \ t1 \leq t (v \oplus t1)$**   
 $\in ?R''$  **by auto**  
**from  $\text{stability } v' \langle t > 0 \rangle$  obtain  $t2$  where  $t2: t2 \geq 0 \ t2 \leq t (v' \oplus t2)$**   
 $\in ?R''$  **by auto**  
**let  $?v = v \oplus t1$**   
**let  $?t = t - t1$**   
**let  $?C' = \{x \in X. \exists c. I'' x = \text{Intv } c \wedge \text{real } (c + 1) \leq ?v x + ?t\}$**   
**from  $t1 \langle t < 1 \rangle$  have  $tt: 0 \leq ?t \ ?t < 1$  by auto**  
**from  $\text{crit-mono } \langle t < 1 \rangle \ t1(1,3) \langle v \in ?R \rangle$  have  $\text{crit}:$**   
 $\{x \in X. \exists c. I x = \text{Intv } c \wedge \text{real } (c + 1) \leq v x + t\}$   
 $= \{x \in X. \exists c. I'' x = \text{Intv } c \wedge \text{real } (c + 1) \leq (v \oplus t1) x + (t -$   
 $t1)\}$   
**by auto**  
**with  $C$  have  $C: ?C' = \{\}$  by blast**  
**from  $\text{critical-empty-intro}[OF \text{ no-consts } t1(3) \ tt(1) \ \text{this}]$  have  $((v \oplus t1)$   
 $\oplus ?t) \in ?R''$  .  
**from  $\text{region-unique}[OF \text{ less}(2) \ \text{this}] \ \text{less}(2) \ \text{succ-valid } t2$  have  $\exists t' \geq 0.$**   
 $(v' \oplus t') \in [v \oplus t]_{\mathcal{R}}$**

```

  by (auto simp: cval-add-def)
} note intro-const = this
{ fix v I r t x c v'
  let ?R = region X I r
  assume v: v ∈ ?R
  assume v': v' ∈ ?R
  assume F2: ∀ x ∈ X. ¬ isConst (I x)
  assume x: x ∈ X I x = Intv c v x + t ≥ c + 1
  assume valid: valid-region X k I r
  assume t: t ≥ 0 t < 1
  let ?C' = {x ∈ X. ∃ c. I x = Intv c ∧ real (c + 1) ≤ v x + t}
  assume C: ?C = ?C'
  have not-in-R: (v ⊕ t) ∉ ?R
  proof (rule ccontr, auto)
    assume (v ⊕ t) ∈ ?R
    with x(1,2) have v x + t < c + 1 by (fastforce simp: cval-add-def)
    with x(3) show False by simp
  qed
  have not-in-R': (v' ⊕ 1) ∉ ?R
  proof (rule ccontr, auto)
    assume (v' ⊕ 1) ∈ ?R
    with x have v' x + 1 < c + 1 by (fastforce simp: cval-add-def)
    moreover from x v' have c < v' x by fastforce
    ultimately show False by simp
  qed
  let ?X0 = {x ∈ X. isIntv (I x)}
  let ?M = {x ∈ ?X0. ∀ y ∈ ?X0. (x, y) ∈ r ⟶ (y, x) ∈ r}
  from x have x: x ∈ X ¬ isGreater (I x) and c: I x = Intv c by auto
  with ⟨x ∈ X⟩ have *: ?X0 ≠ {} by auto
  have ?X0 = {x ∈ X. ∃ d. I x = Intv d} by auto
  with valid have r: total-on ?X0 r trans r by auto
  from total-finite-trans-max[OF * - this] ⟨finite X⟩
  obtain x' where x': x' ∈ ?X0 ∀ y ∈ ?X0. x' ≠ y ⟶ (y, x') ∈ r by
fastforce
  from this(2) have ∀ y ∈ ?X0. (x', y) ∈ r ⟶ (y, x') ∈ r by auto
  with x'(1) have ?M ≠ {} by fastforce
  from closest-prestable-2[OF F2 less.prem(4) valid this] closest-valid-2[OF
F2 less.prem(4) valid this]
  obtain I' r'
  where stability:
    ∀ v ∈ region X I r. ∀ t ≥ 0. (v ⊕ t) ∉ region X I r ⟶ (∃ t' ≤ t. (v ⊕
t') ∈ region X I' r' ∧ t' ≥ 0)
  and critical-mono: ∀ v ∈ region X I r. ∀ t. ∀ t'.
    {x. x ∈ X ∧ (∃ c. I' x = Intv c ∧ (v ⊕ t') x + (t -

```

$t') \geq \text{real } (c + 1))\}$   
 $= \{x. x \in X \wedge (\exists c. I x = \text{Intv } c \wedge v x + t \geq \text{real}$   
 $(c + 1))\} - ?M$   
**and** *const-ex*:  $\exists x \in X. \text{isConst } (I' x)$   
**and** *succ-valid*: *valid-region*  $X$   $k$   $I' r'$   
**by** *auto*  
**let**  $?R' = \text{region } X$   $I' r'$   
**from** *not-in-R stability*  $\langle t \geq 0 \rangle$   $v$  **obtain**  $t'$  **where**  
 $t': t' \geq 0$   $t' \leq t$   $(v \oplus t') \in ?R'$   
**by** *blast*  
**have**  $(1::t) \geq 0$  **by** *auto*  
**with** *not-in-R' stability*  $v'$  **obtain**  $t1$  **where**  
 $t1: t1 \geq 0$   $t1 \leq 1$   $(v' \oplus t1) \in ?R'$   
**by** *blast*  
**let**  $?v = v \oplus t'$   
**let**  $?t = t - t'$   
**let**  $?C'' = \{x \in X. \exists c. I' x = \text{Intv } c \wedge \text{real } (c + 1) \leq ?v x + ?t\}$   
**have**  $\exists t' \geq 0. (v' \oplus t') \in [v \oplus t]_{\mathcal{R}}$   
**proof** (*cases*  $t = t'$ )  
**case** *True*  
**with**  $t'$  **have**  $(v \oplus t) \in ?R'$  **by** *auto*  
**from** *region-unique*[*OF less(2) this*] *succ-valid*  $\mathcal{R}$ -*def* **have**  $[v \oplus t]_{\mathcal{R}}$   
 $= ?R'$  **by** *blast*  
**with**  $t1(1,3)$  **show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**with**  $\langle t < 1 \rangle$   $t'$  **have**  $tt: 0 \leq ?t$   $?t < 1$   $?t > 0$  **by** *auto*  
**from** *critical-mono*  $\langle v \in ?R \rangle$  **have**  $C\text{-eq}: ?C'' = ?C' - ?M$  **by** *auto*  
**show**  $\exists t' \geq 0. (v' \oplus t') \in [v \oplus t]_{\mathcal{R}}$   
**proof** (*cases*  $?C' \cap ?M = \{\}$ )  
**case** *False*  
**from**  $\langle \text{finite } X \rangle$  **have** *finite*  $?C''$  *finite*  $?C'$  *finite*  $?M$  **by** *auto*  
**then** **have**  $\text{card } ?C'' < \text{card } ?C$  **using**  $C\text{-eq}$   $C$  *False* **by** (*intro*  
*card-mono-strict-subset*) *auto*  
**from** *less(1)*[*OF this less(2) succ-valid*  $t'(3)$   $t1(3)$   $\langle \text{finite } X \rangle$   $tt(1,2)$ ]  
**obtain**  $t2$  **where**  $t2 \geq 0$   $((v' \oplus t1) \oplus t2) \in [(v \oplus t)]_{\mathcal{R}}$  **by** (*auto*  
*simp: cval-add-def*)  
**moreover** **have**  $(v' \oplus (t1 + t2)) = ((v' \oplus t1) \oplus t2)$  **by** (*auto* *simp:*  
*cval-add-def*)  
**moreover** **have**  $t1 + t2 \geq 0$  **using**  $\langle t2 \geq 0 \rangle$   $t1(1)$  **by** *auto*  
**ultimately** **show** *?thesis* **by** *metis*  
**next**  
**case** *True*  
**{ fix**  $x$   $c$  **assume**  $x: x \in X$   $I x = \text{Intv } c$   $\text{real } (c + 1) \leq v x + t$

**with** *True* **have**  $x \notin ?M$  **by** *force*  
**from**  $x$  **have**  $x \in ?X_0$  **by** *auto*  
**from**  $x(1,2)$   $\langle v \in ?R \rangle$  **have**  $*$ :  $c < v x$   $v x < c + 1$  **by** *fastforce+*  
**with**  $\langle t < 1 \rangle$  **have**  $v x + t < c + 2$  **by** *auto*  
**have** *ge-1*:  $\text{frac}(v x) + t \geq 1$   
**proof** (*rule ccontr, goal-cases*)  
  **case** 1  
    **then** **have** *A*:  $\text{frac}(v x) + t < 1$  **by** *auto*  
    **from**  $*$  **have**  $\text{floor}(v x) + \text{frac}(v x) < c + 1$  **unfolding** *frac-def*  
**by** *auto*  
    **with** *nat-intv-frac-gt0[OF \*]* **have**  $\text{floor}(v x) \leq c$  **by** *linarith*  
    **with** *A* **have**  $v x + t < c + 1$  **by** (*auto simp: frac-def*)  
    **with**  $x(3)$  **show** *False* **by** *auto*  
  **qed**  
**from**  $\langle ?M \neq \{\} \rangle$  **obtain**  $y$  **where**  $y \in ?M$  **by** *force*  
**with**  $\langle x \in ?X_0 \rangle$  **have**  $y: y \in ?X_0$   $(y, x) \in r \longrightarrow (x, y) \in r$  **by** *auto*  
**from**  $y$  **obtain**  $c'$  **where**  $c': y \in X$   $I y = \text{Intv } c'$  **by** *auto*  
**with**  $\langle v \in ?R \rangle$  **have**  $c' < v y$  **by** *fastforce*  
**from**  $\langle y \in ?M \rangle$   $\langle x \notin ?M \rangle$  **have**  $x \neq y$  **by** *auto*  
  **with**  $y$   $r(1)$   $x(1,2)$  **have**  $(x, y) \in r$  **unfolding** *total-on-def* **by**  
*fastforce*  
  **with**  $\langle v \in ?R \rangle$   $c' x$  **have**  $\text{frac}(v x) \leq \text{frac}(v y)$  **by** *fastforce*  
  **with** *ge-1* **have** *frac*:  $\text{frac}(v y) + t \geq 1$  **by** *auto*  
  **have**  $\text{real}(c' + 1) \leq v y + t$   
  **proof** (*rule ccontr, goal-cases*)  
    **case** 1  
      **from**  $\langle c' < v y \rangle$  **have**  $\text{floor}(v y) \geq c'$  **by** *linarith*  
      **with** *frac* **have**  $v y + t \geq c' + 1$  **unfolding** *frac-def* **by** *linarith*  
      **with** 1 **show** *False* **by** *simp*  
  **qed**  
  **with**  $c'$  *True*  $\langle y \in ?M \rangle$  **have** *False* **by** *auto*  
}

**then** **have** *C*:  $?C' = \{\}$  **by** *auto*  
**with** *C-eq* **have**  $C'': ?C'' = \{\}$  **by** *auto*  
**from** *intro-const[OF t'(3) t1(3) succ-valid tt(3) tt(2) C'' const-ex]*  
**obtain**  $t2$  **where**  $t2 \geq 0$   $((v' \oplus t1) \oplus t2) \in [v \oplus t]_{\mathcal{R}}$  **by** (*auto simp:*  
*cval-add-def*)  
  **moreover** **have**  $(v' \oplus (t1 + t2)) = ((v' \oplus t1) \oplus t2)$  **by** (*auto simp:*  
*cval-add-def*)  
  **moreover** **have**  $t1 + t2 \geq 0$  **using**  $\langle t2 \geq 0 \rangle$   $t1(1)$  **by** *auto*  
  **ultimately** **show** *?thesis* **by** *metis*  
**qed**  
**qed**  
} **note** *intro-intv = this*

**from** *regions-closed*[*OF less*(2) *R less*(4,  $\gamma$ )] *less*(2) **obtain**  $I' r'$  **where**  
 $R'$ :  
 $[v \oplus t]_{\mathcal{R}} = \text{region } X I' r' \text{ valid-region } X k I' r'$   
**by** *auto*  
**with** *regions-closed'*[*OF less*(2) *R less*(4,  $\gamma$ )] *assms*(1) **have**  
 $R'2: (v \oplus t) \in [v \oplus t]_{\mathcal{R}} (v \oplus t) \in \text{region } X I' r'$   
**by** *auto*  
**let**  $?R' = \text{region } X I' r'$   
**from** *less*(2)  $R'$  **have**  $?R' \in \mathcal{R}$  **by** *auto*  
**show** *?case*  
**proof** (*cases*  $?R' = ?R$ )  
**case** *True* **with** *less*(3,5)  $R'(1)$  **have**  $(v' \oplus 0) \in [v \oplus t]_{\mathcal{R}}$  **by** (*auto simp: cval-add-def*)  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**have**  $t > 0$   
**proof** (*rule ccontr*)  
**assume**  $\neg 0 < t$   
**with**  $R' \langle t \geq 0 \rangle$  **have**  $[v]_{\mathcal{R}} = ?R'$  **by** (*simp add: cval-add-def*)  
**with** *region-unique*[*OF less*(2) *less*(4)  $R$ ]  $\langle ?R' \neq ?R \rangle$  **show** *False* **by**  
*auto*  
**qed**  
**show** *?thesis*  
**proof** (*cases*  $?C = \{\}$ )  
**case** *True*  
**show** *?thesis*  
**proof** (*cases*  $\exists x \in X. \text{isConst } (I x)$ )  
**case** *False*  
**then have** *no-consts:  $\forall x \in X. \neg \text{isConst } (I x)$*  **by** *auto*  
**from** *critical-empty-intro*[*OF this*  $\langle v \in ?R \rangle \langle t \geq 0 \rangle \text{True}$ ] **have**  $(v \oplus t) \in ?R$ .  
**from** *region-unique*[*OF less*(2) *this R*] *less*(5) **have**  $(v' \oplus 0) \in [v \oplus t]_{\mathcal{R}}$   
**by** (*auto simp: cval-add-def*)  
**then show** *?thesis* **by** *blast*  
**next**  
**case** *True*  
**from** *const-dest*[*OF this*  $\langle t > 0 \rangle$ ] **obtain**  $t1 t2 I' r'$   
**where**  $t1: t1 \geq 0 t1 \leq t (v \oplus t1) \in \text{region } X I' r'$   
**and**  $t2: t2 \geq 0 t2 \leq t (v' \oplus t2) \in \text{region } X I' r'$   
**and** *valid: valid-region*  $X k I' r'$   
**and** *no-consts:  $\forall x \in X. \neg \text{isConst } (I' x)$*   
**and**  $C: ?C = \{x \in X. \exists c. I' x = \text{Intv } c \wedge \text{real } (c + 1) \leq (v \oplus$

$t1) x + (t - t1)\}$   
 by *auto*  
 let  $?v = v \oplus t1$   
 let  $?t = t - t1$   
 let  $?C' = \{x \in X. \exists c. I' x = Intv\ c \wedge real\ (c + 1) \leq ?v\ x + ?t\}$   
 let  $?R' = region\ X\ I'\ r'$   
 from  $C \langle ?C = \{\}\rangle$  have  $?C' = \{\}$  by *blast*  
 from *critical-empty-intro*[*OF no-consts t1(3) - this*]  $t1$  have  $(?v \oplus ?t) \in ?R'$  by *auto*  
 from *region-unique*[*OF less(2) this*] *less(2) valid t2* show *?thesis*  
 by (*auto simp: cval-add-def*)  
 qed  
 next  
 case *False*  
 then obtain  $x\ c$  where  $x: x \in X\ I\ x = Intv\ c\ v\ x + t \geq c + 1$  by *auto*  
 then have  $F: \neg (\forall x \in X. \exists c. I\ x = Greater\ c)$  by *force*  
 show *?thesis*  
 proof (*cases*  $\exists x \in X. isConst\ (I\ x)$ )  
 case *False*  
 then have  $\forall x \in X. \neg isConst\ (I\ x)$  by *auto*  
 from *intro-intv*[*OF*  $\langle v \in ?R \rangle \langle v' \in ?R \rangle$  *this x less(3,7,8)*] show *?thesis* by *auto*  
 next  
 case *True*  
 then have  $\{x \in X. \exists c. I\ x = Const\ c\} \neq \{\}$  by *auto*  
 from *const-dest*[*OF True*  $\langle t > 0 \rangle$ ] obtain  $t1\ t2\ I'\ r'$   
 where  $t1: t1 \geq 0\ t1 \leq t\ (v \oplus t1) \in region\ X\ I'\ r'$   
 and  $t2: t2 \geq 0\ t2 \leq t\ (v' \oplus t2) \in region\ X\ I'\ r'$   
 and *valid: valid-region X k I' r'*  
 and *no-consts:  $\forall x \in X. \neg isConst\ (I'\ x)$*   
 and  $C: ?C = \{x \in X. \exists c. I' x = Intv\ c \wedge real\ (c + 1) \leq (v \oplus t1)\ x + (t - t1)\}$   
 by *auto*  
 let  $?v = v \oplus t1$   
 let  $?t = t - t1$   
 let  $?C' = \{x \in X. \exists c. I' x = Intv\ c \wedge real\ (c + 1) \leq ?v\ x + ?t\}$   
 let  $?R' = region\ X\ I'\ r'$   
 show *?thesis*  
 proof (*cases*  $?C' = \{\}$ )  
 case *False*  
 with *intro-intv*[*OF t1(3) t2(3) no-consts - - - valid - - C*]  $\langle t < 1 \rangle$   
 $t1$  obtain  $t'$  where  
 $t' \geq 0\ ((v' \oplus t2) \oplus t') \in [(v \oplus t)]_{\mathcal{R}}$

```

    by (auto simp: cval-add-def)
    moreover have  $((v' \oplus t2) \oplus t') = (v' \oplus (t2 + t'))$  by (auto simp:
cval-add-def)
    moreover have  $t2 + t' \geq 0$  using  $\langle t' \geq 0 \rangle \langle t2 \geq 0 \rangle$  by auto
    ultimately show ?thesis by metis
  next
    case True
    from critical-empty-intro[OF no-consts t1(3) - this] t1 have  $((v \oplus
t1) \oplus ?t) \in ?R'$  by auto
    from region-unique[OF less(2) this] less(2) valid t2 show ?thesis
    by (auto simp: cval-add-def)
  qed
qed
qed
qed
qed

```

Finally, we can put the two pieces together: for a non-negative shift  $t$ , we first shift  $\lfloor t \rfloor$  and then  $\text{frac } t$ .

**lemma** *set-of-regions*:

```

  fixes  $X k$ 
  defines  $\mathcal{R} \equiv \{ \text{region } X I r \mid I r. \text{ valid-region } X k I r \}$ 
  assumes  $R \in \mathcal{R} v \in R R' \in \text{Succ } \mathcal{R} R \text{ finite } X$ 
  shows  $\exists t \geq 0. [v \oplus t]_{\mathcal{R}} = R'$  using assms
proof -
  from assms(4) obtain  $v' t$  where  $v': v' \in R R' \in \mathcal{R} 0 \leq t R' = [v' \oplus
t]_{\mathcal{R}}$  by fastforce
  obtain  $t1 :: \text{int}$  where  $t1: t1 = \text{floor } t$  by auto
  with  $v'(3)$  have  $t1 \geq 0$  by auto
  from int-shift-equiv[OF  $v'(1) \langle v \in R \rangle$  assms(2)[unfolded  $\mathcal{R}$ -def] this]
 $\mathcal{R}$ -def
  have  $*$ :  $(v \oplus t1) \in [v' \oplus t1]_{\mathcal{R}}$  by auto
  let  $?v = (v \oplus t1)$ 
  let  $?t2 = \text{frac } t$ 
  have frac:  $0 \leq ?t2 ?t2 < 1$  by (auto simp: frac-lt-1)
  let  $?R = [v' \oplus t1]_{\mathcal{R}}$ 
  from regions-closed[OF - assms(2)  $v'(1)$ ]  $\langle t1 \geq 0 \rangle$   $\mathcal{R}$ -def have  $?R \in \mathcal{R}$ 
by auto
  with assms obtain  $I r$  where  $R: ?R = \text{region } X I r \text{ valid-region } X k I r$ 
by auto
  with  $*$  have  $v: ?v \in \text{region } X I r$  by auto
  from regions-closed'[OF - assms(2)  $v'(1)$ ]  $\langle t1 \geq 0 \rangle$   $\mathcal{R}$ -def have  $(v' \oplus
t1) \in \text{region } X I r$  by auto
  from set-of-regions-lt-1[OF R(2) this v assms(5) frac]  $\mathcal{R}$ -def obtain  $t2$ 

```

where

$t2 \geq 0$  ( $?v \oplus t2$ )  $\in [(v' \oplus t1) \oplus ?t2]_{\mathcal{R}}$   
 by *auto*  
 moreover from  $t1$  have  $(v \oplus (t1 + t2)) = (?v \oplus t2) v' \oplus t = ((v' \oplus t1) \oplus ?t2)$   
 by (*auto simp: frac-def cval-add-def*)  
 ultimately have  $(v \oplus (t1 + t2)) \in [v' \oplus t]_{\mathcal{R}}$   $t1 + t2 \geq 0$  using  $\langle t1 \geq 0 \rangle \langle t2 \geq 0 \rangle$  by *auto*  
 with *region-unique*[*OF - this(1)*]  $v'(2,4)$   $\mathcal{R}$ -def show *?thesis* by *blast qed*

## 5.4 Compability With Clock Constraints

**definition** *ccval* ( $\langle \{-\} \rangle$  [100]) where  $ccval\ cc \equiv \{v. v \vdash cc\}$

**definition** *acompatible*

where

*acompatible*  $\mathcal{R}\ ac \equiv \forall R \in \mathcal{R}. R \subseteq \{v. v \vdash_a ac\} \vee \{v. v \vdash_a ac\} \cap R = \{\}$

**lemma** *acompatibleD*:

assumes *acompatible*  $\mathcal{R}\ ac\ R \in \mathcal{R}\ u \in R\ v \in R\ u \vdash_a ac$

shows  $v \vdash_a ac$

using *assms unfolding compatible-def* by *auto*

**lemma** *ccompatible1*:

fixes  $X\ k$  fixes  $c :: real$

defines  $\mathcal{R} \equiv \{region\ X\ I\ r\ |I\ r. valid-region\ X\ k\ I\ r\}$

assumes  $c \leq k\ x\ c \in \mathbb{N}\ x \in X$

shows *acompatible*  $\mathcal{R}\ (EQ\ x\ c)$  using *assms unfolding compatible-def*

**proof** (*auto, goal-cases*)

case  $A: (1\ I\ r\ v\ u)$

from  $A(3,9)$  obtain  $d$  where  $d: c = of-nat\ d$  unfolding *Nats-def* by *auto*

with  $A(8,9)$  have  $u: u\ x = c\ u\ x = d$  unfolding *ccval-def* by *auto*

have  $I\ x = Const\ d$

**proof** (*cases\ I\ x, goal-cases*)

case  $(1\ c')$

with  $A$  have  $u\ x = c'$  by *fastforce*

with  $1\ u$  show *?case* by *auto*

next

case  $(2\ c')$

with  $A$  have  $c' < u\ x\ u\ x < c' + 1$  by *fastforce+*

with  $2\ u$  show *?case* by *auto*

next

**case**  $(\exists c')$   
**with**  $A$  **have**  $c' < u\ x$  **by** *fastforce*  
**moreover from**  $\exists A(4,5)$  **have**  $c' \geq k\ x$  **by** *fastforce*  
**ultimately show** *?case* **using**  $u\ A(2)$  **by** *auto*  
**qed**  
**with**  $A(4,6)$   $d$  **have**  $v\ x = c$  **by** *fastforce*  
**with**  $A(3,5)$  **have**  $v \vdash_a EQ\ x\ c$  **by** *auto*  
**with**  $A$  **show** *False* **unfolding** *ccval-def* **by** *auto*  
**qed**

**lemma** *ccompatible2*:

**fixes**  $X\ k$  **fixes**  $c :: real$   
**defines**  $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$   
**assumes**  $c \leq k\ x\ c \in \mathbb{N}\ x \in X$   
**shows** *acompatible*  $\mathcal{R}$   $(LT\ x\ c)$  **using** *assms* **unfolding** *acompatible-def*  
**proof** (*auto, goal-cases*)  
**case**  $A: (1\ I\ r\ v\ u)$   
**from**  $A(3)$  **obtain**  $d :: nat$  **where**  $d: c = of-nat\ d$  **unfolding** *Nats-def*  
**by** *blast*  
**with**  $A$  **have**  $u: u\ x < c\ u\ x < d$  **unfolding** *ccval-def* **by** *auto*  
**have**  $v\ x < c$   
**proof** (*cases\ I\ x, goal-cases*)  
**case**  $(1\ c')$   
**with**  $A$  **have**  $u\ x = c'\ v\ x = c'$  **by** *fastforce+*  
**with**  $u$  **show**  $v\ x < c$  **by** *auto*  
**next**  
**case**  $(2\ c')$   
**with**  $A$  **have**  $B: c' < u\ x\ u\ x < c' + 1\ c' < v\ x\ v\ x < c' + 1$  **by**  
*fastforce+*  
**with**  $u\ A(3)$  **have**  $c' + 1 \leq d$  **by** *auto*  
**with**  $d$  **have**  $c' + 1 \leq c$  **by** *auto*  
**with**  $B\ u$  **show**  $v\ x < c$  **by** *auto*  
**next**  
**case**  $(3\ c')$   
**with**  $A$  **have**  $c' < u\ x$  **by** *fastforce*  
**moreover from**  $\exists A(4,5)$  **have**  $c' \geq k\ x$  **by** *fastforce*  
**ultimately show** *?case* **using**  $u\ A(2)$  **by** *auto*  
**qed**  
**with**  $A(4,6)$  **have**  $v \vdash_a LT\ x\ c$  **by** *auto*  
**with**  $A(7)$  **show** *False* **unfolding** *ccval-def* **by** *auto*  
**qed**

**lemma** *ccompatible3*:

**fixes**  $X\ k$  **fixes**  $c :: real$

**defines**  $\mathcal{R} \equiv \{\text{region } X \text{ I } r \mid \text{I } r. \text{ valid-region } X \text{ k I } r\}$   
**assumes**  $c \leq k \ x \ c \in \mathbb{N} \ x \in X$   
**shows** *acompatible*  $\mathcal{R}$  ( $LE \ x \ c$ ) **using** *assms* **unfolding** *acompatible-def*  
**proof** (*auto, goal-cases*)  
**case**  $A: (1 \text{ I } r \ v \ u)$   
**from**  $A(3)$  **obtain**  $d :: \text{nat}$  **where**  $d: c = \text{of-nat } d$  **unfolding** *Nats-def*  
**by** *blast*  
**with**  $A$  **have**  $u: u \ x \leq c \ u \ x \leq d$  **unfolding** *ccval-def* **by** *auto*  
**have**  $v \ x \leq c$   
**proof** (*cases I x, goal-cases*)  
**case**  $(1 \ c')$  **with**  $A \ u$  **show** *?case* **by** *fastforce*  
**next**  
**case**  $(2 \ c')$   
**with**  $A$  **have**  $B: c' < u \ x \ u \ x < c' + 1 \ c' < v \ x \ v \ x < c' + 1$  **by**  
*fastforce+*  
**with**  $u \ A(3)$  **have**  $c' + 1 \leq d$  **by** *auto*  
**with**  $d \ u \ A(3)$  **have**  $c' + 1 \leq c$  **by** *auto*  
**with**  $B \ u$  **show**  $v \ x \leq c$  **by** *auto*  
**next**  
**case**  $(3 \ c')$   
**with**  $A$  **have**  $c' < u \ x$  **by** *fastforce*  
**moreover** **from**  $3 \ A(4,5)$  **have**  $c' \geq k \ x$  **by** *fastforce*  
**ultimately** **show** *?case* **using**  $u \ A(2)$  **by** *auto*  
**qed**  
**with**  $A(4,6)$  **have**  $v \vdash_a \ LE \ x \ c$  **by** *auto*  
**with**  $A(7)$  **show** *False* **unfolding** *ccval-def* **by** *auto*  
**qed**

**lemma** *ccompatible4*:

**fixes**  $X \ k$  **fixes**  $c :: \text{real}$   
**defines**  $\mathcal{R} \equiv \{\text{region } X \text{ I } r \mid \text{I } r. \text{ valid-region } X \text{ k I } r\}$   
**assumes**  $c \leq k \ x \ c \in \mathbb{N} \ x \in X$   
**shows** *acompatible*  $\mathcal{R}$  ( $GT \ x \ c$ ) **using** *assms* **unfolding** *acompatible-def*  
**proof** (*auto, goal-cases*)  
**case**  $A: (1 \text{ I } r \ v \ u)$   
**from**  $A(3)$  **obtain**  $d :: \text{nat}$  **where**  $d: c = \text{of-nat } d$  **unfolding** *Nats-def*  
**by** *blast*  
**with**  $A$  **have**  $u: u \ x > c \ u \ x > d$  **unfolding** *ccval-def* **by** *auto*  
**have**  $v \ x > c$   
**proof** (*cases I x, goal-cases*)  
**case**  $(1 \ c')$  **with**  $A \ u$  **show** *?case* **by** *fastforce*  
**next**  
**case**  $(2 \ c')$   
**with**  $A$  **have**  $B: c' < u \ x \ u \ x < c' + 1 \ c' < v \ x \ v \ x < c' + 1$  **by**

*fastforce+*  
**with**  $d\ u$  **have**  $c' \geq c$  **by** *auto*  
**with**  $B\ u$  **show**  $v\ x > c$  **by** *auto*  
**next**  
**case**  $(\exists\ c')$   
**with**  $A(4,6)$  **have**  $c' < v\ x$  **by** *fastforce*  
**moreover from**  $\exists\ A(4,5)$  **have**  $c' \geq k\ x$  **by** *fastforce*  
**ultimately show** *?case* **using**  $A(2)\ u(1)$  **by** *auto*  
**qed**  
**with**  $A(4,6)$  **have**  $v \vdash_a\ GT\ x\ c$  **by** *auto*  
**with**  $A(7)$  **show** *False* **unfolding** *ccval-def* **by** *auto*  
**qed**

**lemma** *ccompatible5*:  
**fixes**  $X\ k$  **fixes**  $c :: real$   
**defines**  $\mathcal{R} \equiv \{region\ X\ I\ r\ |I\ r.\ valid-region\ X\ k\ I\ r\}$   
**assumes**  $c \leq k\ x\ c \in \mathbb{N}\ x \in X$   
**shows** *acompatible*  $\mathcal{R}\ (GE\ x\ c)$  **using** *assms* **unfolding** *acompatible-def*  
**proof** (*auto, goal-cases*)  
**case**  $A: (1\ I\ r\ v\ u)$   
**from**  $A(3)$  **obtain**  $d :: nat$  **where**  $d: c = of-nat\ d$  **unfolding** *Nats-def*  
**by** *blast*  
**with**  $A$  **have**  $u\ x \geq c\ u\ x \geq d$  **unfolding** *ccval-def* **by** *auto*  
**have**  $v\ x \geq c$   
**proof** (*cases\ I\ x, goal-cases*)  
**case**  $(1\ c')$  **with**  $A\ u$  **show** *?case* **by** *fastforce*  
**next**  
**case**  $(2\ c')$   
**with**  $A$  **have**  $B: c' < u\ x\ u\ x < c' + 1\ c' < v\ x\ v\ x < c' + 1$  **by**  
*fastforce+*  
**with**  $d\ u$  **have**  $c' \geq c$  **by** *auto*  
**with**  $B\ u$  **show**  $v\ x \geq c$  **by** *auto*  
**next**  
**case**  $(\exists\ c')$   
**with**  $A(4,6)$  **have**  $c' < v\ x$  **by** *fastforce*  
**moreover from**  $\exists\ A(4,5)$  **have**  $c' \geq k\ x$  **by** *fastforce*  
**ultimately show** *?case* **using**  $A(2)\ u(1)$  **by** *auto*  
**qed**  
**with**  $A(4,6)$  **have**  $v \vdash_a\ GE\ x\ c$  **by** *auto*  
**with**  $A(7)$  **show** *False* **unfolding** *ccval-def* **by** *auto*  
**qed**

**lemma** *acompatible*:  
**fixes**  $X\ k$  **fixes**  $c :: real$

**defines**  $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$   
**assumes**  $c \leq k \ x \ c \in \mathbb{N} \ x \in X \ \text{constraint-pair } ac = (x, c)$   
**shows** *acompatible*  $\mathcal{R} \ ac$  **using** *assms*  
**by** (*cases* *ac*) (*auto intro: ccompatible1 ccompatible2 ccompatible3 ccompatible4 ccompatible5*)

**definition** *ccompatible*

**where**

*ccompatible*  $\mathcal{R} \ cc \equiv \forall R \in \mathcal{R}. R \subseteq \{\{cc\} \vee \{cc\} \cap R = \{\}$

**lemma** *ccompatible*:

**fixes**  $X \ k$  **fixes**  $c :: \text{nat}$

**defines**  $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$

**assumes**  $\forall (x, m) \in \text{collect-clock-pairs } cc. m \leq k \ x \wedge x \in X \wedge m \in \mathbb{N}$

**shows** *ccompatible*  $\mathcal{R} \ cc$  **using** *assms*

**proof** (*induction* *cc*)

**case** *Nil*

**then show** *?case* **by** (*auto simp: ccompatible-def ccval-def clock-val-def*)

**next**

**case** (*Cons* *ac* *cc*)

**then have** *ccompatible*  $\mathcal{R} \ cc$  **by** (*auto simp: collect-clock-pairs-def*)

**moreover have**

*acompatible*  $\mathcal{R} \ ac$

**using** *Cons.prem*s **by** (*auto intro: acompatible simp: collect-clock-pairs-def*  
*\mathcal{R}*-*def*)

**ultimately show** *?case*

**unfolding** *ccompatible-def* *acompatible-def* *ccval-def* **by** (*fastforce simp: clock-val-def*)

**qed**

## 5.5 Compability with Resets

**definition** *region-set*

**where**

*region-set*  $R \ x \ c = \{v(x := c) \mid v. v \in R\}$

**lemma** *region-set-id*:

**fixes**  $X \ k$

**defines**  $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$

**assumes**  $R \in \mathcal{R} \ v \in R \ \text{finite } X \ 0 \leq c \ c \leq k \ x \ x \in X$

**shows**  $[v(x := c)]_{\mathcal{R}} = \text{region-set } R \ x \ c \ [v(x := c)]_{\mathcal{R}} \in \mathcal{R} \ v(x := c) \in [v(x := c)]_{\mathcal{R}}$

**proof** –

**from** *assms* **obtain**  $I r$  **where**  $R: R = \text{region } X I r \ \text{valid-region } X k I r$

```

v ∈ region X I r by auto
  let ?I = λ y. if x = y then Const c else I y
  let ?r = {(y,z) ∈ r. x ≠ y ∧ x ≠ z}
  let ?X0 = {x ∈ X. ∃ c. I x = Intv c}
  let ?X'0 = {x ∈ X. ∃ c. ?I x = Intv c}

  have r-subset: r ⊆ ?X0 × ?X0 and refl: refl-on ?X0 r and trans: trans
r and
    total: total-on ?X0 r
    using R(2) by auto

  have valid: valid-region X k ?I ?r
proof
  show ?X0 - {x} = ?X'0 by auto
next
  show ?r ⊆ (?X0 - {x}) × (?X0 - {x})
    using r-subset by blast
next
  from refl show refl-on (?X0 - {x}) ?r unfolding refl-on-def by auto
next
  from trans show trans ?r unfolding trans-def by blast
next
  from total show total-on (?X0 - {x}) ?r unfolding total-on-def by
auto
next
  from R(2) have ∀ x ∈ X. valid-intv (k x) (I x) by auto
  with ⟨c ≤ k x⟩ show ∀ x ∈ X. valid-intv (k x) (?I x) by auto
qed

{ fix v assume v: v ∈ region-set R x c
  with R(1) obtain v' where v': v' ∈ region X I r v = v'(x := c)
unfolding region-set-def by auto
  have v ∈ region X ?I ?r
proof (standard, goal-cases)
  case 1
  from v' ⟨0 ≤ c⟩ show ?case by auto
next
  case 2
  from v' show ?case
proof (auto, goal-cases)
  case (1 y)
  then have intv-elem y v' (I y) by auto
  with ⟨x ≠ y⟩ show intv-elem y (v'(x := c)) (I y) by (cases I y) auto
qed

```

```

next
  show  $?X_0 - \{x\} = ?X_0'$  by auto
next
  from  $v'$  show  $\forall y \in ?X_0 - \{x\}. \forall z \in ?X_0 - \{x\}. (y,z) \in ?r \longleftrightarrow$ 
frac (v y)  $\leq$  frac (v z) by auto
qed
} moreover
{ fix v assume v: v  $\in$  region X ?I ?r
  have  $\exists c. v(x := c) \in$  region X I r
  proof (cases I x)
    case (Const c)
      from R(2) have  $c \geq 0$  by auto
      let  $?v = v(x := c)$ 
      have  $?v \in$  region X I r
      proof (standard, goal-cases)
        case 1
          from  $\langle c \geq 0 \rangle v$  show ?case by auto
        next
          case 2
            show ?case
            proof (auto, goal-cases)
              case (1 y)
                with v have intv-elem y v (?I y) by fast
                with Const show intv-elem y ?v (I y) by (cases x = y, auto) (cases
I y, auto)
            qed
          next
            from Const show  $?X_0' = ?X_0$  by auto
            with r-subset have  $r \subseteq ?X_0' \times ?X_0'$  by auto
            then have r:  $?r = r$  by auto
            from v have  $\forall y \in ?X_0'. \forall z \in ?X_0'. (y,z) \in ?r \longleftrightarrow$  frac (v y)  $\leq$ 
frac (v z) by fastforce
            with r show  $\forall y \in ?X_0'. \forall z \in ?X_0'. (y,z) \in r \longleftrightarrow$  frac (?v y)  $\leq$ 
frac (?v z)
            by auto
          qed
        then show ?thesis by auto
      next
        case (Greater c)
          from R(2) have  $c \geq 0$  by auto
          let  $?v = v(x := c + 1)$ 
          have  $?v \in$  region X I r
          proof (standard, goal-cases)
            case 1

```

```

    from ⟨ $c \geq 0$ ⟩  $v$  show ?case by auto
  next
    case 2
    show ?case
    proof (standard, goal-cases)
      case (1  $y$ )
      with  $v$  have intv-elem  $y$   $v$  (? $I$   $y$ ) by fast
      with Greater show intv-elem  $y$  ? $v$  ( $I$   $y$ ) by (cases  $x = y$ , auto)
(cases  $I$   $y$ , auto)
    qed
  next
    from Greater show ? $X_0'$  = ? $X_0$  by auto
    with  $r$ -subset have  $r \subseteq ?X_0' \times ?X_0'$  by auto
    then have  $r$ : ? $r$  =  $r$  by auto
    from  $v$  have  $\forall y \in ?X_0'. \forall z \in ?X_0'. (y, z) \in ?r \iff \text{frac } (v y) \leq$ 
frac ( $v z$ ) by fastforce
    with  $r$  show  $\forall y \in ?X_0'. \forall z \in ?X_0'. (y, z) \in r \iff \text{frac } (?v y) \leq$ 
frac (? $v z$ )
    by auto
    qed
    then show ?thesis by auto
  next
    case (Intv  $c$ )
    from  $R(2)$  have  $c \geq 0$  by auto
    let ? $L$  = {frac ( $v y$ ) |  $y. y \in ?X_0 \wedge x \neq y \wedge (y, x) \in r$ }
    let ? $U$  = {frac ( $v y$ ) |  $y. y \in ?X_0 \wedge x \neq y \wedge (x, y) \in r$ }
    let ? $l$  = if ? $L$  ≠ {} then  $c + \text{Max } ?L$  else if ? $U$  ≠ {} then  $c$  else  $c +$ 
0.5
    let ? $u$  = if ? $U$  ≠ {} then  $c + \text{Min } ?U$  else if ? $L$  ≠ {} then  $c + 1$  else
 $c + 0.5$ 
    from ⟨finite  $X$ ⟩ have fin: finite ? $L$  finite ? $U$  by auto
    { fix  $y$  assume  $y: y \in ?X_0 \wedge x \neq y \wedge (y, x) \in r$ 
      then have  $L$ : frac ( $v y$ ) ∈ ? $L$  by auto
      with Max-in[OF fin(1)] have In:  $\text{Max } ?L \in ?L$  by auto
      then have frac ( $\text{Max } ?L$ ) = ( $\text{Max } ?L$ ) using frac-frac by fastforce
      from Max-ge[OF fin(1)  $L$ ] have frac ( $v y$ ) ≤  $\text{Max } ?L$  .
      also have ... = frac ( $\text{Max } ?L$ ) using In frac-frac[symmetric] by
fastforce
      also have ... = frac ( $c + \text{Max } ?L$ ) by (auto simp: frac-nat-add-id)
      finally have frac ( $v y$ ) ≤ frac ? $l$  using  $L$  by auto
    } note  $L$ -bound = this
    { fix  $y$  assume  $y: y \in ?X_0 \wedge x \neq y \wedge (x, y) \in r$ 
      then have  $U$ : frac ( $v y$ ) ∈ ? $U$  by auto
      with Min-in[OF fin(2)] have In:  $\text{Min } ?U \in ?U$  by auto

```

```

    then have frac (Min ?U) = (Min ?U) using frac-frac by fastforce
  have frac (c + Min ?U) = frac (Min ?U) by (auto simp: frac-nat-add-id)
  also have ... = Min ?U using In frac-frac by fastforce
  also from Min-le[OF fin(2) U] have Min ?U ≤ frac (v y) .
  finally have frac ?u ≤ frac (v y) using U by auto
} note U-bound = this
{ assume ?L ≠ {}
  from Max-in[OF fin(1) this] obtain l d where l:
    Max ?L = frac (v l) l ∈ X x ≠ l I l = Intv d
  by auto
  with v have d < v l v l < d + 1 by fastforce+
  with nat-intv-frac-gt0[OF this] frac-lt-1 l(1) have 0 < Max ?L Max
?L < 1 by auto
  then have c < c + Max ?L c + Max ?L < c + 1 by simp+
} note L-intv = this
{ assume ?U ≠ {}
  from Min-in[OF fin(2) this] obtain u d where u:
    Min ?U = frac (v u) u ∈ X x ≠ u I u = Intv d
  by auto
  with v have d < v u v u < d + 1 by fastforce+
  with nat-intv-frac-gt0[OF this] frac-lt-1 u(1) have 0 < Min ?U Min
?U < 1 by auto
  then have c < c + Min ?U c + Min ?U < c + 1 by simp+
} note U-intv = this
have l-bound: c ≤ ?l
proof (cases ?L = {})
  case True
  note T = this
  show ?thesis
  proof (cases ?U = {})
    case True
    with T show ?thesis by simp
  next
  case False
  with U-intv T show ?thesis by simp
qed
next
  case False
  with L-intv show ?thesis by simp
qed
have l-bound': c < ?u
proof (cases ?L = {})
  case True
  note T = this

```

```

show ?thesis
proof (cases ?U = {})
  case True
    with T show ?thesis by simp
  next
    case False
      with U-intv T show ?thesis by simp
    qed
next
  case False
    with U-intv show ?thesis by simp
  qed
have u-bound: ?u ≤ c + 1
proof (cases ?U = {})
  case True
    note T = this
    show ?thesis
    proof (cases ?L = {})
      case True
        with T show ?thesis by simp
      next
        case False
          with L-intv T show ?thesis by simp
        qed
    next
      case False
        with U-intv show ?thesis by simp
      qed
have u-bound': ?l < c + 1
proof (cases ?U = {})
  case True
    note T = this
    show ?thesis
    proof (cases ?L = {})
      case True
        with T show ?thesis by simp
      next
        case False
          with L-intv T show ?thesis by simp
        qed
    next
      case False
        with L-intv show ?thesis by simp
      qed

```

```

have frac-c: frac c = 0 frac (c+1) = 0 by auto
have l-u: ?l ≤ ?u
proof (cases ?L = {})
  case True
  note T = this
  show ?thesis
  proof (cases ?U = {})
    case True
    with T show ?thesis by simp
  next
  case False
  with T show ?thesis using Min-in[OF fin(2) False] by (auto
simp: frac-c)
  qed
next
  case False
  with Max-in[OF fin(1) this] have l: ?l = c + Max ?L Max ?L ∈ ?L
by auto
  note F = False
  from l(1) have *: Max ?L < 1 using False L-intv(2) by linarith
  show ?thesis
  proof (cases ?U = {})
    case True
    with F l * show ?thesis by simp
  next
  case False
  from Min-in[OF fin(2) this] l(2) obtain l u where l-u:
    Max ?L = frac (v l) Min ?U = frac (v u) l ∈ ?X0 u ∈ ?X0 (l,x)
  ∈ r (x,u) ∈ r
    x ≠ l x ≠ u
  by auto
  from trans l-u(5-) have (l,u) ∈ ?r unfolding trans-def by blast
  with l-u(1-4) v have *: Max ?L ≤ Min ?U by fastforce
  with l-u(1,2) have frac (Max ?L) ≤ frac (Min ?U) by (simp add:
frac-frac)
  with frac-nat-add-id l(1) False have frac ?l ≤ frac ?u by simp
  with l(1) * False show ?thesis by simp
  qed
qed
obtain d where d: d = (?l + ?u) / 2 by blast
with l-u have d2: ?l ≤ d d ≤ ?u by simp+
from d l-bound l-bound' u-bound u-bound' have d3: c < d d < c + 1
d ≥ 0 by simp+
have floor ?l = c

```

```

proof (cases ?L = {})
  case False
    from L-intv[OF False] have  $0 \leq \text{Max } ?L \text{ Max } ?L < 1$  by auto
    from floor-nat-add-id[OF this] False show ?thesis by simp
  next
    case True
      note T = this
      show ?thesis
      proof (cases ?U = {})
        case True
          with T show ?thesis by (simp add: floor-nat-add-id)
        next
          case False
            from U-intv[OF False] have  $0 \leq \text{Min } ?U \text{ Min } ?U < 1$  by auto
            from floor-nat-add-id[OF this] T False show ?thesis by simp
      qed
    qed
  have floor-u: floor ?u = (if ?U = {}  $\wedge$  ?L  $\neq$  {} then c + 1 else c)
  proof (cases ?U = {})
    case False
      from U-intv[OF False] have  $0 \leq \text{Min } ?U \text{ Min } ?U < 1$  by auto
      from floor-nat-add-id[OF this] False show ?thesis by simp
    next
      case True
        note T = this
        show ?thesis
        proof (cases ?L = {})
          case True
            with T show ?thesis by (simp add: floor-nat-add-id)
          next
            case False
              from L-intv[OF False] have  $0 \leq \text{Max } ?L \text{ Max } ?L < 1$  by auto
              from floor-nat-add-id[OF this] T False show ?thesis by auto
        qed
      qed
  { assume ?L  $\neq$  {} ?U  $\neq$  {}
    from Max-in[OF fin(1)  $\langle$ ?L  $\neq$  {} $\rangle$ ] obtain w where w:
      w  $\in$  ?X0 x  $\neq$  w (w,x)  $\in$  r Max ?L = frac (v w)
    by auto
    from Min-in[OF fin(2)  $\langle$ ?U  $\neq$  {} $\rangle$ ] obtain z where z:
      z  $\in$  ?X0 x  $\neq$  z (x,z)  $\in$  r Min ?U = frac (v z)
    by auto
    from w z trans have (w,z)  $\in$  r unfolding trans-def by blast
    with v w z have Max ?L  $\leq$  Min ?U by fastforce
  }

```

```

} note l-le-u = this
{ fix y assume y: y ∈ ?X₀ x ≠ y
  from total y ⟨x ∈ X⟩ Intv have total: (x,y) ∈ r ∨ (y,x) ∈ r unfolding
total-on-def by auto
  have frac (v y) = frac d ⟷ (y,x) ∈ r ∧ (x,y) ∈ r
  proof safe
    assume A: (y,x) ∈ r (x,y) ∈ r
    from L-bound[OF y A(1)] U-bound[OF y A(2)] have *:
      frac (v y) ≤ frac ?l frac ?u ≤ frac (v y)
    by auto
    from A y have **: ?L ≠ {} ?U ≠ {} by auto
    with L-intv[OF this(1)] U-intv[OF this(2)] have frac ?l = Max ?L
frac ?u = Min ?U
    by (auto simp: frac-nat-add-id frac-eq)
    with * ** l-le-u have frac ?l = frac ?u frac (v y) = frac ?l by auto
    with d have d = ((floor ?l + floor ?u) + (frac (v y) + frac (v y)))
/ 2
    unfolding frac-def by auto
    also have ... = c + frac (v y) using ⟨floor ?l = c⟩ floor-u ⟨?U ≠
{}⟩ by auto
    finally show frac (v y) = frac d using frac-nat-add-id frac-frac by
metis
  next
  assume A: frac (v y) = frac d
  show (y, x) ∈ r
  proof (rule ccontr)
    assume B: (y,x) ∉ r
    with total have B': (x,y) ∈ r by auto
    from U-bound[OF y this] have u-y: frac ?u ≤ frac (v y) by auto
    from y B' have U: ?U ≠ {} and frac (v y) ∈ ?U by auto
    then have u: frac ?u = Min ?U using Min-in[OF fin(2) ⟨?U ≠
{}⟩]
    by (auto simp: frac-nat-add-id frac-frac)
    show False
    proof (cases ?L = {})
      case True
      from U-intv[OF U] have 0 < Min ?U Min ?U < 1 by auto
      then have *: frac (Min ?U / 2) = Min ?U / 2 unfolding
frac-eq by simp
      from d U True have d = ((c + c) + Min ?U) / 2 by auto
      also have ... = c + Min ?U / 2 by simp
      finally have frac d = Min ?U / 2 using * by (simp add:
frac-nat-add-id)
      also have ... < Min ?U using ⟨0 < Min ?U⟩ by auto

```

```

    finally have  $\text{frac } d < \text{frac } ?u$  using  $u$  by auto
    with  $u-y A$  show False by auto
  next
    case False
    then have  $l: ?l = c + \text{Max } ?L$  by simp
    from  $\text{Max-in}[OF \text{fin}(1) \langle ?L \neq \{\} \rangle]$ 
    obtain  $w$  where  $w$ :
       $w \in ?X_0 \ x \neq w \ (w,x) \in r \ \text{Max } ?L = \text{frac } (v \ w)$ 
    by auto
    with  $\langle (y,x) \notin r \rangle$  trans have **:  $(y,w) \notin r$  unfolding trans-def
  by blast
    from  $\text{Min-in}[OF \text{fin}(2) \langle ?U \neq \{\} \rangle] \ \text{Max-in}[OF \text{fin}(1) \langle ?L \neq \{\} \rangle]$  frac-lt-1
    have  $0 \leq \text{Max } ?L \ \text{Max } ?L < 1 \ 0 \leq \text{Min } ?U \ \text{Min } ?U < 1$  by
  auto
    then have  $0 \leq (\text{Max } ?L + \text{Min } ?U) / 2 \ (\text{Max } ?L + \text{Min } ?U) / 2 < 1$  by auto
    then have **:  $\text{frac } ((\text{Max } ?L + \text{Min } ?U) / 2) = (\text{Max } ?L + \text{Min } ?U) / 2$  unfolding frac-eq ..
    from  $y \ w$  have  $y \in ?X_0' \ w \in ?X_0'$  by auto
    with  $v$  ** have lt:  $\text{frac } (v \ y) > \text{frac } (v \ w)$  by fastforce
    from  $d \ U \ l$  have  $d = ((c + c) + (\text{Max } ?L + \text{Min } ?U)) / 2$  by
  auto
    also have  $\dots = c + (\text{Max } ?L + \text{Min } ?U) / 2$  by simp
    finally have  $\text{frac } d = \text{frac } ((\text{Max } ?L + \text{Min } ?U) / 2)$  by (simp
  add: frac-nat-add-id)
    also have  $\dots = (\text{Max } ?L + \text{Min } ?U) / 2$  using ** by simp
    also have  $\dots < (\text{frac } (v \ y) + \text{Min } ?U) / 2$  using lt w(4) by
  auto
    also have  $\dots \leq \text{frac } (v \ y)$  using  $\text{Min-le}[OF \text{fin}(2) \langle \text{frac } (v \ y) \in ?U \rangle]$  by auto
    finally show False using  $A$  by auto
  qed
  qed
  next
    assume  $A: \text{frac } (v \ y) = \text{frac } d$ 
    show  $(x, y) \in r$ 
    proof (rule ccontr)
      assume  $B: (x,y) \notin r$ 
      with total have  $B': (y,x) \in r$  by auto
      from  $L\text{-bound}[OF \text{fin}(1) \text{ this}]$  have l-y:  $\text{frac } ?l \geq \text{frac } (v \ y)$  by auto
      from  $y \ B'$  have  $L: ?L \neq \{\}$  and  $\text{frac } (v \ y) \in ?L$  by auto
      then have  $l: \text{frac } ?l = \text{Max } ?L$  using  $\text{Max-in}[OF \text{fin}(1) \langle ?L \neq \{\} \rangle]$ 

```

```

by (auto simp: frac-nat-add-id frac-frac)
show False
proof (cases ?U = {})
  case True
  from L-intv[OF L] have *: 0 < Max ?L Max ?L < 1 by auto
  from d L True have d = ((c + c) + (1 + Max ?L)) / 2 by
auto
  also have ... = c + (1 + Max ?L) / 2 by simp
  finally have frac d = frac ((1 + Max ?L) / 2) by (simp add:
frac-nat-add-id)
  also have ... = (1 + Max ?L) / 2 using * unfolding frac-eq
by auto
  also have ... > Max ?L using * by auto
  finally have frac d > frac ?l using l by auto
  with l-y A show False by auto
next
case False
then have u: ?u = c + Min ?U by simp
from Min-in[OF fin(2) ‹?U ≠ {}›]
obtain w where w:
  w ∈ ?X0 x ≠ w (x,w) ∈ r Min ?U = frac (v w)
by auto
with ‹(x,y) ∉ r› trans have **: (w,y) ∉ r unfolding trans-def
by blast
  from Min-in[OF fin(2) ‹?U ≠ {}›] Max-in[OF fin(1) ‹?L ≠
{}›] frac-lt-1
  have 0 ≤ Max ?L Max ?L < 1 0 ≤ Min ?U Min ?U < 1 by
auto
  then have 0 ≤ (Max ?L + Min ?U) / 2 (Max ?L + Min ?U)
/ 2 < 1 by auto
  then have **: frac ((Max ?L + Min ?U) / 2) = (Max ?L +
Min ?U) / 2 unfolding frac-eq ..
  from y w have y ∈ ?X0' w ∈ ?X0' by auto
  with v ** have lt: frac (v y) < frac (v w) by fastforce
  from d L u have d = ((c + c) + (Max ?L + Min ?U))/2 by
auto
  also have ... = c + (Max ?L + Min ?U) / 2 by simp
  finally have frac d = frac ((Max ?L + Min ?U) / 2) by (simp
add: frac-nat-add-id)
  also have ... = (Max ?L + Min ?U) / 2 using ** by simp
  also have ... > (Max ?L + frac (v y)) / 2 using lt w(4) by
auto
  finally have frac d > frac (v y) using Max-ge[OF fin(1) ‹frac
(v y) ∈ ?L›] by auto

```

```

      then show False using A by auto
    qed
  qed
  qed
} note d-frac-equiv = this
have frac-l: frac ?l ≤ frac d
proof (cases ?L = {})
  case True
  note T = this
  show ?thesis
  proof (cases ?U = {})
    case True
    with T have ?l = ?u by auto
    with d have d = ?l by auto
    then show ?thesis by auto
  next
  case False
  with T have frac ?l = 0 by auto
  moreover have frac d ≥ 0 by auto
  ultimately show ?thesis by linarith
  qed
next
case False
note F = this
then have l: ?l = c + Max ?L frac ?l = Max ?L using Max-in[OF
fin(1) <?L ≠ {}>]
by (auto simp: frac-nat-add-id frac-frac)
from L-intv[OF F] have *: 0 < Max ?L Max ?L < 1 by auto
show ?thesis
proof (cases ?U = {})
  case True
  from True F have ?u = c + 1 by auto
  with l d have d = ((c + c) + (Max ?L + 1)) / 2 by auto
  also have ... = c + (1 + Max ?L) / 2 by simp
  finally have frac d = frac ((1 + Max ?L) / 2) by (simp add:
frac-nat-add-id)
  also have ... = (1 + Max ?L) / 2 using * unfolding frac-eq by
auto
  also have ... > Max ?L using * by auto
  finally show frac d ≥ frac ?l using l by auto
next
case False
then have u: ?u = c + Min ?U frac ?u = Min ?U using Min-in[OF
fin(2) False]

```

```

    by (auto simp: frac-nat-add-id frac-frac)
    from U-intv[OF False] have **:  $0 < \text{Min } ?U \text{ Min } ?U < 1$  by auto
    from l u d have  $d = ((c + c) + (\text{Max } ?L + \text{Min } ?U)) / 2$  by auto
    also have  $\dots = c + (\text{Max } ?L + \text{Min } ?U) / 2$  by simp
    finally have  $\text{frac } d = \text{frac } ((\text{Max } ?L + \text{Min } ?U) / 2)$  by (simp
add: frac-nat-add-id)
    also have  $\dots = (\text{Max } ?L + \text{Min } ?U) / 2$  using * ** unfolding
frac-eq by auto
    also have  $\dots \geq \text{Max } ?L$  using l-le-u[OF F False] by auto
    finally show ?thesis using l by auto
qed
qed
have frac-u:  $?U \neq \{\} \vee ?L = \{\} \longrightarrow \text{frac } d \leq \text{frac } ?u$ 
proof (cases ?U = {})
  case True
  note T = this
  show ?thesis
  proof (cases ?L = {})
    case True
    with T have ?l = ?u by auto
    with d have  $d = ?u$  by auto
    then show ?thesis by auto
  next
  case False
  with T show ?thesis by auto
qed
next
case False
note F = this
then have u:  $?u = c + \text{Min } ?U \text{ frac } ?u = \text{Min } ?U$  using Min-in[OF
fin(2) ⟨?U ≠ {}⟩]
by (auto simp: frac-nat-add-id frac-frac)
from U-intv[OF F] have *:  $0 < \text{Min } ?U \text{ Min } ?U < 1$  by auto
show ?thesis
proof (cases ?L = {})
  case True
  from True F have ?l = c by auto
  with u d have  $d = ((c + c) + \text{Min } ?U) / 2$  by auto
  also have  $\dots = c + \text{Min } ?U / 2$  by simp
  finally have  $\text{frac } d = \text{frac } (\text{Min } ?U / 2)$  by (simp add: frac-nat-add-id)
  also have  $\dots = \text{Min } ?U / 2$  unfolding frac-eq using * by auto
  also have  $\dots \leq \text{Min } ?U$  using ⟨ $0 < \text{Min } ?U$ ⟩ by auto
  finally have  $\text{frac } d \leq \text{frac } ?u$  using u by auto
  then show ?thesis by auto

```

```

next
  case False
  then have  $l: ?l = c + \text{Max } ?L \text{ frac } ?l = \text{Max } ?L$  using Max-in[OF
fin(1) False]
    by (auto simp: frac-nat-add-id frac-frac)
  from L-intv[OF False] have  $** : 0 < \text{Max } ?L \text{ Max } ?L < 1$  by auto
  from  $l \ u \ d$  have  $d = ((c + c) + (\text{Max } ?L + \text{Min } ?U)) / 2$  by auto
  also have  $\dots = c + (\text{Max } ?L + \text{Min } ?U) / 2$  by simp
  finally have  $\text{frac } d = \text{frac } ((\text{Max } ?L + \text{Min } ?U) / 2)$  by (simp
add: frac-nat-add-id)
  also have  $\dots = (\text{Max } ?L + \text{Min } ?U) / 2$  using  $**$  unfolding
frac-eq by auto
  also have  $\dots \leq \text{Min } ?U$  using l-le-u[OF False F] by auto
  finally show ?thesis using  $u$  by auto
  qed
qed
have  $\forall y \in ?X_0 - \{x\}. (y,x) \in r \longleftrightarrow \text{frac } (v \ y) \leq \text{frac } d$ 
proof (safe, goal-cases)
  case ( $1 \ y \ k$ )
  with L-bound[of y] frac-l show ?case by auto
next
  case ( $2 \ y \ k$ )
  show ?case
  proof (rule ccontr, goal-cases)
    case  $1$ 
    with total 2  $\langle x \in X \rangle \text{ Intv}$  have  $(x,y) \in r$  unfolding total-on-def
by auto
    with  $2 \ U\text{-bound}[of y]$  have  $?U \neq \{\}$   $\text{frac } ?u \leq \text{frac } (v \ y)$  by auto
    with frac-u have  $\text{frac } d \leq \text{frac } (v \ y)$  by auto
    with  $2 \ d\text{-frac-equiv } 1$  show False by auto
    qed
  qed
  moreover have  $\forall y \in ?X_0 - \{x\}. (x,y) \in r \longleftrightarrow \text{frac } d \leq \text{frac } (v \ y)$ 
proof (safe, goal-cases)
  case ( $1 \ y \ k$ )
  then have  $?U \neq \{\}$  by auto
  with  $1 \ U\text{-bound}[of y] \text{ frac-u}$  show ?case by auto
next
  case ( $2 \ y \ k$ )
  show ?case
  proof (rule ccontr, goal-cases)
    case  $1$ 
    with total 2  $\langle x \in X \rangle \text{ Intv}$  have  $(y,x) \in r$  unfolding total-on-def
by auto

```

```

    with 2 L-bound[of y] have frac (v y) ≤ frac ?l by auto
    with frac-l have frac (v y) ≤ frac d by auto
    with 2 d-frac-equiv 1 show False by auto
  qed
qed
ultimately have d:
  c < d d < c + 1 ∀ y ∈ ?X0 - {x}. (y,x) ∈ r ⟷ frac (v y) ≤ frac
d
  ∀ y ∈ ?X0 - {x}. (x,y) ∈ r ⟷ frac d ≤ frac (v y)
using d3 by auto
let ?v = v(x := d)
have ?v ∈ region X I r
proof (standard, goal-cases)
  case 1
  from ⟨d ≥ 0⟩ v show ?case by auto
next
  case 2
  show ?case
  proof (safe, goal-cases)
    case (1 y)
    with v have intv-elem y v (?I y) by fast
    with Intv d(1,2) show intv-elem y ?v (I y) by (cases x = y, auto)
(cases I y, auto)
  qed
next
  from ⟨x ∈ X⟩ Intv show ?X0' ∪ {x} = ?X0 by auto
  with r-subset have r ⊆ (?X0' ∪ {x}) × (?X0' ∪ {x}) by auto
  have ∀ x ∈ ?X0'. ∀ y ∈ ?X0'. (x,y) ∈ r ⟷ (x,y) ∈ ?r by auto
  with v have ∀ x ∈ ?X0'. ∀ y ∈ ?X0'. (x,y) ∈ r ⟷ frac (v x) ≤
frac (v y) by fastforce
  then have ∀ x ∈ ?X0'. ∀ y ∈ ?X0'. (x,y) ∈ r ⟷ frac (?v x) ≤
frac (?v y) by auto
  with d(3,4) show ∀ y ∈ ?X0' ∪ {x}. ∀ z ∈ ?X0' ∪ {x}. (y,z) ∈ r
⟷ frac (?v y) ≤ frac (?v z)
  proof (auto, goal-cases)
    case 1
    from refl ⟨x ∈ X⟩ Intv show ?case by (auto simp: refl-on-def)
  qed
qed
then show ?thesis by auto
qed
then obtain d where v(x := d) ∈ R using R by auto
  then have (v(x := d))(x := c) ∈ region-set R x c unfolding re-
gion-set-def by blast

```

**moreover from**  $v \langle x \in X \rangle$  **have**  $(v(x := d))(x := c) = v$  **by** *fastforce*  
**ultimately have**  $v \in \text{region-set } R \ x \ c$  **by** *simp*  
**}**  
  
**ultimately have**  $\text{region-set } R \ x \ c = \text{region } X \ ?I \ ?r$  **by** *blast*  
**with** *valid*  $\mathcal{R}$ -*def* **have**  $*$ :  $\text{region-set } R \ x \ c \in \mathcal{R}$  **by** *auto*  
**moreover from** *assms* **have**  $**$ :  $v \langle x := c \rangle \in \text{region-set } R \ x \ c$  **unfolding**  
*region-set-def* **by** *auto*  
**ultimately show**  $[v(x := c)]_{\mathcal{R}} = \text{region-set } R \ x \ c \ [v(x := c)]_{\mathcal{R}} \in \mathcal{R} \ v(x := c) \in [v(x := c)]_{\mathcal{R}}$   
**using** *region-unique*[*OF* -  $** \ *$ ]  $\mathcal{R}$ -*def* **by** *auto*  
**qed**

**definition** *region-set'*  
**where**

$\text{region-set}' \ R \ r \ c = \{[r \rightarrow c]v \mid v. v \in R\}$

**lemma** *region-set'-id*:

**fixes**  $X \ k$  **and**  $c :: \text{nat}$   
**defines**  $\mathcal{R} \equiv \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$   
**assumes**  $R \in \mathcal{R} \ v \in R \ \text{finite } X \ 0 \leq c \ \forall \ x \in \text{set } r. \ c \leq k \ x \ \text{set } r \subseteq X$   
**shows**  $[[r \rightarrow c]v]_{\mathcal{R}} = \text{region-set}' \ R \ r \ c \ \wedge \ [[r \rightarrow c]v]_{\mathcal{R}} \in \mathcal{R} \ \wedge \ [r \rightarrow c]v \in [[r \rightarrow c]v]_{\mathcal{R}}$  **using** *assms*  
**proof** (*induction r*)  
**case** *Nil*  
**from** *regions-closed*[*OF* - *Nil*(2,3)] *regions-closed'*[*OF* - *Nil*(2,3)] *region-unique*[*OF* - *Nil*(3,2)] *Nil*(1)  
**have**  $[v]_{\mathcal{R}} = R \ [v \oplus 0]_{\mathcal{R}} \in \mathcal{R} \ (v \oplus 0) \in [v \oplus 0]_{\mathcal{R}}$  **by** *auto*  
**then show**  $?case$  **unfolding** *region-set'-def* *eval-add-def* **by** *simp*  
**next**  
**case** (*Cons x xs*)  
**then have**  $[[xs \rightarrow c]v]_{\mathcal{R}} = \text{region-set}' \ R \ xs \ c \ [[xs \rightarrow c]v]_{\mathcal{R}} \in \mathcal{R} \ [xs \rightarrow c]v \in [[xs \rightarrow c]v]_{\mathcal{R}}$  **by** *force+*  
**note**  $IH = \text{this}[\text{unfolded } \mathcal{R}\text{-def}]$   
**let**  $?v = ([xs \rightarrow c]v)(x := c)$   
**from** *region-set-id*[*OF* *IH*(2,3)  $\langle \text{finite } X \rangle \langle c \geq 0 \rangle$ , *of x*]  $\mathcal{R}$ -*def* *Cons.prem*s(5,6)  
**have**  $[?v]_{\mathcal{R}} = \text{region-set} \ ([xs \rightarrow \text{real } c]v)_{\mathcal{R}} \ x \ c \ [?v]_{\mathcal{R}} \in \mathcal{R} \ ?v \in [?v]_{\mathcal{R}}$  **by** *auto*  
**moreover have**  $\text{region-set}' \ R \ (x \# \ xs) \ (\text{real } c) = \text{region-set} \ ([xs \rightarrow \text{real } c]v)_{\mathcal{R}} \ x \ c$   
**unfolding** *region-set-def* *region-set'-def*  
**proof** (*safe, goal-cases*)  
**case** (*1 y u*)  
**let**  $?u = [xs \rightarrow \text{real } c]u$

```

have  $[x \# xs \rightarrow \text{real } c]u = ?u(x := \text{real } c)$  by auto
moreover from  $IH(1)$  1 have  $?u \in [[xs \rightarrow \text{real } c]v]_{\mathcal{R}}$  unfolding  $\mathcal{R}\text{-def}$ 
region-set'-def by auto
ultimately show  $?case$  by auto
next
case  $(2\ y\ u)$ 
with  $IH(1)[\text{unfolded } \text{region-set}'\text{-def } \mathcal{R}\text{-def}[\text{symmetric}]]$  show  $?case$  by
auto
qed
moreover have  $[x \# xs \rightarrow \text{real } c]v = ?v$  by simp
ultimately show  $?case$  by presburger
qed

```

This is the only additional lemma necessary to make local  $\alpha$ -closures work.

**lemma** *region-set-subs*:

```

fixes  $X\ k\ k'$  and  $c :: \text{nat}$ 
defines  $\mathcal{R} \equiv \{\text{region } X\ I\ r \mid I\ r.\ \text{valid-region } X\ k\ I\ r\}$ 
defines  $\mathcal{R}' \equiv \{\text{region } X\ I\ r \mid I\ r.\ \text{valid-region } X\ k'\ I\ r\}$ 
assumes  $R \in \mathcal{R}\ v \in R\ \text{finite } X\ 0 \leq c\ \text{set } cs \subseteq X\ \forall\ y.\ y \notin \text{set } cs \longrightarrow k$ 
 $y \geq k'\ y$ 
shows  $[[cs \rightarrow c]v]_{\mathcal{R}'} \supseteq \text{region-set}'\ R\ cs\ c\ [[cs \rightarrow c]v]_{\mathcal{R}'} \in \mathcal{R}'\ [cs \rightarrow c]v \in$ 
 $[[cs \rightarrow c]v]_{\mathcal{R}'}$ 
proof  $-$ 
from assms obtain  $I\ r$  where  $R: R = \text{region } X\ I\ r\ \text{valid-region } X\ k\ I\ r$ 
 $v \in \text{region } X\ I\ r$  by auto
 $-$  The set of movers, that is all intervals that now are unbounded due to
changing from  $k$  to  $k'$ 
let  $?M = \{x \in X.\ \text{isIntv } (I\ x) \wedge \text{intv-const } (I\ x) \geq k'\ x \vee \text{intv-const } (I\ x) > k'\ x\}$ 
let  $?I = \lambda\ y.$ 
 $\text{if } y \in \text{set } cs \text{ then } (\text{if } c \leq k'\ y \text{ then } \text{Const } c \text{ else } \text{Greater } (k'\ y))$ 
 $\text{else if } (\text{isIntv } (I\ y) \wedge \text{intv-const } (I\ y) \geq k'\ y \vee \text{intv-const } (I\ y) > k'\ y)$ 
 $\text{then } \text{Greater } (k'\ y)$ 
 $\text{else } I\ y$ 
let  $?r = \{(y, z) \in r.\ y \notin \text{set } cs \wedge z \notin \text{set } cs \wedge y \notin ?M \wedge z \notin ?M\}$ 
let  $?X_0 = \{x \in X.\ \exists\ c.\ I\ x = \text{Intv } c\}$ 
let  $?X_0' = \{x \in X.\ \exists\ c.\ ?I\ x = \text{Intv } c\}$ 

have  $r\text{-subset}: r \subseteq ?X_0 \times ?X_0$  and  $\text{refl}: \text{refl-on } ?X_0\ r$  and  $\text{trans}: \text{trans}$ 
 $r$  and
 $\text{total}: \text{total-on } ?X_0\ r$ 
using  $R(2)$  by auto

have  $\text{valid}: \text{valid-region } X\ k'\ ?I\ ?r$ 

```

```

proof
  show  $?X_0' = ?X_0'$  by auto
next
  show  $?r \subseteq ?X_0' \times ?X_0'$ 
    using r-subset by auto
next
  from refl show refl-on  $?X_0'$   $?r$  unfolding refl-on-def by auto
next
  from trans show trans  $?r$  unfolding trans-def by auto
next
  from total show total-on  $?X_0'$   $?r$  unfolding total-on-def by auto
next
  from R(2) have  $\forall x \in X. \text{valid-intv } (k \ x) \ (I \ x)$  by auto
  then show  $\forall x \in X. \text{valid-intv } (k' \ x) \ (?I \ x)$ 
    apply safe
    subgoal for  $x'$ 
      using  $\langle \forall y. y \notin \text{set } cs \longrightarrow k \ y \geq k' \ y \rangle$ 
      by (cases  $I \ x'$ ; force)
    done
qed

{ fix  $v$  assume  $v: v \in \text{region-set}' \ R \ cs \ c$ 
  with R(1) obtain  $v'$  where  $v': v' \in \text{region } X \ I \ r \ v = [cs \rightarrow c]v'$ 
    unfolding region-set'-def by auto
  have  $v \in \text{region } X \ ?I \ ?r$ 
  proof (standard, goal-cases)
    case 1
    from  $v' \langle 0 \leq c \rangle$  show  $?case$ 
      apply –
      apply rule
      subgoal for  $x$ 
        by (cases  $x \in \text{set } cs$ ) auto
      done
    next
    case 2
    from  $v'$  show  $?case$ 
      apply –
      apply rule
      subgoal for  $x'$ 
        by (cases  $I \ x'$ ; cases  $x' \in \text{set } cs$ ; force)
      done
    next
    show  $?X_0' = ?X_0'$  by auto
  next

```

**from**  $v'$  **show**  $\forall y \in ?X_0'. \forall z \in ?X_0'. (y,z) \in ?r \longleftrightarrow \text{frac}(v y) \leq \text{frac}(v z)$  **by** *auto*  
**qed**  
**}**  
**then have** *region-set'*  $R$   $cs$   $c \subseteq \text{region } X$   $?I$   $?r$  **by** *blast*  
**moreover from** *valid* **have**  $*$ : *region*  $X$   $?I$   $?r \in \mathcal{R}'$  **unfolding**  $\mathcal{R}'$ -*def* **by** *blast*  
**moreover from** *assms* **have**  $*$ :  $[cs \rightarrow c]v \in \text{region-set}' R$   $cs$   $c$  **unfolding** *region-set'-def* **by** *auto*  
**ultimately show**  
 $[[cs \rightarrow c]v]_{\mathcal{R}'} \supseteq \text{region-set}' R$   $cs$   $c$   $[[cs \rightarrow c]v]_{\mathcal{R}'} \in \mathcal{R}'$   $[cs \rightarrow c]v \in [[cs \rightarrow c]v]_{\mathcal{R}'}$   
**using** *region-unique*[*of*  $\mathcal{R}'$ , *OF* - -  $*$ , *unfolded*  $\mathcal{R}'$ -*def*, *OF* *HOL.refl*]  
**unfolding**  $\mathcal{R}'$ -*def*[*symmetric*] **by** *auto*  
**qed**

## 5.6 A Semantics Based on Regions

### 5.6.1 Single step

**inductive** *step-r* ::

$\langle 'a, 'c, t, 's \rangle \text{ta} \Rightarrow \langle 'c, t \rangle \text{zone set} \Rightarrow 's \Rightarrow \langle 'c, t \rangle \text{zone} \Rightarrow 's \Rightarrow \langle 'c, t \rangle \text{zone} \Rightarrow \text{bool}$

$\langle \langle -, - \rangle \vdash \langle -, - \rangle \rightsquigarrow \langle -, - \rangle \rangle [61,61,61,61]$  61)

**where**

*step-t-r*:

$\llbracket \mathcal{R} = \{ \text{region } X \text{ I } r \mid \text{I } r. \text{valid-region } X \text{ k I } r \}; \text{valid-abstraction } A \text{ X k}; R \in \mathcal{R}; R' \in \text{Succ } \mathcal{R} R;$

$R' \subseteq \llbracket \text{inv-of } A \text{ l} \rrbracket \rrbracket \Longrightarrow A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l, R' \rangle \mid$

*step-a-r*:

$\llbracket \mathcal{R} = \{ \text{region } X \text{ I } r \mid \text{I } r. \text{valid-region } X \text{ k I } r \}; \text{valid-abstraction } A \text{ X k}; A \vdash l \longrightarrow^{g,a,r} l'; R \in \mathcal{R} \rrbracket$

$\Longrightarrow A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', \text{region-set}'(R \cap \{u. u \vdash g\}) \text{ r } 0 \cap \{u. u \vdash \text{inv-of } A \text{ l}'\} \rangle$

**inductive-cases**[*elim!*]:  $A, \mathcal{R} \vdash \langle l, u \rangle \rightsquigarrow \langle l', u' \rangle$

**declare** *step-r.intros*[*intro*]

**lemma** *region-cover'*:

**assumes**  $\mathcal{R} = \{ \text{region } X \text{ I } r \mid \text{I } r. \text{valid-region } X \text{ k I } r \}$  **and**  $\forall x \in X. 0 \leq v x$

**shows**  $v \in [v]_{\mathcal{R}}$   $[v]_{\mathcal{R}} \in \mathcal{R}$

**proof** –

**from** *region-cover*[*OF* *assms*(2), *of* *k*] *assms* **obtain** *R* **where** *R*:  $R \in \mathcal{R}$   
 $v \in R$  **by** *auto*  
**from** *regions-closed'*[*OF* *assms*(1) *R*, *of* 0] **show**  $v \in [v]_{\mathcal{R}}$  **unfolding**  
*cval-add-def* **by** *auto*  
**from** *regions-closed*[*OF* *assms*(1) *R*, *of* 0] **show**  $[v]_{\mathcal{R}} \in \mathcal{R}$  **unfolding**  
*cval-add-def* **by** *auto*  
**qed**

**lemma** *step-r-complete-aux*:

**fixes** *R r A l' g*  
**defines**  $R' \equiv \text{region-set}' (R \cap \{u. u \vdash g\}) \ r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**assumes**  $\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$   
**and** *valid-abstraction*  $A \ X \ k$   
**and**  $u \in R$   
**and**  $R \in \mathcal{R}$   
**and**  $A \vdash l \longrightarrow^{g,a,r} l'$   
**and**  $u \vdash g$   
**and**  $[r \rightarrow 0]u \vdash \text{inv-of } A \ l'$   
**shows**  $R = R \cap \{u. u \vdash g\} \wedge R' = \text{region-set}' \ R \ r \ 0 \wedge R' \in \mathcal{R}$   
**proof** –  
**note**  $A = \text{assms}(2-)$   
**from**  $A(2)$  **have** \*:  
 $\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
 $\text{collect-clkvt } (\text{trans-of } A) \subseteq X$   
 $\text{finite } X$   
**by** (*fastforce elim: valid-abstraction.cases*)+  
**from**  $A(5) \ *(2)$  **have**  $r$ :  $\text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by** *fastforce*  
**from**  $\*(1) \ A(5)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } g. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clkt-def* **by** *fastforce*  
**from** *ccompatible*[*OF* *this*, *folded*  $A(1)$ ]  $A(3,4,6)$  **have**  $R \subseteq \{g\}$   
**unfolding** *ccompatible-def cval-def* **by** *blast*  
**then** **have**  $R\text{-id}$ :  $R \cap \{u. u \vdash g\} = R$  **unfolding** *cval-def* **by** *auto*  
**from** *region-set'-id*[*OF*  $A(4)$ [*unfolded*  $A(1)$ ]  $A(3) \ *(3)$  - -  $r$ , *of* 0, *folded*  $A(1)$ ]  
**have** \*\*:  
 $[[r \rightarrow 0]u]_{\mathcal{R}} = \text{region-set}' \ R \ r \ 0 \ [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R} \ [r \rightarrow 0]u \in [[r \rightarrow 0]u]_{\mathcal{R}}$   
**by** *auto*  
**let**  $?R = [[r \rightarrow 0]u]_{\mathcal{R}}$   
**from**  $\*(1) \ A(5)$  **have** \*\*\*:  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l'). m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *inv-of-def clkp-set-def collect-clki-def* **by** *fastforce*  
**from** *ccompatible*[*OF* *this*, *folded*  $A(1)$ ]  $\*(2-)$   $A(7)$  **have**  $?R \subseteq \{\text{inv-of}$

$A \ l'\}$   
**unfolding** *ccompatible-def ccval-def by blast*  
**then have**  $***: ?R \cap \{u. u \vdash \text{inv-of } A \ l'\} = ?R$  **unfolding** *ccval-def by auto*  
**with**  $*(1,2)$  *R-id* **show** *?thesis by (auto simp: R'-def)*  
**qed**

**lemma** *step-r-complete:*

$\llbracket A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle; \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\};$   
*valid-abstraction*  $A \ X \ k;$

$\forall x \in X. u \ x \geq 0 \rrbracket \implies \exists R'. A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l', R' \rangle \wedge u' \in R' \wedge R' \in \mathcal{R}$

**proof** (*induction rule: step.induct, goal-cases*)

**case**  $(1 \ A \ l \ u \ a \ l' \ u')$

**note**  $A = \text{this}$

**then obtain**  $g \ r$  **where**  $u': u' = [r \rightarrow 0]u \ A \vdash l \xrightarrow{g, a, r} l' \ u \vdash g \ u' \vdash \text{inv-of } A \ l'$

**by** (*cases rule: step-a.cases*) *auto*

**let**  $?R' = \text{region-set}'([u]_{\mathcal{R}}) \cap \{u. u \vdash g\} \ r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$

**from** *region-cover'[OF A(2,4)]* **have**  $R: [u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$  **by** *auto*

**from** *step-r-complete-aux[OF A(2,3) this(2,1) u'(2,3)]*  $u'$

**have**  $*$ :  $[u]_{\mathcal{R}} = ([u]_{\mathcal{R}}) \cap \{u. u \vdash g\} \ ?R' = \text{region-set}'([u]_{\mathcal{R}}) \ r \ 0 \ ?R' \in \mathcal{R}$

**by** *auto*

**from**  $1(2,3)$  **have** *collect-clkvt (trans-of A)  $\subseteq X$  finite X* **by** (*auto elim: valid-abstraction.cases*)

**with**  $u'(2)$  **have**  $r: \text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by** *fastforce*

**from**  $* \ u'(1) \ R(2)$  **have**  $u' \in ?R'$  **unfolding** *region-set'-def* **by** *auto*

**moreover** **have**  $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l', ?R' \rangle$  **using**  $R(1) \ A(2,3) \ u'(2)$

**by** *auto*

**ultimately show** *?case using \*(3)* **by** *meson*

**next**

**case**  $(2 \ A \ l \ u \ d \ l' \ u')$

**hence**  $u': u' = (u \oplus d) \ u \oplus d \vdash \text{inv-of } A \ l \ 0 \leq d$  **and**  $l = l'$  **by** (*auto elim!: step-t.cases*)

**from** *region-cover'[OF 2(2,4)]* **have**  $R: [u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$  **by** *auto*

**from** *SuccI2[OF 2(2) this(2,1)  $\langle 0 \leq d \rangle$ , of  $[u]_{\mathcal{R}}$   $u'(1)$ ]* **have**  $u'1:$

$[u]_{\mathcal{R}} \in \text{Succ } \mathcal{R} \ ([u]_{\mathcal{R}}) \ [u]_{\mathcal{R}} \in \mathcal{R}$

**by** *auto*

**from** *regions-closed'[OF 2(2) R(1,2)  $\langle 0 \leq d \rangle$   $u'(1)$ ]* **have**  $u'2: u' \in [u]_{\mathcal{R}}$

**by** *simp*

**from**  $2(3)$  **have**  $*$ :

$\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$

*collect-clkvt (trans-of A)  $\subseteq X$*

*finite X*

**by** (*fastforce elim: valid-abstraction.cases*) +  
**from**  $\ast(1)$   $u'(2)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs} \text{ (inv-of } A \ l). \ m \leq \text{real}$   
 $(k \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** *fastforce*  
**from** *ccompatible[OF this, folded 2(2)]*  $u'1(2)$   $u'2$   $u'(1,2,3)$   $R$  **have**  
 $[u']_{\mathcal{R}} \subseteq \{\text{inv-of } A \ l\}$   
**unfolding** *ccompatible-def ccval-def* **by** *auto*  
**with**  $2$   $u'1$   $R(1)$  **have**  $A, \mathcal{R} \vdash \langle l, ([u']_{\mathcal{R}}) \rangle \rightsquigarrow \langle l, ([u']_{\mathcal{R}}) \rangle$  **by** *auto*  
**with**  $u'1(2)$   $u'2$   $\langle l = l' \rangle$  **show** *?case* **by** *meson*  
**qed**

Compare this to lemma *step-z-sound*. This version is weaker because for regions we may very well arrive at a successor for which not every valuation can be reached by the predecessor. This is the case for e.g. the region with only Greater (k x) bounds.

**lemma** *step-r-sound*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \ \text{valid-region } X \ k \ I \ r\}$   
 $\implies R' \neq \{\}$   $\implies (\forall u \in R. \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle)$

**proof** (*induction rule: step-r.induct*)

**case** (*step-t-r*  $\mathcal{R}$   $X$   $k$   $A$   $R$   $R'$   $l$ )

**note**  $A = \text{this}[\text{unfolded this}(1)]$

**show** *?case*

**proof**

**fix**  $u$  **assume**  $u: u \in R$

**from** *set-of-regions[OF A(3) this A(4), folded step-t-r(1)]*  $A(2)$

**obtain**  $t$  **where**  $t: t \geq 0$   $[u \oplus t]_{\mathcal{R}} = R'$  **by** (*auto elim: valid-abstraction.cases*)

**with** *regions-closed'[OF A(1,3) u this(1)]* *step-t-r(1)* **have**  $\ast: (u \oplus t)$

$\in R'$  **by** *auto*

**with**  $u$   $t(1)$   $A(5,6)$  **have**  $A \vdash \langle l, u \rangle \rightarrow \langle l, (u \oplus t) \rangle$  **unfolding** *ccval-def*

**by** *auto*

**with**  $t \ast$  **show**  $\exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$  **by** *meson*

**qed**

**next**

**case**  $A: (\text{step-a-r } \mathcal{R} \ X \ k \ A \ l \ g \ a \ r \ l' \ R)$

**show** *?case*

**proof**

**fix**  $u$  **assume**  $u: u \in R$

**from**  $A(6)$  **obtain**  $v$  **where**  $v: v \in R$   $v \vdash g$   $[r \rightarrow 0]v \vdash \text{inv-of } A \ l'$

**unfolding** *region-set'-def* **by** *auto*

**let**  $?R' = \text{region-set}'(R \cap \{u. u \vdash g\})$   $r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$

**from** *step-r-complete-aux[OF A(1,2) v(1) A(4,3) v(2-)]* **have**  $R:$

$R = R \cap \{u. u \vdash g\}$   $?R' = \text{region-set}' R \ r \ 0$

**by** *auto*

**from**  $A$  **have** *collect-clkvt (trans-of A)  $\subseteq X$*  **by** (*auto elim: valid-abstraction.cases*)

**with**  $A(\beta)$  **have**  $r: \text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by** *fastforce*  
**from**  $u \in R$  **have**  $*$ :  $[r \rightarrow 0]u \in ?R' \ u \vdash g \ [r \rightarrow 0]u \vdash \text{inv-of } A \ l'$  **unfolding**  
*region-set'-def* **by** *auto*  
**with**  $A(\beta)$  **have**  $A \vdash \langle l, u \rangle \rightarrow \langle l', [r \rightarrow 0]u \rangle$  **apply** (*intro step.intros(1)*)  
**apply** *rule* **by** *auto*  
**with**  $*$  **show**  $\exists a \in ?R'. A \vdash \langle l, u \rangle \rightarrow \langle l', a \rangle$  **by** *meson*  
**qed**  
**qed**

### 5.6.2 Multi Step

**inductive**

$\text{steps-r} :: ('a, 'c, t, 's) \text{ta} \Rightarrow ('c, t) \text{zone set} \Rightarrow 's \Rightarrow ('c, t) \text{zone} \Rightarrow 's \Rightarrow$   
 $('c, t) \text{zone} \Rightarrow \text{bool}$   
 $(\langle -, - \rangle \vdash \langle -, - \rangle \rightsquigarrow^* \langle -, - \rangle) \ [61,61,61,61,61,61] \ 61$

**where**

$\text{refl}: A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l, R \rangle \ |$   
 $\text{step}: A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \Longrightarrow A, \mathcal{R} \vdash \langle l', R' \rangle \rightsquigarrow \langle l'', R'' \rangle \Longrightarrow A, \mathcal{R} \vdash$   
 $\langle l, R \rangle \rightsquigarrow^* \langle l'', R'' \rangle$

**declare**  $\text{steps-r.intros}[\text{intro}]$

**lemma** *steps-alt*:

$A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle \Longrightarrow A \vdash \langle l', u' \rangle \rightarrow \langle l'', u'' \rangle \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^* \langle l'', u'' \rangle$   
**by** (*induction rule: steps.induct*) *auto*

**lemma** *emptiness-preservation*:  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \Longrightarrow R = \{\} \Longrightarrow R' = \{\}$

**by** (*induction rule: step-r.cases*) (*auto simp: region-set'-def*)

**lemma** *emptiness-preservation-steps*:  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \Longrightarrow R = \{\} \Longrightarrow R' = \{\}$

**apply** (*induction rule: steps-r.induct*)

**apply** *blast*

**apply** (*subst emptiness-preservation*)

**by** *blast+*

Note how it is important to define the multi-step semantics “the right way round”. This is also the direction Bouyer implies for her implicit induction.

**lemma** *steps-r-sound*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \Longrightarrow \mathcal{R} = \{\text{region } X \ I \ r \ | \ I \ r. \text{valid-region } X \ k \ I \ r\}$

$\Longrightarrow R' \neq \{\} \Longrightarrow u \in R \Longrightarrow \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$

**proof** (*induction rule: steps-r.induct*)

**case** *refl* **then show** *?case* **by** *auto*  
**next**  
**case** (*step A R l R l' R' l'' R''*)  
**from** *emptiness-preservance[OF step.hyps(2)] step.prem* **have**  $R' \neq \{\}$   
**by** *fastforce*  
**with** *step* **obtain**  $u'$  **where**  $u' \in R' \ A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$  **by** *auto*  
**with** *step-r-sound[OF step(2,4,5)]* **obtain**  $u''$  **where**  $u'' \in R'' \ A \vdash \langle l', u' \rangle \rightarrow \langle l'', u'' \rangle$  **by** *blast*  
**with**  $u'$  **show** *?case* **by** (*auto 4 5 intro: steps-alt*)  
**qed**

**lemma** *steps-r-sound'*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \implies \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\}$   
 $\implies R' \neq \{\} \implies (\exists u' \in R'. \exists u \in R. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle)$

**proof** *goal-cases*

**case** *1*

**with** *emptiness-preservance-steps[OF this(1)]* **obtain**  $u$  **where**  $u \in R$  **by** *auto*

**with** *steps-r-sound[OF 1 this]* **show** *?case* **by** *auto*

**qed**

**lemma** *single-step-r*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle$   
**by** (*metis steps-r.refl steps-r.step*)

**lemma** *steps-r-alt*:

$A, \mathcal{R} \vdash \langle l', R' \rangle \rightsquigarrow^* \langle l'', R'' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l'', R'' \rangle$

**apply** (*induction rule: steps-r.induct*)

**apply** (*rule single-step-r*)

**by** *auto*

**lemma** *single-step*:

$x1 \vdash \langle x2, x3 \rangle \rightarrow \langle x4, x5 \rangle \implies x1 \vdash \langle x2, x3 \rangle \rightarrow^* \langle x4, x5 \rangle$

**by** (*metis steps.intros*)

**lemma** *steps-r-complete*:

$\llbracket A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle; \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\}; \text{ valid-abstraction } A \ X \ k;$

$\forall x \in X. u \ x \geq 0 \rrbracket \implies \exists R'. A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow^* \langle l', R' \rangle \wedge u' \in R'$

**proof** (*induction rule: steps.induct*)

**case** (*refl A l u*)

**from** *region-cover'[OF refl(1,3)]* **show** *?case* **by** *auto*

**next**  
 case (step A l u l' u' l'' u'')  
 from step-r-complete[OF step(1,4-6)] obtain R' where R':  
 $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l', R' \rangle$   $u' \in R'$   $R' \in \mathcal{R}$   
 by auto  
 with step(4)  $\langle u' \in R' \rangle$  have  $\forall x \in X. 0 \leq u' x$  by auto  
 with step obtain R'' where R'':  $A, \mathcal{R} \vdash \langle l', ([u']_{\mathcal{R}}) \rangle \rightsquigarrow^* \langle l'', R'' \rangle$   $u'' \in R''$   
 by auto  
 with region-unique[OF step(4) R'(2,3)] R'(1) have  $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow^* \langle l'', R'' \rangle$   
 by (subst steps-r-alt) auto  
 with R'' region-cover'[OF step(4,6)] show ?case by auto  
**qed**

**end**  
**theory** Closure  
 imports Regions  
**begin**

## 5.7 Correct Approximation of Zones with $\alpha$ -regions

**lemma** subset-int-mono:  $A \subseteq B \implies A \cap C \subseteq B \cap C$  by blast

**lemma** zone-set-mono:

$A \subseteq B \implies \text{zone-set } A \ r \subseteq \text{zone-set } B \ r$

**unfolding** zone-set-def by auto

**lemma** zone-delay-mono:

$A \subseteq B \implies A^\dagger \subseteq B^\dagger$

**unfolding** zone-delay-def by auto

**lemma** step-z-mono:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies Z \subseteq W \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle \wedge Z' \subseteq W'$

**proof** (cases rule: step-z.cases, assumption, goal-cases)

case A: 1

let  $?W' = W^\dagger \cap \{u. u \vdash \text{inv-of } A \ l\}$

from A have  $A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', ?W' \rangle$  by auto

moreover have  $Z' \subseteq ?W'$

apply (subst A(5))

apply (rule subset-int-mono)

by (auto intro!: zone-delay-mono A(2))

```

ultimately show ?thesis by meson
next
case A: (2 g a r)
let ?W' = zone-set (W ∩ {u. u ⊢ g}) r ∩ {u. u ⊢ inv-of A l'}
from A have A ⊢ ⟨l, W⟩  $\rightsquigarrow$ 1a ⟨l', ?W'⟩ by auto
moreover have Z' ⊆ ?W'
  apply (subst A(4))
  apply (rule subset-int-mono)
  apply (rule zone-set-mono)
  apply (rule subset-int-mono)
  apply (rule A(2))
done
ultimately show ?thesis by (auto simp: A(3))
qed

```

## 5.8 Old Variant Using a Global Set of Regions

Shared Definitions for Local and Global Sets of Regions locale

*Alpha-defs* =

fixes  $X :: 'c \text{ set}$

begin

**definition**  $V :: ('c, t) \text{ cval set}$  where  $V \equiv \{v . \forall x \in X. v \ x \geq 0\}$

**lemma**  $up\text{-}V: Z \subseteq V \implies Z^\uparrow \subseteq V$

**unfolding**  $V\text{-def}$   $zone\text{-}delay\text{-}def$   $cval\text{-}add\text{-}def$  by auto

**lemma**  $reset\text{-}V: Z \subseteq V \implies (zone\text{-}set \ Z \ r) \subseteq V$

**unfolding**  $V\text{-def}$  **unfolding**  $zone\text{-}set\text{-}def$  by (induction  $r$ , auto)

**lemma**  $step\text{-}z\text{-}V: A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$

apply (induction rule:  $step\text{-}z.induct$ )

apply (rule  $le\text{-}infI1$ )

apply (rule  $up\text{-}V$ )

apply blast

apply (rule  $le\text{-}infI1$ )

apply (rule  $reset\text{-}V$ )

by blast

end

This is the classic variant using a global clock ceiling  $k$  and thus a global set of regions. It is also the version that is necessary to prove the classic extrapolation correct. It is preserved here for comparison with P. Bouyer's

proofs and to outline the only slight adoptions that are necessary to obtain the new version.

```

locale AlphaClosure-global =
  Alpha-defs X for X :: 'c set +
  fixes k  $\mathcal{R}$ 
  defines  $\mathcal{R} \equiv \{ \text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r \}$ 
  assumes finite: finite X
begin

lemmas set-of-regions-spec = set-of-regions[OF - - - finite, of - k, folded
\mathcal{R}-def]
lemmas region-cover-spec = region-cover[of X - k, folded \mathcal{R}-def]
lemmas region-unique-spec = region-unique[of  $\mathcal{R}$  X k, folded \mathcal{R}-def, simplified]
lemmas regions-closed'-spec = regions-closed'[of  $\mathcal{R}$  X k, folded \mathcal{R}-def, simplified]

```

```

lemma valid-regions-distinct-spec:
   $R \in \mathcal{R} \implies R' \in \mathcal{R} \implies v \in R \implies v \in R' \implies R = R'$ 
unfolding \mathcal{R}-def using valid-regions-distinct
by auto (drule valid-regions-distinct, assumption+, simp)+

```

```

definition cla ( $\langle \text{Closure}_\alpha \rightarrow [71] \ 71 \rangle$ )
where
   $cla \ Z = \bigcup \{ R \in \mathcal{R}. R \cap Z \neq \{\} \}$ 

```

**The Nice and Easy Properties Proved by Bouyer** **lemma** *closure-constraint-id*:

```

 $\forall (x, m) \in \text{collect-clock-pairs} \ g. \ m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbf{N} \implies$ 
 $\text{Closure}_\alpha \ \{g\} = \{g\} \cap V$ 

```

**proof** *goal-cases*

```

case 1
show ?case
proof auto
  fix v assume v:  $v \in \text{Closure}_\alpha \ \{g\}$ 
  then obtain R where R:  $v \in R \ R \in \mathcal{R} \ R \cap \{g\} \neq \{\}$  unfolding
cla-def by auto
  with compatible[OF 1, folded \mathcal{R}-def] show  $v \in \{g\}$  unfolding compatible-def by auto
  from R show  $v \in V$  unfolding V-def \mathcal{R}-def by auto
next
  fix v assume v:  $v \in \{g\} \ v \in V$ 
  with region-cover[of X v k, folded \mathcal{R}-def] obtain R where  $R \in \mathcal{R} \ v \in$ 

```

*R* unfolding *V*-def by auto  
 then show  $v \in \text{Closure}_\alpha \{g\}$  unfolding *cla-def* using *v* by auto  
 qed  
 qed

lemma *closure-id'*:

$Z \neq \{\}$   $\implies Z \subseteq R \implies R \in \mathcal{R} \implies \text{Closure}_\alpha Z = R$

proof *goal-cases*

case 1

note *A* = *this*

then have  $R \subseteq \text{Closure}_\alpha Z$  unfolding *cla-def* by auto

moreover

{ fix *R'* assume *R'*:  $Z \cap R' \neq \{\}$   $R' \in \mathcal{R}$   $R \neq R'$

with *A* obtain *v* where  $v \in R$   $v \in R'$  by auto

with  $\mathcal{R}$ -regions-distinct[*OF* - *A*(3) *this*(1) *R'*(2-)]  $\mathcal{R}$ -def have *False*

by auto

}

ultimately show *?thesis* unfolding *cla-def* by auto

qed

lemma *closure-id*:

$\text{Closure}_\alpha Z \neq \{\}$   $\implies Z \subseteq R \implies R \in \mathcal{R} \implies \text{Closure}_\alpha Z = R$

proof *goal-cases*

case 1

then have  $Z \neq \{\}$  unfolding *cla-def* by auto

with 1 *closure-id'* show *?case* by *blast*

qed

lemma *closure-update-mono*:

$Z \subseteq V \implies \text{set } r \subseteq X \implies \text{zone-set } (\text{Closure}_\alpha Z) r \subseteq \text{Closure}_\alpha(\text{zone-set } Z r)$

proof –

assume *A*:  $Z \subseteq V$   $\text{set } r \subseteq X$

let  $?U = \{R \in \mathcal{R}. Z \cap R \neq \{\}\}$

from *A*(1) *region-cover-spec* have  $\forall v \in Z. \exists R. R \in \mathcal{R} \wedge v \in R$

unfolding *V*-def by auto

then have  $Z = \bigcup \{Z \cap R \mid R. R \in ?U\}$

proof (*auto*, *goal-cases*)

case (1 *v*)

then obtain *R* where  $R \in \mathcal{R}$   $v \in R$  by auto

moreover with 1 have  $Z \cap R \neq \{\}$   $v \in Z \cap R$  by auto

ultimately show *?case* by auto

qed

then obtain *U* where  $U: Z = \bigcup \{Z \cap R \mid R. R \in U\} \forall R \in U. R \in$

$\mathcal{R}$  by *blast*  
**{ fix  $R$  assume  $R: R \in U$**   
**{ fix  $v'$  assume  $v': v' \in \text{zone-set } (\text{Closure}_\alpha (Z \cap R)) r - \text{Closure}_\alpha(\text{zone-set } (Z \cap R) r)$**   
**then obtain  $v$  where  $*$ :**  
 $v \in \text{Closure}_\alpha (Z \cap R) v' = [r \rightarrow 0]v$   
**unfolding zone-set-def by auto**  
**with closure-id[*of  $Z \cap R R$* ]  $R U(2)$  have  $**$ :**  
 $\text{Closure}_\alpha (Z \cap R) = R \text{Closure}_\alpha (Z \cap R) \in \mathcal{R}$   
**by fastforce+**  
**with region-set'-id[*OF - \*(1) finite - - A(2), of k 0, folded  $\mathcal{R}$ -def, OF this(2)*]**  
**have  $***$ : zone-set  $R r \in \mathcal{R} [r \rightarrow 0]v \in \text{zone-set } R r$**   
**unfolding zone-set-def region-set'-def by auto**  
**from  $*$  have  $Z \cap R \neq \{\}$  unfolding cla-def by auto**  
**then have zone-set  $(Z \cap R) r \neq \{\}$  unfolding zone-set-def by auto**  
**from closure-id'[*OF this - \*\*\*(1)*] have  $\text{Closure}_\alpha \text{zone-set } (Z \cap R) r = \text{zone-set } R r$**   
**unfolding zone-set-def by auto**  
**with  $v' ** (1)$  have *False* by auto**  
**}**  
**then have zone-set  $(\text{Closure}_\alpha (Z \cap R)) r \subseteq \text{Closure}_\alpha(\text{zone-set } (Z \cap R) r)$  by auto**  
**} note  $Z\text{-}i = \text{this}$**   
**from  $U(1)$  have  $\text{Closure}_\alpha Z = \bigcup \{\text{Closure}_\alpha (Z \cap R) \mid R. R \in U\}$**   
**unfolding cla-def by auto**  
**then have zone-set  $(\text{Closure}_\alpha Z) r = \bigcup \{\text{zone-set } (\text{Closure}_\alpha (Z \cap R)) r \mid R. R \in U\}$**   
**unfolding zone-set-def by auto**  
**also have  $\dots \subseteq \bigcup \{\text{Closure}_\alpha(\text{zone-set } (Z \cap R) r) \mid R. R \in U\}$  using  $Z\text{-}i$  by auto**  
**also have  $\dots = \text{Closure}_\alpha \bigcup \{(\text{zone-set } (Z \cap R) r) \mid R. R \in U\}$  unfolding cla-def by auto**  
**also have  $\dots = \text{Closure}_\alpha \text{zone-set } (\bigcup \{Z \cap R \mid R. R \in U\}) r$**   
**proof goal-cases**  
**case 1**  
**have zone-set  $(\bigcup \{Z \cap R \mid R. R \in U\}) r = \bigcup \{(\text{zone-set } (Z \cap R) r) \mid R. R \in U\}$**   
**unfolding zone-set-def by auto**  
**then show ?case by auto**  
**qed**  
**finally show zone-set  $(\text{Closure}_\alpha Z) r \subseteq \text{Closure}_\alpha(\text{zone-set } Z r)$  using  $U$  by simp**  
**qed**

**lemma** *SuccI3*:

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies (v \oplus t) \in R' \implies R' \in \mathcal{R} \implies R' \in \text{Succ}$   
 $\mathcal{R} R$

**apply** (*intro SuccI2*[of  $\mathcal{R} X k$ , *folded  $\mathcal{R}$ -def, simplified*])

**apply** *assumption+*

**apply** (*intro region-unique*[of  $\mathcal{R} X k$ , *folded  $\mathcal{R}$ -def, simplified, symmetric*])

**by** *assumption+*

**lemma** *closure-delay-mono*:

$Z \subseteq V \implies (\text{Closure}_\alpha Z)^\dagger \subseteq \text{Closure}_\alpha (Z^\dagger)$

**proof**

**fix**  $v$  **assume**  $v: v \in (\text{Closure}_\alpha Z)^\dagger$  **and**  $Z: Z \subseteq V$

**then obtain**  $u u' t R$  **where**  $A$ :

$u \in \text{Closure}_\alpha Z v = (u \oplus t) u \in R u' \in R R \in \mathcal{R} u' \in Z t \geq 0$

**unfolding** *cla-def zone-delay-def* **by** *blast*

**from**  $A(3,5)$  **have**  $\forall x \in X. u x \geq 0$  **unfolding**  *$\mathcal{R}$ -def* **by** *fastforce*

**with** *region-cover-spec*[of  $v$ ]  $A(2,7)$  **obtain**  $R'$  **where**  $R'$ :

$R' \in \mathcal{R} v \in R'$

**unfolding** *cval-add-def* **by** *auto*

**with** *set-of-regions-spec*[OF  $A(5,4)$ , OF *SuccI3*, of  $u$ ]  $A$  **obtain**  $t$  **where**

$t$ :

$t \geq 0 [u' \oplus t]_{\mathcal{R}} = R'$

**by** *auto*

**with**  $A$  **have**  $(u' \oplus t) \in Z^\dagger$  **unfolding** *zone-delay-def* **by** *auto*

**moreover from** *regions-closed'-spec*[OF  $A(5,4)$ ]  $t$  **have**  $(u' \oplus t) \in R'$  **by**  
*auto*

**ultimately have**  $R' \cap (Z^\dagger) \neq \{\}$  **by** *auto*

**with**  $R'$  **show**  $v \in \text{Closure}_\alpha (Z^\dagger)$  **unfolding** *cla-def* **by** *auto*

**qed**

**lemma** *region-V*:  $R \in \mathcal{R} \implies R \subseteq V$  **using** *V-def  $\mathcal{R}$ -def region.cases* **by**  
*auto*

**lemma** *closure-V*:

$\text{Closure}_\alpha Z \subseteq V$

**unfolding** *cla-def* **using** *region-V* **by** *auto*

**lemma** *closure-V-int*:

$\text{Closure}_\alpha Z = \text{Closure}_\alpha (Z \cap V)$

**unfolding** *cla-def* **using** *region-V* **by** *auto*

**lemma** *closure-constraint-mono*:

$Closure_\alpha g = g \implies g \cap (Closure_\alpha Z) \subseteq Closure_\alpha (g \cap Z)$   
**unfolding** *cla-def* **by** *auto*

**lemma** *closure-constraint-mono'*:

**assumes**  $Closure_\alpha g = g \cap V$

**shows**  $g \cap (Closure_\alpha Z) \subseteq Closure_\alpha (g \cap Z)$

**proof** –

**from** *assms closure-V-int* **have**  $Closure_\alpha (g \cap V) = g \cap V$  **by** *auto*

**from** *closure-constraint-mono[OF this, of Z]* **have**

$g \cap (V \cap Closure_\alpha Z) \subseteq Closure_\alpha (g \cap Z \cap V)$

**by** (*metis Int-assoc Int-commute*)

**with** *closure-V[of Z] closure-V-int[of g ∩ Z]* **show** *?thesis* **by** *auto*

**qed**

**lemma** *cla-empty-iff*:

$Z \subseteq V \implies Z = \{\} \longleftrightarrow Closure_\alpha Z = \{\}$

**unfolding** *cla-def V-def* **using** *region-cover-spec* **by** *fast*

**lemma** *closure-involutive-aux*:

$U \subseteq \mathcal{R} \implies Closure_\alpha \bigcup U = \bigcup U$

**unfolding** *cla-def* **using** *valid-regions-distinct-spec* **by** *blast*

**lemma** *closure-involutive-aux'*:

$\exists U. U \subseteq \mathcal{R} \wedge Closure_\alpha Z = \bigcup U$

**unfolding** *cla-def* **by** (*rule exI[where x = {R ∈ R. R ∩ Z ≠ {}}*]) *auto*

**lemma** *closure-involutive*:

$Closure_\alpha Closure_\alpha Z = Closure_\alpha Z$

**using** *closure-involutive-aux closure-involutive-aux'* **by** *metis*

**lemma** *closure-involutive'*:

$Z \subseteq Closure_\alpha W \implies Closure_\alpha Z \subseteq Closure_\alpha W$

**unfolding** *cla-def* **using** *valid-regions-distinct-spec* **by** *fast*

**lemma** *closure-subst*:

$Z \subseteq V \implies Z \subseteq Closure_\alpha Z$

**unfolding** *cla-def V-def* **using** *region-cover-spec* **by** *fast*

**lemma** *cla-mono'*:

$Z' \subseteq V \implies Z \subseteq Z' \implies Closure_\alpha Z \subseteq Closure_\alpha Z'$

**by** (*meson closure-involutive' closure-subst subset-trans*)

**lemma** *cla-mono*:

$Z \subseteq Z' \implies Closure_\alpha Z \subseteq Closure_\alpha Z'$

using *closure-V-int cla-mono*[of  $Z' \cap V Z \cap V$ ] by *auto*

## 5.9 A Zone Semantics Abstracting with $Closure_\alpha$

### 5.9.1 Single step

**inductive** *step-z-alpha* ::

$(\text{'a}, \text{'c}, t, \text{'s}) \text{ta} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{'a} \text{action} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{bool}$

$(\text{'-} \vdash \langle -, - \rangle \rightsquigarrow_{\alpha(-)} \langle -, - \rangle)$  [61,61,61] 61)

**where**

*step-alpha*:  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \Longrightarrow A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Closure_\alpha Z' \rangle$

**inductive-cases**[*elim!*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{\alpha(a)} \langle l', u' \rangle$

**declare** *step-z-alpha.intros*[*intro*]

**definition**

*step-z-alpha'* ::  $(\text{'a}, \text{'c}, t, \text{'s}) \text{ta} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{bool}$

$(\text{'-} \vdash \langle -, - \rangle \rightsquigarrow_\alpha \langle -, - \rangle)$  [61,61,61] 61)

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_\alpha \langle l', Z'' \rangle = (\exists Z' a. A \vdash \langle l, Z \rangle \rightsquigarrow_\tau \langle l, Z' \rangle \wedge A \vdash \langle l, Z' \rangle \rightsquigarrow_{\alpha(1a)} \langle l', Z'' \rangle)$

Single-step soundness and completeness follows trivially from *cla-empty-iff*.

**lemma** *step-z-alpha-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z' \rangle \Longrightarrow Z \subseteq V \Longrightarrow Z' \neq \{\} \Longrightarrow \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

**by** (*induction rule: step-z-alpha.induct*) (*auto dest: cla-empty-iff step-z-V*)

**lemma** *step-z-alpha'-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_\alpha \langle l', Z' \rangle \Longrightarrow Z \subseteq V \Longrightarrow Z' \neq \{\} \Longrightarrow \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

**oops**

**lemma** *step-z-alpha-complete'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \Longrightarrow Z \subseteq V \Longrightarrow \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$

**by** (*auto dest: closure-subs step-z-V*)

**lemma** *step-z-alpha-complete*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \Longrightarrow Z \subseteq V \Longrightarrow Z' \neq \{\} \Longrightarrow \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

by (blast dest: step-z-alpha-complete')

**lemma** *step-z-alpha'-complete'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies Z \subseteq V \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$

**unfolding** *step-z-alpha'-def step-z'-def* by (blast dest: step-z-alpha-complete' step-z-V)

**lemma** *step-z-alpha'-complete*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \neq \{\} \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

by (blast dest: step-z-alpha'-complete')

## 5.9.2 Multi step

**abbreviation**

$steps\text{-}z\text{-}\alpha :: ('a, 'c, t, 's) ta \Rightarrow 's \Rightarrow ('c, t) zone \Rightarrow 's \Rightarrow ('c, t) zone \Rightarrow bool$

$(\langle - \vdash \langle -, - \rangle \rightsquigarrow_{\alpha} * \langle -, - \rangle \rangle [61,61,61] 61)$

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} * \langle l', Z'' \rangle \equiv (\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle)** (l, Z) (l', Z'')$

P. Bouyer's calculation for  $Post (Closure_{\alpha} Z, e) \subseteq Closure_{\alpha} Post (Z, e)$

This is now obsolete as we argue solely with monotonicity of *steps-z* w.r.t  $Closure_{\alpha}$

**lemma** *calc*:

$valid\text{-}abstraction A X k \implies Z \subseteq V \implies A \vdash \langle l, Closure_{\alpha} Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$

**proof** (cases rule: step-z.cases, assumption, goal-cases)

**case 1**

**note**  $A = this$

**from**  $A(1)$  **have**  $\forall (x, m) \in clkp\text{-}set A. m \leq real (k x) \wedge x \in X \wedge m \in \mathbb{N}$

**by** (fastforce elim: valid-abstraction.cases)

**then have**  $\forall (x, m) \in collect\text{-}clock\text{-}pairs (inv\text{-}of A l). m \leq real (k x) \wedge x \in X \wedge m \in \mathbb{N}$

**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** auto

**from** *closure-constraint-id[OF this]* **have**  $*$ :  $Closure_{\alpha} \{inv\text{-}of A l\} = \{inv\text{-}of A l\} \cap V$ .

**have**  $(Closure_{\alpha} Z)^{\uparrow} \subseteq Closure_{\alpha} (Z^{\uparrow})$  **using**  $A(2)$  **by** (blast intro!: closure-delay-mono)

**then have**  $Z' \subseteq Closure_{\alpha} (Z^{\uparrow} \cap \{u. u \vdash inv\text{-}of A l\})$

**using** *closure-constraint-mono[OF \*, of Z<sup>↑</sup>]* **unfolding** *ccval-def* **by** (auto simp: Int-commute  $A(6)$ )

**with**  $A(4,3)$  **show** *?thesis* **by** (*auto elim!*: *step-z.cases*)  
**next**  
**case** ( $2\ g\ a\ r$ )  
**note**  $A = \text{this}$   
**from**  $A(1)$  **have** \*:  
 $\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k\ x) \wedge x \in X \wedge m \in \mathbb{N}$   
 $\text{collect-clkvt } (\text{trans-of } A) \subseteq X$   
 $\text{finite } X$   
**by** (*auto elim*: *valid-abstraction.cases*)  
**from**  $*(1)\ A(5)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A\ l'). m \leq \text{real } (k\ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** *fastforce*  
**from** *closure-constraint-id[OF this]* **have** \*\*:  $\text{Closure}_\alpha \{\{\text{inv-of } A\ l'\}\} = \{\{\text{inv-of } A\ l'\}\} \cap V$  .  
**from**  $*(1)\ A(6)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } g. m \leq \text{real } (k\ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clkt-def* **by** *fastforce*  
**from** *closure-constraint-id[OF this]* **have** \*\*\*:  $\text{Closure}_\alpha \{g\} = \{g\} \cap V$  .  
**from**  $*(2)\ A(6)$  **have** \*\*\*\*:  $\text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by** *fastforce*  
**from** *closure-constraint-mono'[OF \*\*\*, of Z]* **have**  
 $(\text{Closure}_\alpha Z) \cap \{u. u \vdash g\} \subseteq \text{Closure}_\alpha (Z \cap \{u. u \vdash g\})$  **unfolding** *cval-def*  
**by** (*subst Int-commute*) (*subst (asm) (2) Int-commute, assumption*)  
**moreover** **have**  $\text{zone-set } \dots r \subseteq \text{Closure}_\alpha (\text{zone-set } (Z \cap \{u. u \vdash g\}) r)$   
**using** \*\*\*\*  $A(2)$   
**by** (*intro closure-update-mono, auto*)  
**ultimately** **have**  $Z' \subseteq \text{Closure}_\alpha (\text{zone-set } (Z \cap \{u. u \vdash g\}) r \cap \{u. u \vdash \text{inv-of } A\ l'\})$   
**using** *closure-constraint-mono'[OF \*\*, of zone-set (Z \cap \{u. u \vdash g\}) r]*  
**unfolding** *cval-def*  
**apply** (*subst A(5)*)  
**apply** (*subst (asm) (5 7) Int-commute*)  
**apply** (*rule subset-trans*)  
**defer**  
**apply** *assumption*  
**apply** (*subst subset-int-mono*)  
**defer**  
**apply** *rule*  
**apply** (*rule subset-trans*)  
**defer**  
**apply** *assumption*  
**apply** (*rule zone-set-mono*)  
**apply** *assumption*

**done**  
**with**  $A(6)$  **show** *?thesis* **by** (*auto simp: A(4)*)  
**qed**

Turning P. Bouyers argument for multiple steps into an inductive proof is not direct. With this initial argument we can get to a point where the induction hypothesis is applicable. This breaks the "information hiding" induced by the different variants of steps.

**lemma** *steps-z-alpha-closure-involutive'-aux:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha W \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V$

$\implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle \wedge \text{Closure}_\alpha Z' \subseteq \text{Closure}_\alpha W'$

**proof** (*induction rule: step-z.induct*)

**case**  $A$ : (*step-t-z A l Z*)

**let**  $?Z' = Z^\uparrow \cap \{u. u \vdash \text{inv-of } A \ l\}$

**let**  $?W' = W^\uparrow \cap \{u. u \vdash \text{inv-of } A \ l\}$

**from**  $\mathcal{R}$ -*def* **have**  $\mathcal{R}$ -*def'*:  $\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$

**by** *simp*

**have** *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_\tau \langle l, ?W' \rangle$  **by** *auto*

**moreover** **have**  $\text{Closure}_\alpha ?Z' \subseteq \text{Closure}_\alpha ?W'$

**proof**

**fix**  $v$  **assume**  $v: v \in \text{Closure}_\alpha ?Z'$

**then obtain**  $R' \ v'$  **where**  $1: R' \in \mathcal{R} \ v \in R' \ v' \in R' \ v' \in ?Z'$  **unfolding**

*cla-def* **by** *auto*

**then obtain**  $u \ d$  **where**

$u \in Z$  **and**  $v': v' = u \oplus d \ u \oplus d \vdash \text{inv-of } A \ l \ 0 \leq d$

**unfolding** *zone-delay-def* **by** *blast*

**with** *closure-subs[OF A(3)] A(1)* **obtain**  $u' \ R$  **where**  $u': u' \in W \ u \in$

$R \ u' \in R \ R \in \mathcal{R}$

**unfolding** *cla-def* **by** *blast*

**then have**  $\forall x \in X. 0 \leq u \ x$  **unfolding**  $\mathcal{R}$ -*def* **by** *fastforce*

**from** *region-cover'[OF \mathcal{R}-def' this]* **have**  $R: [u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$  **by** *auto*

**from** *SuccI2[OF \mathcal{R}-def' this(2,1) \langle 0 \leq d \rangle, of [v']\_{\mathcal{R}} v'(1)]* **have**  $v'1:$

$[v']_{\mathcal{R}} \in \text{Succ } \mathcal{R} \ ([u]_{\mathcal{R}}) \ [v']_{\mathcal{R}} \in \mathcal{R}$

**by** *auto*

**from** *regions-closed'-spec[OF R(1,2) \langle 0 \leq d \rangle] v'(1)* **have**  $v'2: v' \in [v']_{\mathcal{R}}$

**by** *simp*

**from**  $A(2)$  **have**  $*$ :

$\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$

$\text{collect-ckvt } (\text{trans-of } A) \subseteq X$

*finite X*

**by** (*auto elim: valid-abstraction.cases*)

**from**  $*(1) \ u'(2)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l). m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$

**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** *fastforce*  
**from** *ccompatible*[*OF this, folded  $\mathcal{R}$ -def'*]  $v'1(2) v'2 v'(1,2)$  **have**  $3$ :  
 $[v']_{\mathcal{R}} \subseteq \{\text{inv-of } A \ l\}$   
**unfolding** *ccompatible-def ccval-def* **by** *auto*  
**with**  $A v'1 R(1) \mathcal{R}$ -*def'* **have**  $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l, ([v']_{\mathcal{R}}) \rangle$  **by** *auto*  
**with** *valid-regions-distinct-spec*[*OF  $v'1(2) 1(1) v'2 1(3)$* ] *region-unique-spec*[*OF  $u'(2,4)$* ]  
**have** *step-r*:  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l, R' \rangle$  **and**  $2$ :  $[v']_{\mathcal{R}} = R' [u]_{\mathcal{R}} = R$  **by** *auto*  
**from** *set-of-regions-spec*[*OF  $u'(4,3)$* ]  $v'1(1) 2$  **obtain**  $t$  **where**  $t: t \geq 0$   
 $[u' \oplus t]_{\mathcal{R}} = R'$  **by** *auto*  
**with** *regions-closed'-spec*[*OF  $u'(4,3)$  this(1)*] *step-t-r(1)* **have**  $*$ :  $u' \oplus t \in R'$  **by** *auto*  
**with**  $t(1) 3 2 u'(1,3)$  **have**  $A \vdash \langle l, u' \rangle \rightarrow \langle l, u' \oplus t \rangle u' \oplus t \in ?W'$   
**unfolding** *zone-delay-def ccval-def* **by** *auto*  
**with**  $* 1(1)$  **have**  $R' \subseteq \text{Closure}_{\alpha} ?W'$  **unfolding** *cla-def* **by** *auto*  
**with**  $1(2)$  **show**  $v \in \text{Closure}_{\alpha} ?W' ..$   
**qed**  
**ultimately show** *?case* **by** *auto*  
**next**  
**case**  $A$ : (*step-a-z A l g a r l' Z*)  
**let**  $?Z' = \text{zone-set} (Z \cap \{u. u \vdash g\}) r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**let**  $?W' = \text{zone-set} (W \cap \{u. u \vdash g\}) r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  $\mathcal{R}$ -*def* **have**  $\mathcal{R}$ -*def'*:  $\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$   
**by** *simp*  
**from**  $A(1)$  **have** *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_{1a} \langle l', ?W' \rangle$  **by** *auto*  
**moreover** **have**  $\text{Closure}_{\alpha} ?Z' \subseteq \text{Closure}_{\alpha} ?W'$   
**proof**  
**fix**  $v$  **assume**  $v: v \in \text{Closure}_{\alpha} ?Z'$   
**then obtain**  $R' v'$  **where**  $1: R' \in \mathcal{R} v \in R' v' \in R' v' \in ?Z'$  **unfolding** *cla-def* **by** *auto*  
**then obtain**  $u$  **where**  
 $u \in Z$  **and**  $v': v' = [r \rightarrow 0]u u \vdash g v' \vdash \text{inv-of } A \ l'$   
**unfolding** *zone-set-def* **by** *blast*  
**let**  $?R' = \text{region-set}' (([u]_{\mathcal{R}}) \cap \{u. u \vdash g\}) r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  $\langle u \in Z \rangle$  *closure-subs*[*OF  $A(4)$* ]  $A(2)$  **obtain**  $u' R$  **where**  $u': u' \in W u \in R u' \in R R \in \mathcal{R}$   
**unfolding** *cla-def* **by** *blast*  
**then have**  $\forall x \in X. 0 \leq u \ x$  **unfolding**  $\mathcal{R}$ -*def* **by** *fastforce*  
**from** *region-cover'*[*OF  $\mathcal{R}$ -def' this*] **have**  $R: [u]_{\mathcal{R}} \in \mathcal{R} u \in [u]_{\mathcal{R}}$  **by** *auto*  
**from** *step-r-complete-aux*[*OF  $\mathcal{R}$ -def'  $A(3)$  this(2,1)  $A(1) v'(2)$* ]  $v'$   
**have**  $*$ :  $[u]_{\mathcal{R}} = ([u]_{\mathcal{R}}) \cap \{u. u \vdash g\} ?R' = \text{region-set}' ([u]_{\mathcal{R}}) r \ 0 ?R' \in \mathcal{R}$  **by** *auto*  
**from**  $\mathcal{R}$ -*def'*  $A(3)$  **have** *collect-clkvt* (*trans-of*  $A$ )  $\subseteq X$  *finite*  $X$

by (*auto elim: valid-abstraction.cases*)  
 with  $A(1)$  have  $r$ : set  $r \subseteq X$  **unfolding** *collect-clkvt-def* by *fastforce*  
 from  $* v'(1)$   $R(2)$  have  $v' \in ?R'$  **unfolding** *region-set'-def* by *auto*  
 moreover have  $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l', ?R' \rangle$  **using**  $R(1)$   $\mathcal{R}$ -*def'*  $A(1,3)$   
 $v'(2)$  by *auto*  
**thm** *valid-regions-distinct-spec*  
 with *valid-regions-distinct-spec*[ $OF * (3)$   $1(1)$   $\langle v' \in ?R' \rangle$   $1(3)$ ] *re-*  
*gion-unique-spec*[ $OF u'(2,4)$ ]  
 have  $2$ :  $?R' = R' [u]_{\mathcal{R}} = R$  by *auto*  
 with  $* u'$  have  $*$ :  $[r \rightarrow 0]u' \in ?R'$   $u' \vdash g$   $[r \rightarrow 0]u' \vdash \text{inv-of } A$   $l'$   
**unfolding** *region-set'-def* by *auto*  
 with  $A(1)$  have  $A \vdash \langle l, u' \rangle \rightarrow \langle l', [r \rightarrow 0]u' \rangle$  **apply** (*intro step.intros(1)*)  
**apply rule** by *auto*  
 moreover from  $* u'(1)$  have  $[r \rightarrow 0]u' \in ?W'$  **unfolding** *zone-set-def*  
 by *auto*  
 ultimately have  $R' \subseteq \text{Closure}_{\alpha} ?W'$  **using**  $*(1)$   $1(1)$   $2(1)$  **unfolding**  
*cla-def* by *auto*  
 with  $1(2)$  **show**  $v \in \text{Closure}_{\alpha} ?W'$  ..  
**qed**  
 ultimately **show** *?case* by *meson*  
**qed**

**lemma** *steps-z-alpha-closure-involutive'-aux'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \text{Closure}_{\alpha} Z \subseteq \text{Closure}_{\alpha} W \implies \text{valid-abstraction}$   
 $A$   $X$   $k \implies Z \subseteq V \implies W \subseteq Z$   
 $\implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle \wedge \text{Closure}_{\alpha} Z' \subseteq \text{Closure}_{\alpha} W' \wedge W' \subseteq Z'$

**proof** (*induction rule: step-z.induct*)

**case**  $A$ : (*step-t-z*  $A$   $l$   $Z$ )

let  $?Z' = Z^{\uparrow} \cap \{u. u \vdash \text{inv-of } A$   $l\}$

let  $?W' = W^{\uparrow} \cap \{u. u \vdash \text{inv-of } A$   $l\}$

from  $\mathcal{R}$ -*def* have  $\mathcal{R}$ -*def'*:  $\mathcal{R} = \{\text{region } X$   $I$   $r \mid I$   $r. \text{valid-region } X$   $k$   $I$   $r\}$

by *simp*

have *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_{\tau} \langle l, ?W' \rangle$  by *auto*

moreover have  $\text{Closure}_{\alpha} ?Z' \subseteq \text{Closure}_{\alpha} ?W'$

**proof**

**fix**  $v$  **assume**  $v$ :  $v \in \text{Closure}_{\alpha} ?Z'$

**then obtain**  $R'$   $v'$  **where**  $1$ :  $R' \in \mathcal{R}$   $v \in R'$   $v' \in R'$   $v' \in ?Z'$  **unfolding**  
*cla-def* by *auto*

**then obtain**  $u$   $d$  **where**

$u \in Z$  **and**  $v'$ :  $v' = u \oplus d$   $u \oplus d \vdash \text{inv-of } A$   $l$   $0 \leq d$

**unfolding** *zone-delay-def* by *blast*

**with** *closure-subs*[ $OF A(3)$ ]  $A(1)$  **obtain**  $u'$   $R$  **where**  $u'$ :  $u' \in W$   $u \in R$   $u' \in R$   $R \in \mathcal{R}$

**unfolding** *cla-def* **by** *blast*  
**then have**  $\forall x \in X. 0 \leq u \ x$  **unfolding**  *$\mathcal{R}$ -def* **by** *fastforce*  
**from** *region-cover'*[*OF*  *$\mathcal{R}$ -def'* *this*] **have**  $R: [u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$  **by** *auto*  
**from** *SuccI2*[*OF*  *$\mathcal{R}$ -def'* *this*(2,1)  $\langle 0 \leq d \rangle$ , of  $[v']_{\mathcal{R}}$ ]  $v'(1)$  **have**  $v'1:$   
 $[v']_{\mathcal{R}} \in \text{Succ } \mathcal{R} \ ([u]_{\mathcal{R}}) \ [v']_{\mathcal{R}} \in \mathcal{R}$   
**by** *auto*  
**from** *regions-closed'-spec*[*OF*  $R(1,2) \langle 0 \leq d \rangle$ ]  $v'(1)$  **have**  $v'2: v' \in [v']_{\mathcal{R}}$   
**by** *simp*  
**from**  $A(2)$  **have**  $*$ :  
 $\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbf{N}$   
 $\text{collect-clkvt } (\text{trans-of } A) \subseteq X$   
 $\text{finite } X$   
**by** (*auto elim: valid-abstraction.cases*)  
**from**  $*(1) \ u'(2)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l). m \leq \text{real}$   
 $(k \ x) \wedge x \in X \wedge m \in \mathbf{N}$   
**unfolding** *clkp-set-def* *collect-clki-def* *inv-of-def* **by** *fastforce*  
**from** *compatible*[*OF* *this*, *folded*  *$\mathcal{R}$ -def'*]  $v'1(2) \ v'2 \ v'(1,2)$  **have**  $\exists:$   
 $[v']_{\mathcal{R}} \subseteq \{\text{inv-of } A \ l\}$   
**unfolding** *compatible-def* *ccval-def* **by** *auto*  
**with**  $A \ v'1 \ R(1) \ \mathcal{R}$ -*def'* **have**  $A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l, ([v']_{\mathcal{R}}) \rangle$  **by** *auto*  
**with** *valid-regions-distinct-spec*[*OF*  $v'1(2) \ 1(1) \ v'2 \ 1(3)$ ] *region-unique-spec*[*OF*  
 $u'(2,4)$ ]  
**have** *step-r*:  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l, R' \rangle$  **and**  $2: [v']_{\mathcal{R}} = R' \ [u]_{\mathcal{R}} = R$  **by**  
*auto*  
**from** *set-of-regions-spec*[*OF*  $u'(4,3)$ ]  $v'1(1) \ 2$  **obtain**  $t$  **where**  $t: t \geq 0$   
 $[u' \oplus t]_{\mathcal{R}} = R'$  **by** *auto*  
**with** *regions-closed'-spec*[*OF*  $u'(4,3) \ \text{this}(1)$ ] *step-t-r(1)* **have**  $*$ :  $u' \oplus t$   
 $\in R'$  **by** *auto*  
**with**  $t(1) \ 3 \ 2 \ u'(1,3)$  **have**  $A \vdash \langle l, u' \rangle \rightarrow \langle l, u' \oplus t \rangle \ u' \oplus t \in ?W'$   
**unfolding** *zone-delay-def* *ccval-def* **by** *auto*  
**with**  $* \ 1(1)$  **have**  $R' \subseteq \text{Closure}_{\alpha} \ ?W'$  **unfolding** *cla-def* **by** *auto*  
**with**  $1(2)$  **show**  $v \in \text{Closure}_{\alpha} \ ?W' \ ..$   
**qed**  
**moreover** **have**  $?W' \subseteq ?Z'$  **using**  $\langle W \subseteq Z \rangle$  **unfolding** *zone-delay-def*  
**by** *auto*  
**ultimately show** *?case* **by** *auto*  
**next**  
**case**  $A: (\text{step-a-z } A \ l \ g \ a \ r \ l' \ Z)$   
**let**  $?Z' = \text{zone-set } (Z \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**let**  $?W' = \text{zone-set } (W \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  *$\mathcal{R}$ -def* **have**  *$\mathcal{R}$ -def'*:  $\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$   
**by** *simp*  
**from**  $A(1)$  **have** *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_{1a} \langle l', ?W' \rangle$  **by** *auto*  
**moreover** **have**  $\text{Closure}_{\alpha} \ ?Z' \subseteq \text{Closure}_{\alpha} \ ?W'$

**proof**  
**fix**  $v$  **assume**  $v: v \in \text{Closure}_\alpha ?Z'$   
**then obtain**  $R' v'$  **where**  $R' \in \mathcal{R} v \in R' v' \in R' v' \in ?Z'$  **unfolding**  
*cla-def by auto*  
**then obtain**  $u$  **where**  
 $u \in Z$  **and**  $v': v' = [r \rightarrow 0]u$   $u \vdash g$   $v' \vdash \text{inv-of } A \ l'$   
**unfolding** *zone-set-def by blast*  
**let**  $?R' = \text{region-set}' ([u]_{\mathcal{R}}) \cap \{u. u \vdash g\}$   $r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  $\langle u \in Z \rangle \text{closure-subs}[OF \ A(4)] \ A(2)$  **obtain**  $u' R$  **where**  $u': u' \in$   
 $W \ u \in R \ u' \in R \ R \in \mathcal{R}$   
**unfolding** *cla-def by blast*  
**then have**  $\forall x \in X. 0 \leq u \ x$  **unfolding**  $\mathcal{R}$ -*def by fastforce*  
**from** *region-cover'*[ $OF \ \mathcal{R}$ -*def' this*] **have**  $[u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$  **by auto**  
**have** \*:  
 $[u]_{\mathcal{R}} = ([u]_{\mathcal{R}}) \cap \{u. u \vdash g\}$   
 $\text{region-set}' ([u]_{\mathcal{R}}) \ r \ 0 \subseteq [[r \rightarrow 0]u]_{\mathcal{R}} \ [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R}$   
 $([[r \rightarrow 0]u]_{\mathcal{R}}) \cap \{u. u \vdash \text{inv-of } A \ l'\} = [[r \rightarrow 0]u]_{\mathcal{R}}$   
**proof** –  
**from**  $A(3)$  **have** *collect-ckvt (trans-of A)  $\subseteq X$*   
**by** (*auto elim: valid-abstraction.cases*)  
**with**  $A(1)$  **have** *set  $r \subseteq X \ \forall y. y \notin \text{set } r \longrightarrow k \ y \leq k \ y$*   
**unfolding** *collect-ckvt-def by fastforce+*  
**with**  
 $\text{region-set-subs}[of \ - \ X \ k \ - \ 0, \ \text{where } k' = k, \ \text{folded } \mathcal{R}\text{-def}, \ OF \ \langle [u]_{\mathcal{R}}$   
 $\in \mathcal{R} \rangle \ \langle u \in [u]_{\mathcal{R}} \rangle \ \text{finite}]$   
**show**  $\text{region-set}' ([u]_{\mathcal{R}}) \ r \ 0 \subseteq [[r \rightarrow 0]u]_{\mathcal{R}} \ [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R}$  **by auto**  
**from**  $A(3)$  **have** \*:  
 $\forall (x, m) \in \text{clkp-set } A. m \leq \text{real } (k \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**by** (*fastforce elim: valid-abstraction.cases*)  
**from** \*  $A(1)$  **have** \*\*\*:  $\forall (x, m) \in \text{collect-clock-pairs } g. m \leq \text{real } (k \ x)$   
 $\wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clkt-def by fastforce*  
**from**  $\langle u \in [u]_{\mathcal{R}} \rangle \ \langle [u]_{\mathcal{R}} \in \mathcal{R} \rangle \ \text{ccompatible}[OF \ \text{this}, \ \text{folded } \mathcal{R}\text{-def}] \ \langle u \vdash$   
 $g \rangle$  **show**  
 $[u]_{\mathcal{R}} = ([u]_{\mathcal{R}}) \cap \{u. u \vdash g\}$   
**unfolding** *ccompatible-def cval-def by blast*  
**have** \*\*:  $[r \rightarrow 0]u \in [[r \rightarrow 0]u]_{\mathcal{R}}$   
**using**  $\langle R' \in \mathcal{R} \rangle \ \langle v' \in R' \rangle \ \text{region-unique-spec } v'(1)$  **by blast**  
**from** \* **have**  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l'). m \leq \text{real } (k \ x) \wedge x \in X \wedge$   
 $m \in \mathbb{N}$   
**unfolding** *inv-of-def clkp-set-def collect-clki-def by fastforce*  
**from** \*\*  $\langle [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R} \rangle \ \text{ccompatible}[OF \ \text{this}, \ \text{folded } \mathcal{R}\text{-def}] \ \langle v' \vdash \rightarrow$   
**show**

$$([[r \rightarrow 0]u]_{\mathcal{R}}) \cap \{u. u \vdash \text{inv-of } A \ l'\} = [[r \rightarrow 0]u]_{\mathcal{R}}$$
**unfolding** *ccompatible-def cval-def*  $\langle v' = - \rangle$  **by** *blast*  
**qed**  
**from**  $*$   $\langle v' = - \rangle \langle u \in [u]_{\mathcal{R}} \rangle$  **have**  $v' \in [[r \rightarrow 0]u]_{\mathcal{R}}$  **unfolding** *region-set'-def* **by** *auto*  
**from** *valid-regions-distinct-spec*[*OF*  $*$ (3)  $\langle R' \in \mathcal{R} \rangle \langle v' \in [[r \rightarrow 0]u]_{\mathcal{R}} \rangle \langle v' \in R' \rangle$ ]  
**have**  $[[r \rightarrow 0]u]_{\mathcal{R}} = R'$  .  
**from** *region-unique-spec*[*OF*  $u'(2,4)$ ] **have**  $[u]_{\mathcal{R}} = R$  **by** *auto*  
**from**  $\langle [u]_{\mathcal{R}} = R \rangle * (1,2) * (4) \langle u' \in R \rangle$  **have**  
 $[r \rightarrow 0]u' \in [[r \rightarrow 0]u]_{\mathcal{R}}$   $u' \vdash g$   $[r \rightarrow 0]u' \vdash \text{inv-of } A \ l'$   
**unfolding** *region-set'-def* **by** *auto*  
**with**  $u'(1)$  **have**  $[r \rightarrow 0]u' \in ?W'$  **unfolding** *zone-set-def* **by** *auto*  
**with**  $\langle [r \rightarrow 0]u' \in [[r \rightarrow 0]u]_{\mathcal{R}} \rangle \langle [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R} \rangle$  **have**  $[[r \rightarrow 0]u]_{\mathcal{R}} \subseteq \text{Closure}_{\alpha} ?W'$   
**unfolding** *cla-def* **by** *auto*  
**with**  $\langle v \in R' \rangle$  **show**  $v \in \text{Closure}_{\alpha} ?W'$  **unfolding**  $\langle - = R' \rangle$  ..  
**qed**  
**moreover** **have**  $?W' \subseteq ?Z'$  **using**  $\langle W \subseteq Z \rangle$  **unfolding** *zone-set-def* **by** *auto*  
**ultimately show** *?case* **by** *meson*  
**qed**

**lemma** *steps-z-alpha-V*:  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} * \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$   
**by** (*induction rule: rtranclp-induct2*)  
*(use closure-V in*  $\langle \text{auto dest: step-z-V simp: step-z-alpha'-def} \rangle$ *)*

**lemma** *steps-z-alpha-closure-involutive'*:  
 $A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} * \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow_{\tau} \langle l', Z'' \rangle \implies A \vdash \langle l', Z'' \rangle \rightsquigarrow_{1a} \langle l'', Z''' \rangle$   
 $\implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V$   
 $\implies \exists W''' . A \vdash \langle l, Z \rangle \rightsquigarrow * \langle l'', W''' \rangle \wedge \text{Closure}_{\alpha} Z''' \subseteq \text{Closure}_{\alpha} W''' \wedge W''' \subseteq Z'''$

**proof** (*induction arbitrary: a*  $Z'' Z''' l''$  *rule: rtranclp-induct2*)

**case** *refl* **then show** *?case* **unfolding** *step-z'-def* **by** *blast*

**next**

**case** *A*: (*step*  $l' Z' l''1 Z''1$ )

**from** *A*(2) **obtain**  $Z'1 \ \mathcal{Z} \ a'$  **where**  $Z''1$ :

$Z''1 = \text{Closure}_{\alpha} \ \mathcal{Z} \ A \vdash \langle l', Z' \rangle \rightsquigarrow_{\tau} \langle l', Z'1 \rangle \ A \vdash \langle l', Z'1 \rangle \rightsquigarrow_{1a'} \langle l''1, \mathcal{Z} \rangle$

**unfolding** *step-z-alpha'-def* **by** *auto*

**from** *A*(3)[*OF this*(2,3) *A*(6,7)] **obtain**  $W'''$  **where**  $W'''$ :

$A \vdash \langle l, Z \rangle \rightsquigarrow * \langle l''1, W''' \rangle \ \text{Closure}_{\alpha} \ \mathcal{Z} \subseteq \text{Closure}_{\alpha} \ W''' \ W''' \subseteq \mathcal{Z}$

**by** *auto*

**have**  $Z'' \subseteq V$   
**by** (*metis*  $A(4)$ )  $Z''1(1)$  *closure-V step-z-V*  
**have**  $Z \subseteq V$   
**by** (*meson*  $A$   $Z''1$  *step-z-V steps-z-alpha-V*)  
**from** *closure-subs[OF this]*  $\langle W''' \subseteq Z \rangle$  **have**  $*$ :  $W''' \subseteq \text{Closure}_\alpha Z$  **by**  
*auto*  
**from**  $A(4)$   $\langle Z''1 = - \rangle$  **have**  $A \vdash \langle l''1, \text{Closure}_\alpha Z \rangle \rightsquigarrow_\tau \langle l''1, Z'' \rangle$  **by** *simp*  
**from** *steps-z-alpha-closure-involutive'-aux'[OF this - A(6) closure-V \*]*  
 $W'''(2)$  **obtain**  $W'$   
**where**  $***$ :  $A \vdash \langle l''1, W''' \rangle \rightsquigarrow_\tau \langle l''1, W' \rangle$   $\text{Closure}_\alpha Z'' \subseteq \text{Closure}_\alpha W'$   
 $W' \subseteq Z''$   
**by** *atomize-elim (auto simp: closure-involutive)*

This shows how we could easily add more steps before doing the final closure operation!

**from** *steps-z-alpha-closure-involutive'-aux'[OF A(5) this(2) A(6)  $\langle Z'' \subseteq V \rangle$  this(3)]* **obtain**  $W''$   
**where**  
 $A \vdash \langle l''1, W' \rangle \rightsquigarrow_{1a} \langle l'', W'' \rangle$   $\text{Closure}_\alpha Z''' \subseteq \text{Closure}_\alpha W''$   $W'' \subseteq Z'''$   
**by** *auto*  
**with**  $***$   $W'''$  **show** *?case*  
**unfolding** *step-z'-def* **by** (*blast intro: rtranclp.rtrancl-into-rtrancl*)  
**qed**

**lemma** *steps-z-alpha-closure-involutive*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha*} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V$   
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z'' \rangle \wedge \text{Closure}_\alpha Z' \subseteq \text{Closure}_\alpha Z'' \wedge Z'' \subseteq Z'$

**proof** (*induction rule: rtranclp-induct2*)

**case** *refl* **show** *?case* **by** *blast*

**next**

**case** *2*: (*step*  $l'$   $Z'$   $l''$   $Z'''$ )

**then obtain**  $Z''$  *a*  $Z''1$  **where**  $*$ :

$A \vdash \langle l', Z' \rangle \rightsquigarrow_\tau \langle l', Z'' \rangle$   $A \vdash \langle l', Z'' \rangle \rightsquigarrow_{1a} \langle l'', Z''1 \rangle$   $Z''' = \text{Closure}_\alpha Z''1$

**unfolding** *step-z-alpha'-def* **by** *auto*

**from** *steps-z-alpha-closure-involutive'[OF 2(1) this(1,2) 2(4,5)]* **obtain**  
 $W'''$  **where**  $W'''$ :

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l'', W''' \rangle$   $\text{Closure}_\alpha Z''1 \subseteq \text{Closure}_\alpha W'''$   $W''' \subseteq Z''1$  **by**  
*blast*

**have**  $W''' \subseteq Z'''$

**unfolding**  $*$

**by** (*rule order-trans[OF  $\langle W''' \subseteq Z''1 \rangle$ ]* *closure-subs step-z-V steps-z-alpha-V*)

\* 2(1,5))+  
**with** \* *closure-involutive*  $W'''$  **show** ?*case by auto*  
**qed**

**lemma** *steps-z-V*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$

**unfolding** *step-z'-def* **by** (*induction rule: rtranclp-induct2*) (*auto dest!:*  
*step-z-V*)

**lemma** *steps-z-alpha-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies Z' \neq \{\}$   
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z'' \rangle \wedge Z'' \neq \{\} \wedge Z'' \subseteq Z'$

**proof** *goal-cases*

**case** 1

**from** *steps-z-alpha-closure-involutive*[*OF* 1(1-3)] **obtain**  $Z''$  **where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z'' \rangle \text{ Closure}_{\alpha} Z' \subseteq \text{Closure}_{\alpha} Z'' \ Z'' \subseteq Z'$

**by** *blast*

**moreover with** 1(4) *cla-empty-iff*[*OF* *steps-z-alpha-V*[*OF* 1(1)], *OF*  
1(3)]

*cla-empty-iff*[*OF* *steps-z-V*, *OF* *this*(1) 1(3)] **have**  $Z'' \neq \{\}$  **by** *auto*

**ultimately show** ?*case by auto*

**qed**

**lemma** *step-z-alpha-mono*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle$   
 $\rightsquigarrow_{\alpha(a)} \langle l', W' \rangle \wedge Z' \subseteq W'$

**proof** *goal-cases*

**case** 1

**then obtain**  $Z''$  **where** \*:  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z'' \rangle \ Z' = \text{Closure}_{\alpha} Z''$  **by**  
*auto*

**from** *step-z-mono*[*OF* *this*(1) 1(2)] **obtain**  $W'$  **where**  $A \vdash \langle l, W \rangle \rightsquigarrow_a$   
 $\langle l', W' \rangle \ Z'' \subseteq W'$  **by** *auto*

**moreover with** \*(2) **have**  $Z' \subseteq \text{Closure}_{\alpha} W'$  **unfolding** *cla-def* **by** *auto*  
**ultimately show** ?*case by blast*

**qed**

**end**

## 5.10 New Variant

**New Definitions** **hide-const** *collect-clkt collect-clki clkp-set valid-abstraction*

**definition** *collect-clkt* :: ('a, 'c, 't, 's) transition set  $\Rightarrow$  's  $\Rightarrow$  ('c \*'t) set  
**where**  
*collect-clkt* S l =  $\bigcup$  {*collect-clock-pairs* (fst (snd t)) | t . t  $\in$  S  $\wedge$  fst t = l}

**definition** *collect-clki* :: ('c, 't, 's) invassn  $\Rightarrow$  's  $\Rightarrow$  ('c \*'t) set  
**where**  
*collect-clki* I s = *collect-clock-pairs* (I s)

**definition** *clkp-set* :: ('a, 'c, 't, 's) ta  $\Rightarrow$  's  $\Rightarrow$  ('c \*'t) set  
**where**  
*clkp-set* A s = *collect-clki* (inv-of A) s  $\cup$  *collect-clkt* (trans-of A) s

**lemma** *collect-clkt-alt-def*:  
*collect-clkt* S l =  $\bigcup$  (*collect-clock-pairs* ' (fst o snd) ' {t. t  $\in$  S  $\wedge$  fst t = l})  
**unfolding** *collect-clkt-def* by *fastforce*

**inductive** *valid-abstraction*  
**where**  
 $\llbracket \forall l. \forall (x,m) \in \text{clkp-set } A \ l. \ m \leq k \ l \ x \wedge x \in X \wedge m \in \mathbb{N}; \text{collect-clkvt}$   
 $(\text{trans-of } A) \subseteq X; \text{finite } X;$   
 $\forall l \ g \ a \ r \ l' \ c. \ A \vdash l \xrightarrow{g,a,r} l' \wedge c \notin \text{set } r \longrightarrow k \ l' \ c \leq k \ l \ c$   
 $\rrbracket$   
 $\implies \text{valid-abstraction } A \ X \ k$

**locale** *AlphaClosure* =  
*Alpha-defs* X for X :: 'c set +  
**fixes** k :: 's  $\Rightarrow$  'c  $\Rightarrow$  nat **and**  $\mathcal{R}$   
**defines**  $\mathcal{R} \ l \equiv \{\text{region } X \ I \ r \mid I \ r. \ \text{valid-region } X \ (k \ l) \ I \ r\}$   
**assumes** *finite*: finite X  
**begin**

## 5.11 A Semantics Based on Localized Regions

### 5.11.1 Single step

**inductive** *step-r* ::  
('a, 'c, t, 's) ta  $\Rightarrow$  -  $\Rightarrow$  's  $\Rightarrow$  ('c, t) zone  $\Rightarrow$  'a action  $\Rightarrow$  's  $\Rightarrow$  ('c, t) zone  
 $\Rightarrow$  bool  
( $\langle -, - \vdash \langle -, - \rangle \rightsquigarrow_- \langle -, - \rangle$ ) [61,61,61,61,61] 61)  
**where**  
*step-t-r*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{\tau} \langle l, R^{\wedge} \rangle$  **if**  
*valid-abstraction*  $A \ X \ (\lambda x. \text{real } o \ k \ x) \ R \in \mathcal{R} \ l \ R' \in \text{Succ} (\mathcal{R} \ l) \ R \ R' \subseteq$   
 $\{\!\{ \text{inv-of } A \ l \}\!\}$  |

*step-a-r:*

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{|a} \langle l', R^{\wedge} \rangle$  **if**  
*valid-abstraction*  $A \ X \ (\lambda x. \text{real } o \ k \ x) \ A \vdash l \longrightarrow^{g, a, r} l' \ R \in \mathcal{R} \ l$   
 $R \subseteq \{\!\{ g \}\!\}$  *region-set'*  $R \ r \ 0 \subseteq R' \ R' \subseteq \{\!\{ \text{inv-of } A \ l' \}\!\} \ R' \in \mathcal{R} \ l'$

**inductive-cases**[*elim!*]:  $A, \mathcal{R} \vdash \langle l, u \rangle \rightsquigarrow_a \langle l', u^{\wedge} \rangle$

**declare** *step-r.intros*[*intro*]

**inductive** *step-r'* ::

$(\ 'a, \ 'c, \ t, \ 's) \ \text{ta} \Rightarrow - \Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ \text{zone} \Rightarrow \ 'a \Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ \text{zone} \Rightarrow \text{bool}$   
 $(\langle -, - \rangle \rightsquigarrow - \langle -, - \rangle) \ [61, 61, 61, 61, 61] \ 61)$

**where**

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R'^{\wedge} \rangle$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{\tau} \langle l, R^{\wedge} \rangle \ A, \mathcal{R} \vdash \langle l, R^{\wedge} \rangle \rightsquigarrow_{|a} \langle l', R'^{\wedge} \rangle$

**lemmas**  $\mathcal{R}\text{-def}' = \text{meta-eq-to-obj-eq}[OF \ \mathcal{R}\text{-def}]$

**lemmas**  $\text{region-cover}' = \text{region-cover}'[OF \ \mathcal{R}\text{-def}']$

**abbreviation**  $\text{part}'' (\langle [-] \rangle \ [61, 61] \ 61)$  **where**  $\text{part}'' \ u \ l1 \equiv \text{part } u \ (\mathcal{R} \ l1)$

**no-notation**  $\text{part} (\langle [-] \rangle \ [61, 61] \ 61)$

**lemma** *step-r-complete-aux:*

**fixes**  $R \ u \ r \ A \ l' \ g$

**defines**  $R' \equiv [[r \rightarrow 0]u]_{l'}$

**assumes** *valid-abstraction*  $A \ X \ (\lambda x. \text{real } o \ k \ x)$

**and**  $u \in R$

**and**  $R \in \mathcal{R} \ l$

**and**  $A \vdash l \longrightarrow^{g, a, r} l'$

**and**  $u \vdash g$

**and**  $[r \rightarrow 0]u \vdash \text{inv-of } A \ l'$

**shows**  $R = R \cap \{u. u \vdash g\} \wedge \text{region-set}' \ R \ r \ 0 \subseteq R' \wedge R' \in \mathcal{R} \ l' \wedge R' \subseteq$   
 $\{\!\{ \text{inv-of } A \ l' \}\!\}$

**proof** –

**note**  $A = \text{assms}(2-)$

**from**  $A(1)$  **obtain**  $a1 \ b1$  **where** \*:

$A = (a1, b1)$

$\forall l. \forall x \in \text{clkp-set} \ (a1, b1) \ l. \text{case } x \text{ of } (x, m) \Rightarrow m \leq \text{real} \ (k \ l \ x) \wedge x \in$

$X \wedge m \in \mathbb{N}$

$\text{collect-clkvt} \ (\text{trans-of} \ (a1, b1)) \subseteq X$

*finite*  $X$

$\forall l g a r l' c. (a1, b1) \vdash l \longrightarrow^{g,a,r} l' \wedge c \notin \text{set } r \longrightarrow k l' c \leq k l c$   
**by** (*clarsimp elim!:* *valid-abstraction.cases*)  
**from**  $A(4) * (1,3)$  **have**  $r: \text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by**  
*fastforce*  
**from**  $A(4) * (1,5)$  **have** *ceiling-mono*:  $\forall y. y \notin \text{set } r \longrightarrow k l' y \leq k l y$  **by**  
*auto*  
**from**  $A(4) * (1,2)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } g. m \leq \text{real } (k l x) \wedge$   
 $x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clkt-def* **by** *fastforce*  
**from** *ccompatible[OF this, folded  $\mathcal{R}$ -def]*  $A(2,3,5)$  **have**  $R \subseteq \{g\}$   
**unfolding** *ccompatible-def ccval-def* **by** *blast*  
**then have**  $R\text{-id}: R \cap \{u. u \vdash g\} = R$  **unfolding** *ccval-def* **by** *auto*  
**from**  
*region-set-subs[OF A(3)[unfolded  $\mathcal{R}$ -def] A(2)  $\langle \text{finite } X \rangle$  -  $r$  ceiling-mono,*  
*of 0, folded  $\mathcal{R}$ -def]*  
**have** \*\*:  
 $[[r \rightarrow 0]u]_{l'} \supseteq \text{region-set}' R r 0 [[r \rightarrow 0]u]_{l'} \in \mathcal{R} l' [r \rightarrow 0]u \in [[r \rightarrow 0]u]_{l'}$   
**by** *auto*  
**let**  $?R = [[r \rightarrow 0]u]_{l'}$   
**from**  $* (1,2)$  **have** \*\*\*:  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A l'). m \leq \text{real } (k l' x) \wedge x \in X \wedge$   
 $m \in \mathbb{N}$   
**unfolding** *inv-of-def clkp-set-def collect-clki-def* **by** *fastforce*  
**from** *ccompatible[OF this, folded  $\mathcal{R}$ -def]*  $** (2-)$   $A(6)$  **have**  $?R \subseteq \{\text{inv-of}$   
 $A l'\}$   
**unfolding** *ccompatible-def ccval-def* **by** *blast*  
**then have** \*\*\*:  $?R \cap \{u. u \vdash \text{inv-of } A l'\} = ?R$  **unfolding** *ccval-def* **by**  
*auto*  
**with**  $** (1,2)$   $R\text{-id} \langle ?R \subseteq \rightarrow \rangle$  **show** *?thesis* **by** (*auto simp: R'-def*)  
**qed**

**lemma** *step-t-r-complete*:

**assumes**  
 $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$  *valid-abstraction*  $A X (\lambda x. \text{real } o k x) \forall x \in X. u$   
 $x \geq 0$   
**shows**  $\exists R'. A, \mathcal{R} \vdash \langle l, ([u]_l) \rangle \rightsquigarrow_{\tau} \langle l', R' \rangle \wedge u' \in R' \wedge R' \in \mathcal{R} l'$   
**using** *assms(1)* **proof** (*cases*)  
**case**  $A: 1$   
**hence**  $u'$ :  $u' = (u \oplus d) u \oplus d \vdash \text{inv-of } A l 0 \leq d$  **and**  $l = l'$  **by** *auto*  
**from** *region-cover'[OF assms(3)]* **have**  $R: [u]_l \in \mathcal{R} l u \in [u]_l$  **by** *auto*  
**from** *SuccI2[OF  $\mathcal{R}$ -def' this(2,1)  $\langle 0 \leq d \rangle$ , of  $[u]_l$   $u'(1)$ ]* **have**  $u'1$ :  
 $[u']_l \in \text{Succ } (\mathcal{R} l) ([u]_l) [u]_l \in \mathcal{R} l$   
**by** *auto*  
**from** *regions-closed'[OF  $\mathcal{R}$ -def' R  $\langle 0 \leq d \rangle$ ]*  $u'(1)$  **have**  $u'2: u' \in [u]_l$  **by**

*simp*

**from** *assms(2)* **obtain** *a1 b1* **where**  
 $A = (a1, b1)$   
 $\forall l. \forall x \in \text{clkp-set } (a1, b1) l. \text{ case } x \text{ of } (x, m) \Rightarrow m \leq \text{real } (k l x) \wedge x \in X \wedge m \in \mathbb{N}$   
 $\text{collect-clkvt } (\text{trans-of } (a1, b1)) \subseteq X$   
*finite*  $X$   
 $\forall l g a r l' c. (a1, b1) \vdash l \xrightarrow{g,a,r} l' \wedge c \notin \text{set } r \longrightarrow k l' c \leq k l c$   
**by** (*clarsimp elim!: valid-abstraction.cases*)  
**note**  $*$  = *this*  
**from**  $*(1,2)$   $u'(2)$  **have**  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A l). m \leq \text{real } (k l x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** *fastforce*  
**from** *ccompatible[OF this, folded  $\mathcal{R}$ -def]*  $u'1(2)$   $u'2$   $u'(1,2)$  **have**  $[u']_l \subseteq \{\text{inv-of } A l\}$   
**unfolding** *ccompatible-def cval-def* **by** *auto*  
**with**  $u'1 R(1)$  *assms* **have**  $A, \mathcal{R} \vdash \langle l, ([u]_l) \rangle \rightsquigarrow_{\tau} \langle l, ([u']_l) \rangle$  **by** *auto*  
**with**  $u'1(2)$   $u'2$   $\langle l = l' \rangle$  **show** *?thesis* **by** *meson*  
**qed**

**lemma** *step-a-r-complete:*

**assumes**  
 $A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$  *valid-abstraction*  $A X (\lambda x. \text{real } o k x) \forall x \in X. u x \geq 0$   
**shows**  $\exists R'. A, \mathcal{R} \vdash \langle l, ([u]_l) \rangle \rightsquigarrow_{1a} \langle l', R' \rangle \wedge u' \in R' \wedge R' \in \mathcal{R} l'$   
**using** *assms(1)* **proof** *cases*  
**case**  $A: (1 g r)$   
**then obtain**  $g r$  **where**  $u': u' = [r \rightarrow 0]u$   $A \vdash l \xrightarrow{g,a,r} l' u \vdash g u' \vdash \text{inv-of } A l'$   
**by** *auto*  
**let**  $?R' = [[r \rightarrow 0]u]_{l'}$   
**from** *region-cover'[OF assms(3)]* **have**  $R: [u]_l \in \mathcal{R} l u \in [u]_l$  **by** *auto*  
**from** *step-r-complete-aux[OF assms(2) this(2,1) u'(2,3)]*  $u'$  **have**  $*$ :  
 $[u]_l \subseteq \{\!|g|\!\} ?R' \supseteq \text{region-set}' ([u]_l) r 0 ?R' \in \mathcal{R} l' ?R' \subseteq \{\!|\text{inv-of } A l'|\!\}$   
**by** (*auto simp: cval-def*)  
**from** *assms(2,3)* **have**  $\text{collect-clkvt } (\text{trans-of } A) \subseteq X$  *finite*  $X$   
**by** (*auto elim: valid-abstraction.cases*)  
**with**  $u'(2)$  **have**  $r: \text{set } r \subseteq X$  **unfolding** *collect-clkvt-def* **by** *fastforce*  
**from**  $*$   $u'(1)$   $R(2)$  **have**  $u' \in ?R'$  **unfolding** *region-set'-def* **by** *auto*  
**moreover** **have**  $A, \mathcal{R} \vdash \langle l, ([u]_l) \rangle \rightsquigarrow_{1a} \langle l', ?R' \rangle$  **using**  $R(1)$   $u'(2)$   $*$  *assms(2,3)*  
**by** (*auto 4 3*)  
**ultimately show** *?thesis* **using**  $*(3)$  **by** *meson*  
**qed**

**lemma** *step-r-complete*:

**assumes**

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$  *valid-abstraction*  $A \ X \ (\lambda x. \text{real } o \ k \ x) \ \forall x \in X. \ u \ x \geq 0$

**shows**  $\exists R' a. A, \mathcal{R} \vdash \langle l, ([u]_i) \rangle \rightsquigarrow_a \langle l', R' \rangle \wedge u' \in R' \wedge R' \in \mathcal{R} \ l'$

**using** *assms* **by cases** (*drule step-a-r-complete step-t-r-complete; auto*)<sup>+</sup>

Compare this to lemma *step-z-sound*. This version is weaker because for regions we may very well arrive at a successor for which not every valuation can be reached by the predecessor. This is the case for e.g. the region with only Greater (k x) bounds.

**lemma** *step-t-r-sound*:

**assumes**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_\tau \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. \exists d \geq 0. A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$

**using** *assms(1)* **proof cases**

**case** *A: step-t-r*

**show** *?thesis*

**proof**

**fix** *u* **assume**  $u \in R$

**from** *set-of-regions*[*OF*  $A(3)$ ][*unfolded*  $\mathcal{R}$ -*def*], *folded*  $\mathcal{R}$ -*def*, *OF* *this*  $A(4)$ ]  $A(2)$

**obtain** *t* **where**  $t \geq 0 \ [u \oplus t]_l = R'$  **by** (*auto elim: valid-abstraction.cases*)

**with** *regions-closed*'[*OF*  $\mathcal{R}$ -*def'*  $A(3)$   $\langle u \in R \rangle$  *this(1)*] *step-t-r(1)* **have**  $(u \oplus t) \in R'$  **by** *auto*

**with**  $t(1)$   $A(5)$  **have**  $A \vdash \langle l, u \rangle \rightarrow^t \langle l, (u \oplus t) \rangle$  **unfolding** *ccval-def* **by** *auto*

**with**  $\langle - \in R' \rangle \langle l' = l \rangle$  **show**  $\exists u' \in R'. \exists t \geq 0. A \vdash \langle l, u \rangle \rightarrow^t \langle l', u' \rangle$

**by** *meson*

**qed**

**qed**

**lemma** *step-a-r-sound*:

**assumes**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{1a} \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$

**using** *assms* **proof cases**

**case** *A: (step-a-r g r)*

**show** *?thesis*

**proof**

**fix** *u* **assume**  $u \in R$

**from**  $\langle u \in R \rangle$   $A(4-6)$  **have**  $u \vdash g \ [r \rightarrow 0]u \vdash \text{inv-of } A \ l' \ [r \rightarrow 0]u \in R'$

**unfolding** *region-set'-def ccval-def* **by** *auto*

**with**  $A(2)$  **have**  $A \vdash \langle l, u \rangle \rightarrow_a \langle l', [r \rightarrow 0]u \rangle$  **by** (*blast intro: step-a.intros*)

**with**  $\langle - \in R' \rangle$  **show**  $\exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$  **by** *meson*

qed  
qed

**lemma** *step-r-sound*:

**assumes**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$

**using** *assms*

**by** (*cases a; simp*) (*drule step-a-r-sound step-t-r-sound; fastforce*)+

**lemma** *step-r'-sound*:

**assumes**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$

**using** *assms* **by** *cases* (*blast dest!: step-a-r-sound step-t-r-sound*)

## 5.12 A New Zone Semantics Abstracting with $Closure_{\alpha, l}$

**definition** *cla* ( $\langle Closure_{\alpha, -}(-) \rangle$  [71, 71] 71)

**where**

*cla*  $l Z = \bigcup \{R \in \mathcal{R} \mid R \cap Z \neq \{\}\}$

### 5.12.1 Single step

**inductive** *step-z-alpha* ::

$(\prime a, \prime c, t, \prime s) ta \Rightarrow \prime s \Rightarrow (\prime c, t) zone \Rightarrow \prime a \text{ action} \Rightarrow \prime s \Rightarrow (\prime c, t) zone \Rightarrow$   
*bool*

$(\langle - \vdash \langle -, - \rangle \rightsquigarrow_{\alpha(-)} \langle -, - \rangle \rangle$  [61, 61, 61] 61)

**where**

*step-alpha*:  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \Longrightarrow A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Closure_{\alpha, l'} Z' \rangle$

**inductive-cases**[*elim!*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{\alpha(a)} \langle l', u' \rangle$

**declare** *step-z-alpha.intros*[*intro*]

Single-step soundness and completeness follows trivially from *cla-empty-iff*.

**lemma** *step-z-alpha-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z' \rangle \Longrightarrow Z \subseteq V \Longrightarrow Z' \neq \{\}$

$\Longrightarrow \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \wedge Z'' \neq \{\}$

**apply** (*induction rule: step-z-alpha.induct*)

**apply** (*frule step-z-V*)

**apply** *assumption*

**apply** (*rotate-tac 3*)

**by** (*fastforce simp: cla-def*)

**context**

fixes  $l' :: 's$

**begin**

**interpretation** *alpha*: *AlphaClosure-global - k l' R l' by standard (rule finite)*

**lemma** [*simp*]:

*alpha.cla* = *cla l'*

**unfolding** *cla-def alpha.cla-def ..*

**lemma** *step-z-alpha-complete*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \neq \{\}$   
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

**apply** (*frule step-z-V*)

**apply** *assumption*

**apply** (*rotate-tac 3*)

**apply** (*drule alpha.cla-empty-iff*)

**by** *auto*

**end**

### 5.12.2 Multi step

**definition**

*step-z-alpha'* ::  $( 'a, 'c, t, 's ) \text{ ta} \Rightarrow 's \Rightarrow ( 'c, t ) \text{ zone} \Rightarrow 's \Rightarrow ( 'c, t ) \text{ zone} \Rightarrow \text{bool}$

$( \langle - \vdash \langle -, - \rangle \rightsquigarrow_{\alpha} \langle -, - \rangle \rangle [61,61,61] 61 )$

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle = (\exists Z' a. A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l, Z' \rangle \wedge A \vdash \langle l, Z' \rangle \rightsquigarrow_{\alpha(1a)} \langle l', Z'' \rangle)$

**abbreviation**

*steps-z-alpha* ::  $( 'a, 'c, t, 's ) \text{ ta} \Rightarrow 's \Rightarrow ( 'c, t ) \text{ zone} \Rightarrow 's \Rightarrow ( 'c, t ) \text{ zone} \Rightarrow \text{bool}$

$( \langle - \vdash \langle -, - \rangle \rightsquigarrow_{\alpha^*} \langle -, - \rangle \rangle [61,61,61] 61 )$

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z'' \rangle \equiv (\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle)^{**} (l, Z) (l', Z'')$

P. Bouyer's calculation for  $Post(Closure_{\alpha,l} Z, e) \subseteq Closure_{\alpha,l}(Post(Z, e))$

This is now obsolete as we argue solely with monotonicity of *steps-z* w.r.t  $Closure_{\alpha,l}$

Turning P. Bouyer's argument for multiple steps into an inductive proof is not

direct. With this initial argument we can get to a point where the induction hypothesis is applicable. This breaks the "information hiding" induced by the different variants of steps.

**context**

fixes  $l \ l' :: 's$

**begin**

**interpretation** *alpha*: *AlphaClosure-global - k l R l by standard (rule finite)*

**lemma** [*simp*]: *alpha.cla = cla l unfolding alpha.cla-def cla-def ..*

**interpretation** *alpha'*: *AlphaClosure-global - k l' R l' by standard (rule finite)*

**lemma** [*simp*]: *alpha'.cla = cla l' unfolding alpha'.cla-def cla-def ..*

**lemma** *steps-z-alpha-closure-involutive'-aux'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \text{Closure}_{\alpha, l} Z \subseteq \text{Closure}_{\alpha, l} W \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V$

$\implies W \subseteq Z \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle \wedge \text{Closure}_{\alpha, l'} Z' \subseteq \text{Closure}_{\alpha, l'} W' \wedge W' \subseteq Z'$

**proof** (*induction*  $A \equiv A \ l \equiv l - - \ l' \equiv l'$  -rule: *step-z.induct*)

**case** *A*: (*step-t-z Z*)

**let**  $?Z' = Z^\uparrow \cap \{u. u \vdash \text{inv-of } A \ l\}$

**let**  $?W' = W^\uparrow \cap \{u. u \vdash \text{inv-of } A \ l\}$

**have** *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_\tau \langle l, ?W' \rangle$  **by** *auto*

**moreover have**  $\text{Closure}_{\alpha, l} ?Z' \subseteq \text{Closure}_{\alpha, l} ?W'$

**proof**

**fix** *v* **assume**  $v: v \in \text{Closure}_{\alpha, l} ?Z'$

**then obtain**  $R' \ v'$  **where**  $1: R' \in \mathcal{R} \ l \ v \in R' \ v' \in R' \ v' \in ?Z'$  **unfolding** *cla-def* **by** *auto*

**then obtain**  $u \ d$  **where**

$u \in Z$  **and**  $v': v' = u \oplus d \ u \oplus d \vdash \text{inv-of } A \ l \ 0 \leq d$

**unfolding** *zone-delay-def* **by** *blast*

**with** *alpha.closure-subs*[*OF*  $A(4)$ ]  $A(2)$  **obtain**  $u' \ R$  **where**  $u'$ :

$u' \in W \ u \in R \ u' \in R \ R \in \mathcal{R} \ l$

**by** (*simp* *add: cla-def*) *blast*

**then have**  $\forall x \in X. 0 \leq u \ x$  **unfolding** *R-def* **by** *fastforce*

**from** *region-cover'*[*OF* *this*] **have**  $R: [u]_l \in \mathcal{R} \ l \ u \in [u]_l$  **by** *auto*

**from** *SuccI2*[*OF* *R-def'* *this*(2,1)  $\langle 0 \leq d \rangle$ , of  $[v']_l \ v'(1)$ ] **have**  $v'(1)$ :

$[v']_l \in \text{Succ}(\mathcal{R} \ l) ([u]_l) \ [v']_l \in \mathcal{R} \ l$

**by** *auto*

**from** *alpha.regions-closed'-spec*[*OF*  $R(1,2) \ \langle 0 \leq d \rangle \ v'(1)$ ] **have**  $v'2: v' \in [v']_l$  **by** *simp*

**from**  $A(\beta)$  **have**  
 $\forall (x, m) \in \text{clkp-set } A \ l. \ m \leq \text{real } (k \ l \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**by** (*auto elim!*: *valid-abstraction.cases*)  
**then have**  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l). \ m \leq \text{real } (k \ l \ x) \wedge x \in X \wedge$   
 $m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clki-def inv-of-def* **by** *fastforce*  
**from** *ccompatible*[*OF this, folded*  $\mathcal{R}\text{-def}$ ]  $v'1(2) \ v'2 \ v'(1,2)$  **have**  $\beta$ :  
 $[v']_l \subseteq \{\text{inv-of } A \ l\}$   
**unfolding** *ccompatible-def cval-def* **by** *auto*  
**from**  
 $\text{alpha.valid-regions-distinct-spec}[OF \ v'1(2) \ 1(1) \ v'2 \ 1(3)]$   
 $\text{alpha.region-unique-spec}[OF \ u'(2,4)]$   
**have**  $2$ :  $[v']_l = R' \ [u]_l = R$  **by** *auto*  
**from**  $\text{alpha.set-of-regions-spec}[OF \ u'(4,3)] \ v'1(1) \ 2$  **obtain**  $t$  **where**  $t$ :  
 $t \geq 0 \ [u' \oplus t]_l = R'$  **by** *auto*  
**with**  $\text{alpha.regions-closed'-spec}[OF \ u'(4,3) \ \text{this}(1)] \ \text{step-t-r}(1)$  **have**  $*$ :  
 $u' \oplus t \in R'$  **by** *auto*  
**with**  $t(1) \ 3 \ 2 \ u'(1,3)$  **have**  $A \vdash \langle l, u' \rangle \rightarrow \langle l, u' \oplus t \rangle \ u' \oplus t \in ?W'$   
**unfolding** *zone-delay-def cval-def* **by** *auto*  
**with**  $*$   $1(1)$  **have**  $R' \subseteq \text{Closure}_{\alpha,l} \ ?W'$  **unfolding** *cla-def* **by** *auto*  
**with**  $1(2)$  **show**  $v \in \text{Closure}_{\alpha,l} \ ?W' ..$   
**qed**  
**moreover have**  $?W' \subseteq ?Z'$  **using**  $\langle W \subseteq Z \rangle$  **unfolding** *zone-delay-def*  
**by** *auto*  
**ultimately show**  $?case$  **unfolding**  $\langle l = l' \rangle$  **by** *auto*  
**next**  
**case**  $A$ : (*step-a-z g a r Z*)  
**let**  $?Z' = \text{zone-set } (Z \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**let**  $?W' = \text{zone-set } (W \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  $A(1)$  **have** *step-z*:  $A \vdash \langle l, W \rangle \rightsquigarrow_{1a} \langle l', ?W' \rangle$  **by** *auto*  
**moreover have**  $\text{Closure}_{\alpha,l'} \ ?Z' \subseteq \text{Closure}_{\alpha,l'} \ ?W'$   
**proof**  
**fix**  $v$  **assume**  $v: v \in \text{Closure}_{\alpha,l'} \ ?Z'$   
**then obtain**  $R' \ v'$  **where**  $R' \in \mathcal{R} \ l' \ v \in R' \ v' \in R' \ v' \in ?Z'$  **unfolding**  
*cla-def* **by** *auto*  
**then obtain**  $u$  **where**  
 $u \in Z$  **and**  $v': v' = [r \rightarrow 0]u \ u \vdash g \ v' \vdash \text{inv-of } A \ l'$   
**unfolding** *zone-set-def* **by** *blast*  
**let**  $?R' = \text{region-set}' (([u]_l) \cap \{u. u \vdash g\}) \ r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\}$   
**from**  $\langle u \in Z \rangle$   $\text{alpha.closure-subs}[OF \ A(4)] \ A(2)$  **obtain**  $u' \ R$  **where**  $u'$ :  
 $u' \in W \ u \in R \ u' \in R \ R \in \mathcal{R} \ l$   
**by** (*simp add: cla-def*) *blast*  
**then have**  $\forall x \in X. \ 0 \leq u \ x$  **unfolding**  $\mathcal{R}\text{-def}$  **by** *fastforce*

**from** *region-cover*'[*OF this*] **have**  $[u]_l \in \mathcal{R} \ l \ u \in [u]_l$  **by** *auto*  
**have** \*:  
 $[u]_l = ([u]_l) \cap \{u. u \vdash g\}$   
*region-set'*  $([u]_l) \ r \ 0 \subseteq [[r \rightarrow 0]u]_{l'} \ [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l'$   
 $([[r \rightarrow 0]u]_{l'} \wedge) \cap \{u. u \vdash \text{inv-of } A \ l'\} = [[r \rightarrow 0]u]_{l'}$   
**proof** –  
**from**  $A(3)$  **have** *collect-clkvt* (*trans-of*  $A$ )  $\subseteq X$   
 $\forall l \ g \ a \ r \ l' \ c. A \vdash l \xrightarrow{g,a,r} l' \wedge c \notin \text{set } r \xrightarrow{} k \ l' \ c \leq k \ l \ c$   
**by** (*auto elim: valid-abstraction.cases*)  
**with**  $A(1)$  **have**  $\text{set } r \subseteq X \ \forall y. y \notin \text{set } r \xrightarrow{} k \ l' \ y \leq k \ l \ y$   
**unfolding** *collect-clkvt-def* **by** (*auto 4 8*)  
**with**  
*region-set-subst*  
*of* -  $X \ k \ l - 0$ , **where**  $k' = k \ l'$ , *folded*  $\mathcal{R}$ -*def*, *OF*  $\langle [u]_l \in \mathcal{R} \ l \rangle \langle u \in [u]_l \rangle$  *finite*  
**show** *region-set'*  $([u]_l) \ r \ 0 \subseteq [[r \rightarrow 0]u]_{l'} \ [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l'$  **by** *auto*  
**from**  $A(3)$  **have** \*:  
 $\forall l. \forall (x, m) \in \text{clkp-set } A \ l. m \leq \text{real } (k \ l \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**by** (*fastforce elim: valid-abstraction.cases*) +  
**with**  $A(1)$  **have** \*\*:  $\forall (x, m) \in \text{collect-clock-pairs } g. m \leq \text{real } (k \ l \ x)$   
 $\wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def* *collect-clkt-def* **by** *fastforce*  
**from**  $\langle u \in [u]_l \rangle \langle [u]_l \in \mathcal{R} \ l \rangle$  *ccompatible*[*OF this, folded*  $\mathcal{R}$ -*def*]  $\langle u \vdash g \rangle$  **show**  
 $[u]_l = ([u]_l) \cap \{u. u \vdash g\}$   
**unfolding** *ccompatible-def* *ccval-def* **by** *blast*  
**have** \*\*:  $[r \rightarrow 0]u \in [[r \rightarrow 0]u]_{l'}$   
**using**  $\langle R' \in \mathcal{R} \ l' \rangle \langle v' \in R' \rangle$  *alpha'.region-unique-spec*  $v'(1)$  **by** *blast*  
**from** \* **have**  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l'). m \leq \text{real } (k \ l' \ x) \wedge x \in X$   
 $\wedge m \in \mathbb{N}$   
**unfolding** *inv-of-def* *clkp-set-def* *collect-clki-def* **by** *fastforce*  
**from** \*\*  $\langle [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l' \rangle$  *ccompatible*[*OF this, folded*  $\mathcal{R}$ -*def*]  $\langle v' \vdash \rightarrow \rangle$  **show**  
 $([[r \rightarrow 0]u]_{l'} \wedge) \cap \{u. u \vdash \text{inv-of } A \ l'\} = [[r \rightarrow 0]u]_{l'}$   
**unfolding** *ccompatible-def* *ccval-def*  $\langle v' = \rightarrow \rangle$  **by** *blast*  
**qed**  
**from** \*  $\langle v' = \rightarrow \rangle \langle u \in [u]_l \rangle$  **have**  $v' \in [[r \rightarrow 0]u]_{l'}$  **unfolding** *region-set'-def*  
**by** *auto*  
**from** *alpha'.valid-regions-distinct-spec*[*OF*  $*(3)$ ]  $\langle R' \in \mathcal{R} \ l' \rangle \langle v' \in [[r \rightarrow 0]u]_{l'} \rangle$   
 $\langle v' \in R' \rangle$   
**have**  $[[r \rightarrow 0]u]_{l'} = R'$ .  
**from** *alpha.region-unique-spec*[*OF*  $u'(2,4)$ ] **have**  $[u]_l = R$  **by** *auto*

**from**  $\langle [u]_l = R \rangle * (1,2) * (4) \langle u' \in R \rangle$  **have**  
 $[r \rightarrow 0]u' \in [[r \rightarrow 0]u]_{l'} u' \vdash g [r \rightarrow 0]u' \vdash \text{inv-of } A \ l'$   
**unfolding** *region-set'-def by auto*  
**with**  $u'(1)$  **have**  $[r \rightarrow 0]u' \in ?W'$  **unfolding** *zone-set-def by auto*  
**with**  $\langle [r \rightarrow 0]u' \in [[r \rightarrow 0]u]_{l'} \rangle \langle [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l' \rangle$  **have**  $[[r \rightarrow 0]u]_{l'} \subseteq$   
*Closure <sub>$\alpha, l'$</sub>*   $?W'$   
**unfolding** *cla-def by auto*  
**with**  $\langle v \in R' \rangle$  **show**  $v \in \text{Closure}_{\alpha, l'} ?W'$  **unfolding**  $\langle - = R' \rangle ..$   
**qed**  
**moreover** **have**  $?W' \subseteq ?Z'$  **using**  $\langle W \subseteq Z \rangle$  **unfolding** *zone-set-def by*  
*auto*  
**ultimately** **show** *?case by meson*  
**qed**

**end**

**lemma** *step-z-alpha-mono:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle$   
 $\rightsquigarrow_{\alpha(a)} \langle l', W' \rangle \wedge Z' \subseteq W'$

**proof** *goal-cases*

**case** 1

**then obtain**  $Z''$  **where**  $*$ :  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z'' \rangle$   $Z' = \text{Closure}_{\alpha, l'} Z''$  **by**  
*auto*

**from** *step-z-mono[OF this(1) 1(2)]* **obtain**  $W'$  **where**  $A \vdash \langle l, W \rangle \rightsquigarrow_a$   
 $\langle l', W' \rangle$   $Z'' \subseteq W'$  **by** *auto*

**moreover** **with**  $*(2)$  **have**  $Z' \subseteq \text{Closure}_{\alpha, l'} W'$  **unfolding** *cla-def by*  
*auto*

**ultimately** **show** *?case by blast*

**qed**

**end**

**end**

**theory** *Approx-Beta*

**imports** *DBM-Zone-Semantics Regions-Beta Closure*

**begin**

**no-notation** *infinity* ( $\langle \infty \rangle$ )

## 6 Correctness of $\beta$ -approximation from $\alpha$ -regions

Merging the locales for the two types of regions

**locale** *Regions-defs* =

*Alpha-defs*  $X$  **for**  $X :: 'c$  set+

**fixes**  $v :: 'c \Rightarrow \text{nat}$  **and**  $n :: \text{nat}$

**begin**

**abbreviation**  $vabstr :: ('c, t)$  zone  $\Rightarrow - \Rightarrow -$  **where**

$vabstr S M \equiv S = [M]_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z})$

**definition**  $V' \equiv \{Z. Z \subseteq V \wedge (\exists M. vabstr Z M)\}$

**end**

**locale** *Regions-global* =

*Regions-defs*  $X v n$  **for**  $X :: 'c$  set **and**  $v n +$

**fixes**  $k :: 'c \Rightarrow \text{nat}$  **and** *not-in-X*

**assumes** *finite*: *finite*  $X$

**assumes** *clock-numbering*: *clock-numbering'*  $v n \forall k \leq n. k > 0 \longrightarrow (\exists c \in X. v c = k)$

$\forall c \in X. v c \leq n$

**assumes** *not-in-X*: *not-in-X*  $\notin X$

**assumes** *non-empty*:  $X \neq \{\}$

**begin**

**definition**  $\mathcal{R}\text{-def}$ :  $\mathcal{R} \equiv \{\text{Regions.region } X I r \mid I r. \text{Regions.valid-region } X k I r\}$

**sublocale** *alpha-interp*:

*AlphaClosure-global*  $X k \mathcal{R}$  **by** (*unfold-locales*) (*auto simp*: *finite*  $\mathcal{R}\text{-def}$   $V\text{-def}$ )

**sublocale** *beta-interp*: *Beta-Regions'*  $X k v n$  *not-in-X*

**rewrites** *beta-interp*.  $V = V$

**using** *finite non-empty clock-numbering not-in-X unfolding*  $V\text{-def}$

**by**  $- ((\text{subst } \text{Beta-Regions}.V\text{-def})?, \text{unfold-locales}; (\text{assumption} \mid \text{rule } \text{HOL.refl}))+$

**abbreviation**  $\mathcal{R}_\beta$  **where**  $\mathcal{R}_\beta \equiv \text{beta-interp}.\mathcal{R}$

lemmas  $\mathcal{R}_\beta\text{-def} = \text{beta-interp.}\mathcal{R}\text{-def}$

abbreviation  $\text{Approx}_\beta \equiv \text{beta-interp.}\text{Approx}_\beta$

## 6.1 Preparing Bouyer's Theorem

lemma *region-dbm*:

assumes  $R \in \mathcal{R}$

defines  $v' \equiv \lambda i. \text{THE } c. c \in X \wedge v c = i$

obtains  $M$

where $^{[M]}_{v,n} = R$

and  $\forall i \leq n. \forall j \leq n. M i 0 = \infty \wedge j > 0 \wedge i \neq j \longrightarrow M i j = \infty \wedge M j i = \infty$

and  $\forall i \leq n. M i i = \text{Le } 0$

and  $\forall i \leq n. \forall j \leq n. i > 0 \wedge j > 0 \wedge M i 0 \neq \infty \wedge M j 0 \neq \infty \longrightarrow (\exists d :: \text{int.}$

$(-k (v' j) \leq d \wedge d \leq k (v' i) \wedge M i j = \text{Le } d \wedge M j i = \text{Le } (-d))$

$\vee (-k (v' j) \leq d - 1 \wedge d \leq k (v' i) \wedge M i j = \text{Lt } d \wedge M j i = \text{Lt } (-d + 1)))$

and  $\forall i \leq n. i > 0 \wedge M i 0 \neq \infty \longrightarrow$

$(\exists d :: \text{int. } d \leq k (v' i) \wedge d \geq 0$

$\wedge (M i 0 = \text{Le } d \wedge M 0 i = \text{Le } (-d) \vee M i 0 = \text{Lt } d \wedge M 0 i = \text{Lt } (-d + 1)))$

and  $\forall i \leq n. i > 0 \longrightarrow (\exists d :: \text{int. } -k (v' i) \leq d \wedge d \leq 0 \wedge (M 0 i = \text{Le } d \vee M 0 i = \text{Lt } d))$

and  $\forall i. \forall j. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z}$

and  $\forall i \leq n. \forall j \leq n. M i j \neq \infty \wedge i > 0 \wedge j > 0 \longrightarrow$

$(\exists d :: \text{int. } (M i j = \text{Le } d \vee M i j = \text{Lt } d) \wedge (-k (v' j)) \leq d \wedge d \leq k (v' i))$

**proof** –

**from** *assms* **obtain**  $I r$  **where**  $R: R = \text{region } X \text{ } I r \text{ valid-region } X k \text{ } I r$   
**unfolding**  $\mathcal{R}\text{-def}$  **by** *blast*

**let**  $?X_0 = \{x \in X. \exists d. I x = \text{Regions.intv.Intv } d\}$

**define**  $f$  **where**  $f \equiv$

$\lambda x. \text{if } \text{isIntv } (I x) \text{ then } \text{Lt } (\text{real } (\text{intv-const } (I x) + 1))$

$\text{else if } \text{isConst } (I x) \text{ then } \text{Le } (\text{real } (\text{intv-const } (I x)))$

$\text{else } \infty$

**define**  $g$  **where**  $g \equiv$

$\lambda x. \text{if } \text{isIntv } (I x) \text{ then } \text{Lt } (-\text{real } (\text{intv-const } (I x)))$

$\text{else if } \text{isConst } (I x) \text{ then } \text{Le } (-\text{real } (\text{intv-const } (I x)))$

$\text{else } \text{Lt } (-\text{real } (k x))$

**define**  $h$  **where**  $h \equiv$

$\lambda x y. \text{if } \text{isIntv } (I x) \wedge \text{isIntv } (I y) \text{ then}$

```

    if  $(y, x) \in r \wedge (x, y) \notin r$  then  $Lt$  ( $real-of-int$  ( $int$  ( $intv-const$  ( $I x$ )) -
 $intv-const$  ( $I y$ ) + 1))
    else if  $(x, y) \in r \wedge (y, x) \notin r$  then  $Lt$  ( $int$  ( $intv-const$  ( $I x$ )) -  $intv-const$ 
( $I y$ ))
    else  $Le$  ( $int$  ( $intv-const$  ( $I x$ )) -  $intv-const$  ( $I y$ ))
    else if  $isConst$  ( $I x$ )  $\wedge$   $isConst$  ( $I y$ ) then  $Le$  ( $int$  ( $intv-const$  ( $I x$ )) -
 $intv-const$  ( $I y$ ))
    else if  $isIntv$  ( $I x$ )  $\wedge$   $isConst$  ( $I y$ ) then  $Lt$  ( $int$  ( $intv-const$  ( $I x$ )) + 1 -
 $intv-const$  ( $I y$ ))
    else if  $isConst$  ( $I x$ )  $\wedge$   $isIntv$  ( $I y$ ) then  $Lt$  ( $int$  ( $intv-const$  ( $I x$ )) -
 $intv-const$  ( $I y$ ))
    else  $\infty$ 
let  $?M = \lambda i j.$  if  $i = 0$  then if  $j = 0$  then  $Le$  0 else  $g$  ( $v' j$ )
    else if  $j = 0$  then  $f$  ( $v' i$ ) else if  $i = j$  then  $Le$  0 else  $h$  ( $v' i$ )
( $v' j$ )
have  $[?M]_{v,n} \subseteq R$ 
proof
  fix  $u$  assume  $u: u \in [?M]_{v,n}$ 
  show  $u \in R$  unfolding  $R$ 
  proof (standard, goal-cases)
    case 1
    show  $?case$ 
    proof
      fix  $c$  assume  $c: c \in X$ 
      with clock-numbering have  $c2: v c \leq n \vee c > 0 \vee (v c) = c$  unfolding
 $v'-def$  by auto
      with  $u$  have  $dbm-entry-val$   $u$   $None$  ( $Some$   $c$ ) ( $g$   $c$ )
      unfolding  $DBM-zone-repr-def$   $DBM-val-bounded-def$  by auto
      then show  $0 \leq u c$  by (cases  $isIntv$  ( $I c$ ); cases  $isConst$  ( $I c$ )) (auto
simp: g-def)
    qed
  next
  case 2
  show  $?case$ 
  proof
    fix  $c$  assume  $c: c \in X$ 
    with clock-numbering have  $c2: v c \leq n \vee c > 0 \vee (v c) = c$  unfolding
 $v'-def$  by auto
    with  $u$  have  $*$ :  $dbm-entry-val$   $u$   $None$  ( $Some$   $c$ ) ( $g$   $c$ )  $dbm-entry-val$ 
 $u$  ( $Some$   $c$ )  $None$  ( $f$   $c$ )
    unfolding  $DBM-zone-repr-def$   $DBM-val-bounded-def$  by auto
    show  $intv-elem$   $c$   $u$  ( $I c$ )
    proof (cases  $I c$ )
      case ( $Const$   $d$ )

```

```

    then have  $\neg \text{isIntv } (I\ c) \text{ isConst } (I\ c)$  by auto
    with * Const show ?thesis unfolding g-def f-def using Const by
auto
next
  case (Intv d)
  then have  $\text{isIntv } (I\ c) \neg \text{isConst } (I\ c)$  by auto
  with * Intv show ?thesis unfolding g-def f-def by auto
next
  case (Greater d)
  then have  $\neg \text{isIntv } (I\ c) \neg \text{isConst } (I\ c)$  by auto
  with * Greater R(2) c show ?thesis unfolding g-def f-def by
fastforce
qed
qed
next
  show  $?X_0 = ?X_0$  ..
  show  $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow \text{frac } (u\ x) \leq \text{frac } (u\ y)$ 
  proof (standard, standard)
    fix x y assume A:  $x \in ?X_0\ y \in ?X_0$ 
    show  $(x, y) \in r \longleftrightarrow \text{frac } (u\ x) \leq \text{frac } (u\ y)$ 
    proof (cases  $x = y$ )
      case True
      have refl-on  $?X_0\ r$  using R(2) by auto
      with A True show ?thesis unfolding refl-on-def by auto
    next
      case False
      from A obtain d d' where AA:
         $I\ x = \text{Intv } d\ I\ y = \text{Intv } d' \text{ isIntv } (I\ x) \text{ isIntv } (I\ y) \neg \text{isConst } (I\ x) \neg \text{isConst } (I\ y)$ 
      by auto
      from A False clock-numbering have B:
         $v\ x \leq n\ v\ x > 0\ v' (v\ x) = x\ v\ y \leq n\ v\ y > 0\ v' (v\ y) = y\ v\ x \neq$ 
v y
      unfolding v'-def by auto
      with u have *:
        dbm-entry-val u (Some x) (Some y) (h x y) dbm-entry-val u (Some
y) (Some x) (h y x)
        dbm-entry-val u None (Some x) (g x) dbm-entry-val u (Some x)
None (f x)
        dbm-entry-val u None (Some y) (g y) dbm-entry-val u (Some y)
None (f y)
      unfolding DBM-zone-repr-def DBM-val-bounded-def by force+
      show  $(x, y) \in r \longleftrightarrow \text{frac } (u\ x) \leq \text{frac } (u\ y)$ 
      proof

```

```

assume  $C: (x, y) \in r$ 
show  $\text{frac } (u x) \leq \text{frac } (u y)$ 
proof (cases  $(y, x) \in r$ )
  case False
    with * AA C have **:
       $u x - u y < \text{int } d - d'$ 
       $d < u x \ u x < d + 1 \ d' < u y \ u y < d' + 1$ 
      unfolding f-def g-def h-def by auto
      from nat-intv-frac-decomp[OF  $**(2,3)$ ] nat-intv-frac-decomp[OF
**(4,5)] show
         $\text{frac } (u x) \leq \text{frac } (u y)$ 
        by simp
    next
      case True
        with * AA C have **:
           $u x - u y \leq \text{int } d - d'$ 
           $d < u x \ u x < d + 1 \ d' < u y \ u y < d' + 1$ 
          unfolding f-def g-def h-def by auto
          from nat-intv-frac-decomp[OF  $**(2,3)$ ] nat-intv-frac-decomp[OF
**(4,5)] show
             $\text{frac } (u x) \leq \text{frac } (u y)$ 
            by simp
        qed
    next
      assume  $\text{frac } (u x) \leq \text{frac } (u y)$ 
      show  $(x, y) \in r$ 
      proof (rule ccontr)
        assume  $C: (x,y) \notin r$ 
        moreover from R(2) have total-on ?X0 r by auto
        ultimately have  $(y, x) \in r$  using False A unfolding total-on-def
by auto
        with  $*(2-)$  AA C have **:
           $u y - u x < \text{int } d' - d$ 
           $d < u x \ u x < d + 1 \ d' < u y \ u y < d' + 1$ 
          unfolding f-def g-def h-def by auto
          from nat-intv-frac-decomp[OF  $**(2,3)$ ] nat-intv-frac-decomp[OF
**(4,5)] have
             $\text{frac } (u y) < \text{frac } (u x)$ 
            by simp
          with  $\langle \text{frac } - \leq \rightarrow \rangle$  show False by auto
        qed
      qed
    qed
  qed

```

```

qed
qed
moreover have  $R \subseteq [?M]_{v,n}$ 
proof
  fix u assume u:  $u \in R$ 
  show  $u \in [?M]_{v,n}$  unfolding DBM-zone-repr-def DBM-val-bounded-def
  proof (safe, goal-cases)
    case 1 then show ?case by auto
  next
    case (2 c)
    with clock-numbering have  $c \in X$  by metis
    with clock-numbering have *:  $c \in X \ v \ c > 0 \ v' \ (v \ c) = c$  unfolding
v'-def by auto
    with  $R \ u$  have intv-elem c u (I c) valid-intv (k c) (I c) by auto
    then have dbm-entry-val u None (Some c) (g c) unfolding g-def by
(cases I c) auto
    with * show ?case by auto
  next
    case (3 c)
    with clock-numbering have  $c \in X$  by metis
    with clock-numbering have *:  $c \in X \ v \ c > 0 \ v' \ (v \ c) = c$  unfolding
v'-def by auto
    with  $R \ u$  have intv-elem c u (I c) valid-intv (k c) (I c) by auto
    then have dbm-entry-val u (Some c) None (f c) unfolding f-def by
(cases I c) auto
    with * show ?case by auto
  next
    case (4 c1 c2)
    with clock-numbering have  $c1 \in X \ c2 \in X$  by metis+
    with clock-numbering have *:
       $c1 \in X \ v \ c1 > 0 \ v' \ (v \ c1) = c1 \ c2 \in X \ v \ c2 > 0 \ v' \ (v \ c2) = c2$ 
    unfolding v'-def by auto
    with  $R \ u$  have
      intv-elem c1 u (I c1) valid-intv (k c1) (I c1)
      intv-elem c2 u (I c2) valid-intv (k c2) (I c2)
    by auto
    then have dbm-entry-val u (Some c1) (Some c2) (h c1 c2) unfolding
h-def
    proof(cases I c1, cases I c2, fastforce+, cases I c2, fastforce, goal-cases)
      case (1 d d')
      then show ?case
      proof (cases (c2, c1) ∈ r, goal-cases)
        case 1
        show ?case

```

```

proof (cases (c1, c2) ∈ r)
  case True
    with 1 *(1,4) R(1) u have frac (u c1) = frac (u c2) by auto
    with 1 have u c1 - u c2 = real d - d' by (fastforce dest:
nat-intv-frac-decomp)
    with 1 show ?thesis by auto
  next
    case False with 1 show ?thesis by auto
  qed
next
  case 2
  show ?case
  proof (cases c1 = c2)
    case True then show ?thesis by auto
  next
    case False
      with 2 R(2) *(1,4) have (c1, c2) ∈ r by (fastforce simp:
total-on-def)
      with 2 *(1,4) R(1) u have frac (u c1) < frac (u c2) by auto
      with 2 have u c1 - u c2 < real d - d' by (fastforce dest:
nat-intv-frac-decomp)
      with 2 show ?thesis by auto
    qed
  qed
qed fastforce+
then show ?case
proof (cases v c1 = v c2, goal-cases)
  case True with * clock-numbering have c1 = c2 by auto
  then show ?thesis by auto
next
  case 2 with * show ?case by auto
qed
qed
qed
ultimately have [?M]v,n = R by blast
moreover have ∀ i ≤ n. ∀ j ≤ n. ?M i 0 = ∞ ∧ j > 0 ∧ i ≠ j → ?M
i j = ∞ ∧ ?M j i = ∞
unfolding f-def h-def by auto
moreover have ∀ i ≤ n. ?M i i = Le 0 by auto
moreover
{ fix i j assume A: i ≤ n j ≤ n i > 0 j > 0 ?M i 0 ≠ ∞ ?M j 0 ≠ ∞
with clock-numbering(2) obtain c1 c2 where B: v c1 = i v c2 = j c1
∈ X c2 ∈ X by meson
with clock-numbering(1) A have C: v' i = c1 v' j = c2 unfolding

```

```

v'-def by force+
  from  $R(2)$   $B$  have valid: valid-intv ( $k$   $c1$ ) ( $I$   $c1$ ) valid-intv ( $k$   $c2$ ) ( $I$ 
   $c2$ ) by auto
  have  $\exists d :: \text{int. } (-k(v'j) \leq d \wedge d \leq k(v'i) \wedge ?Mij = Le\ d \wedge ?M$ 
   $ji = Le(-d)$ 
   $\vee (-k(v'j) \leq d - 1 \wedge d \leq k(v'i) \wedge ?Mij = Lt\ d \wedge ?Mji = Lt$ 
   $(-d + 1))$ 
  proof (cases  $i = j$ )
    case True
      then show ?thesis by auto
    next
      case False
        then show ?thesis
          proof (cases  $I\ c1$ , goal-cases)
            case 1
              then show ?case
                proof (cases  $I\ c2$ )
                  case Const
                    let  $?d = \text{int}(intv\text{-const}(I\ c1)) - \text{int}(intv\text{-const}(I\ c2))$ 
                    from Const 1 have isConst ( $I\ c1$ ) isConst ( $I\ c2$ ) by auto
                    with  $A(1-4)$   $C$  valid show ?thesis unfolding  $h\text{-def}$  by (intro
                    exI[where  $x = ?d$ ]) auto
                  next
                    case Intv
                      let  $?d = \text{int}(intv\text{-const}(I\ c1)) - \text{int}(intv\text{-const}(I\ c2))$ 
                      from Intv 1 have isConst ( $I\ c1$ ) isIntv ( $I\ c2$ ) by auto
                      with  $A(1-4)$   $C$  valid show ?thesis unfolding  $h\text{-def}$  by (intro
                      exI[where  $x = ?d$ ]) auto
                  next
                    case Greater
                      then have  $\neg isIntv(I\ c2) \wedge \neg isConst(I\ c2)$  by auto
                      with  $A\ 1(1)$   $C$  have False unfolding  $f\text{-def}$  by simp
                      then show ?thesis by fast
                  qed
                next
                  case 2
                    then show ?case
                      proof (cases  $I\ c2$ )
                        case Const
                          let  $?d = \text{int}(intv\text{-const}(I\ c1)) + 1 - \text{int}(intv\text{-const}(I\ c2))$ 
                          from Const 2 have isIntv ( $I\ c1$ ) isConst ( $I\ c2$ ) by auto
                          with  $A(1-4)$   $C$  valid show ?thesis unfolding  $h\text{-def}$  by (intro
                          exI[where  $x = ?d$ ]) auto
                        next

```

```

case Intv
with  $2$  have  $*$ : isIntv ( $I$   $c1$ ) isIntv ( $I$   $c2$ ) by auto
from Intv  $A(1-4)$   $C$  show ?thesis apply simp
proof (standard, goal-cases)
  case  $1$ 
  show ?case
  proof (cases ( $c2$ ,  $c1$ )  $\in r$ )
    case True
    note  $T = \text{this}$ 
    show ?thesis
    proof (cases ( $c1$ ,  $c2$ )  $\in r$ )
      case True
      let  $?d = \text{int}(\text{intv-const}(I\ c1)) - \text{int}(\text{intv-const}(I\ c2))$ 
      from True  $T * \text{valid}$  show ?thesis unfolding h-def by (intro
exI[where  $x = ?d$ ]) auto
      next
      case False
      let  $?d = \text{int}(\text{intv-const}(I\ c1)) - \text{int}(\text{intv-const}(I\ c2)) + 1$ 
      from False  $T * \text{valid}$  show ?thesis unfolding h-def by (intro
exI[where  $x = ?d$ ]) auto
      qed
    next
    case False
    let  $?d = \text{int}(\text{intv-const}(I\ c1)) - \text{int}(\text{intv-const}(I\ c2))$ 
    from False  $* \text{valid}$  show ?thesis unfolding h-def by (intro
exI[where  $x = ?d$ ]) auto
    qed
  next
  case Greater
  then have  $\neg \text{isIntv}(I\ c2) \neg \text{isConst}(I\ c2)$  by auto
  with  $A\ 2(1)$   $C$  have False unfolding f-def by simp
  then show ?thesis by fast
  qed
next
  case  $3$ 
  then have  $\neg \text{isIntv}(I\ c1) \neg \text{isConst}(I\ c1)$  by auto
  with  $A\ 3(1)$   $C$  have False unfolding f-def by simp
  then show ?thesis by fast
  qed
qed
}
moreover
{ fix  $i$  assume  $A: i \leq n\ i > 0\ ?M\ i\ 0 \neq \infty$ 

```

**with** *clock-numbering*(2) **obtain**  $c1$  **where**  $B: v\ c1 = i\ c1 \in X$  **by**  
*meson*  
**with** *clock-numbering*(1)  $A$  **have**  $C: v'\ i = c1$  **unfolding**  $v'$ -def **by**  
*force+*  
**from**  $R(2)$   $B$  **have** *valid: valid-intv* ( $k\ c1$ ) ( $I\ c1$ ) **by** *auto*  
**have**  $\exists d :: \text{int. } d \leq k\ (v'\ i) \wedge d \geq 0$   
 $\wedge (?M\ i\ 0 = Le\ d \wedge ?M\ 0\ i = Le\ (-d) \vee ?M\ i\ 0 = Lt\ d \wedge ?M\ 0\ i =$   
 $Lt\ (-d + 1))$   
**proof** (*cases*  $i = 0$ )  
**case** *True*  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**then show** *?thesis*  
**proof** (*cases*  $I\ c1$ , *goal-cases*)  
**case** 1  
**let**  $?d = \text{int}\ (\text{intv-const}\ (I\ c1))$   
**from** 1 **have**  $\text{isConst}\ (I\ c1) \neg \text{isIntv}\ (I\ c1)$  **by** *auto*  
**with**  $A\ C$  *valid* **show** *?thesis* **unfolding**  $f$ -def  $g$ -def **by** (*intro*  
*exI*[**where**  $x = ?d$ ]) *auto*  
**next**  
**case** 2  
**let**  $?d = \text{int}\ (\text{intv-const}\ (I\ c1)) + 1$   
**from** 2 **have**  $\text{isIntv}\ (I\ c1) \neg \text{isConst}\ (I\ c1)$  **by** *auto*  
**with**  $A\ C$  *valid* **show** *?thesis* **unfolding**  $f$ -def  $g$ -def **by** (*intro*  
*exI*[**where**  $x = ?d$ ]) *auto*  
**next**  
**case** 3  
**then have**  $\neg \text{isIntv}\ (I\ c1) \neg \text{isConst}\ (I\ c1)$  **by** *auto*  
**with**  $A\ 3(1)\ C$  *have* *False* **unfolding**  $f$ -def **by** *simp*  
**then show** *?thesis* **by** *fast*  
**qed**  
**qed**  
**}**  
**moreover**  
**{** **fix**  $i$  **assume**  $A: i \leq n\ i > 0$   
**with** *clock-numbering*(2) **obtain**  $c1$  **where**  $B: v\ c1 = i\ c1 \in X$  **by**  
*meson*  
**with** *clock-numbering*(1)  $A$  **have**  $C: v'\ i = c1$  **unfolding**  $v'$ -def **by**  
*force+*  
**from**  $R(2)$   $B$  **have** *valid: valid-intv* ( $k\ c1$ ) ( $I\ c1$ ) **by** *auto*  
**have**  $\exists d :: \text{int. } -k\ (v'\ i) \leq d \wedge d \leq 0 \wedge (?M\ 0\ i = Le\ d \vee ?M\ 0\ i =$   
 $Lt\ d)$   
**proof** (*cases*  $i = 0$ )

```

    case True
    then show ?thesis by auto
next
case False
then show ?thesis
proof (cases I c1, goal-cases)
  case 1
  let ?d = - int (intv-const (I c1))
  from 1 have isConst (I c1)  $\neg$  isIntv (I c1) by auto
  with A C valid show ?thesis unfolding f-def g-def by (intro
exI[where x = ?d]) auto
  next
  case 2
  let ?d = - int (intv-const (I c1))
  from 2 have isIntv(I c1)  $\neg$  isConst (I c1) by auto
  with A C valid show ?thesis unfolding f-def g-def by (intro
exI[where x = ?d]) auto
  next
  case 3
  let ?d = - (k c1)
  from 3 have  $\neg$  isIntv (I c1)  $\neg$  isConst (I c1) by auto
  with A C show ?thesis unfolding g-def by (intro exI[where x =
?d]) auto
qed
}
moreover have  $\forall i. \forall j. ?M i j \neq \infty \longrightarrow \text{get-const } (?M i j) \in \mathbb{Z}$ 
unfolding f-def g-def h-def by auto
moreover have  $\forall i \leq n. \forall j \leq n. i > 0 \wedge j > 0 \wedge ?M i j \neq \infty$ 
 $\longrightarrow (\exists d:: \text{int. } (?M i j = \text{Le } d \vee ?M i j = \text{Lt } d) \wedge (-k (v' j)) \leq d \wedge$ 
 $d \leq k (v' i))$ 
proof (auto, goal-cases)
  case A: (1 i j)
  with clock-numbering(2) obtain c1 c2 where B:  $v c1 = i c1 \in X v c2$ 
 $= j c2 \in X$  by meson
  with clock-numbering(1) A have C:  $v' i = c1 v' j = c2$  unfolding
v'-def by force+
  from R(2) B have valid: valid-intv (k c1) (I c1) valid-intv (k c2) (I
c2) by auto
  with A B C show ?case
proof (simp, goal-cases)
  case 1
  show ?case
proof (cases I c1, goal-cases)

```

```

case 1
then show ?case
proof (cases I c2)
  case Const
  let ?d = int (intv-const (I c1)) - int (intv-const (I c2))
  from Const 1 have isConst (I c1) isConst (I c2) by auto
  with A(1-4) C valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto
  next
  case Intv
  let ?d = int(intv-const (I c1)) - int (intv-const (I c2))
  from Intv 1 have isConst (I c1) isIntv (I c2) by auto
  with A(1-4) C valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto
  next
  case Greater
  then have ¬ isIntv (I c2) ¬ isConst (I c2) by auto
  with A 1(1) C show ?thesis unfolding h-def by simp
qed
next
case 2
then show ?case
proof (cases I c2)
  case Const
  let ?d = int (intv-const (I c1)) + 1 - int (intv-const (I c2))
  from Const 2 have isIntv (I c1) isConst (I c2) by auto
  with A(1-4) C valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto
  next
  case Intv
  with 2 have *: isIntv (I c1) isIntv (I c2) by auto
  from Intv A(1-4) C show ?thesis
  proof goal-cases
    case 1
    show ?case
    proof (cases (c2, c1) ∈ r)
      case True
      note T = this
      show ?thesis
      proof (cases (c1, c2) ∈ r)
        case True
        let ?d = int (intv-const (I c1)) - int (intv-const (I c2))
        from True T * valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto

```

```

      next
      case False
      let ?d = int (intv-const (I c1)) - int (intv-const (I c2)) + 1
      from False T * valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto
      qed
    next
    case False
    let ?d = int (intv-const (I c1)) - int (intv-const (I c2))
    from False * valid show ?thesis unfolding h-def by (intro
exI[where x = ?d]) auto
    qed
  next
  case Greater
  then have  $\neg$  isIntv (I c2)  $\neg$  isConst (I c2) by auto
  with A 2(1) C show ?thesis unfolding h-def by simp
  qed
next
case 3
then have  $\neg$  isIntv (I c1)  $\neg$  isConst (I c1) by auto
with A 3(1) C show ?thesis unfolding h-def by simp
qed
qed
moreover show ?thesis
  apply (rule that)
    apply (rule calculation(1))
    apply (rule calculation(2))
    apply (rule calculation(3))
    apply (blast intro: calculation)+
    apply (rule calculation(7))
  using calculation(8) apply blast
done
qed

```

**lemma** *len-inf-elim*:

```

(a, b) ∈ set (arcs i j xs) ⇒ M a b = ∞ ⇒ len M i j xs = ∞
apply (induction rule: arcs.induct)
apply (auto simp: add)
apply (rename-tac a' b' x xs)
apply (case-tac M a' x)
by auto

```

**lemma** *zone-diag-lt*:  
**assumes**  $a \leq n$   $b \leq n$  **and**  $C: v\ c1 = a\ v\ c2 = b$  **and**  $not0: a > 0\ b > 0$   
**shows**  $[(\lambda\ i\ j. \text{if } i = a \wedge j = b \text{ then } Lt\ d \text{ else } \infty)]_{v,n} = \{u. u\ c1 - u\ c2 < d\}$   
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def*  
**proof** (*standard, goal-cases*)  
**case** 1  
**then show** *?case* **using**  $\langle a \leq n \rangle\ \langle b \leq n \rangle\ C$  **by** *fastforce*  
**next**  
**case** 2  
**then show** *?case*  
**proof** (*safe, goal-cases*)  
**case** 1 **from**  $not0$  **show** *?case* **unfolding** *dbm-le-def* **by** *auto*  
**next**  
**case** 2 **with**  $not0$  **show** *?case* **by** *auto*  
**next**  
**case** 3 **with**  $not0$  **show** *?case* **by** *auto*  
**next**  
**case** (4  $u'\ y\ z$ )  
**show** *?case*  
**proof** (*cases*  $v\ y = a \wedge v\ z = b$ )  
**case** *True*  
**with** 4 *clock-numbering*  $C\ \langle a \leq n \rangle\ \langle b \leq n \rangle$  **have**  $u'\ y - u'\ z < d$  **by**  
*metis*  
**with** *True* **show** *?thesis* **by** *auto*  
**next**  
**case** *False* **then show** *?thesis* **by** *auto*  
**qed**  
**qed**  
**qed**

**lemma** *zone-diag-le*:  
**assumes**  $a \leq n$   $b \leq n$  **and**  $C: v\ c1 = a\ v\ c2 = b$  **and**  $not0: a > 0\ b > 0$   
**shows**  $[(\lambda\ i\ j. \text{if } i = a \wedge j = b \text{ then } Le\ d \text{ else } \infty)]_{v,n} = \{u. u\ c1 - u\ c2 \leq d\}$   
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def*  
**proof** (*rule, goal-cases*)  
**case** 1  
**then show** *?case* **using**  $\langle a \leq n \rangle\ \langle b \leq n \rangle\ C$  **by** *fastforce*  
**next**  
**case** 2  
**then show** *?case*  
**proof** (*safe, goal-cases*)  
**case** 1 **from**  $not0$  **show** *?case* **unfolding** *dbm-le-def* **by** *auto*

```

next
  case 2 with not0 show ?case by auto
next
  case 3 with not0 show ?case by auto
next
  case (4 u' y z)
  show ?case
  proof (cases v y = a ∧ v z = b)
    case True
    with 4 clock-numbering C ⟨a ≤ n⟩ ⟨b ≤ n⟩ have u' y - u' z ≤ d by
metis
    with True show ?thesis by auto
  next
    case False then show ?thesis by auto
  qed
qed
qed

```

lemma zone-diag-lt-2:

```

  assumes a ≤ n and C: v c = a and not0: a > 0
  shows [(λ i j. if i = a ∧ j = 0 then Lt d else ∞)]v,n = {u. u c < d}
unfolding DBM-zone-repr-def DBM-val-bounded-def
proof (rule, goal-cases)
  case 1
  then show ?case using ⟨a ≤ n⟩ C by fastforce
next
  case 2
  then show ?case
  proof (safe, goal-cases)
    case 1 from not0 show ?case unfolding dbm-le-def by auto
  next
    case 2 with not0 show ?case by auto
  next
    case (3 u c)
    show ?case
    proof (cases v c = a)
      case False then show ?thesis by auto
    next
      case True
      with 3 clock-numbering C ⟨a ≤ n⟩ have u c < d by metis
      with C show ?thesis by auto
    qed
  next
    case (4 u' y z)

```

```

    from clock-numbering(1) have 0 < v z by auto
    then show ?case by auto
qed
qed

```

lemma zone-diag-le-2:

```

    assumes a ≤ n and C: v c = a and not0: a > 0
    shows [(λ i j. if i = a ∧ j = 0 then Le d else ∞)]v,n = {u. u c ≤ d}
unfolding DBM-zone-repr-def DBM-val-bounded-def
proof (rule, goal-cases)
  case 1
  then show ?case using ⟨a ≤ n⟩ C by fastforce
next
  case 2
  then show ?case
  proof (safe, goal-cases)
    case 1 from not0 show ?case unfolding dbm-le-def by auto
  next
    case 2 with not0 show ?case by auto
  next
    case (3 u c)
    show ?case
    proof (cases v c = a)
      case False then show ?thesis by auto
    next
      case True
      with 3 clock-numbering C ⟨a ≤ n⟩ have u c ≤ d by metis
      with C show ?thesis by auto
    qed
  next
    case (4 u' y z)
    from clock-numbering(1) have 0 < v z by auto
    then show ?case by auto
  qed
qed

```

lemma zone-diag-lt-3:

```

    assumes a ≤ n and C: v c = a and not0: a > 0
    shows [(λ i j. if i = 0 ∧ j = a then Lt d else ∞)]v,n = {u. - u c < d}
unfolding DBM-zone-repr-def DBM-val-bounded-def
proof (rule, goal-cases)
  case 1
  then show ?case using ⟨a ≤ n⟩ C by fastforce
next

```

```

case 2
then show ?case
proof (safe, goal-cases)
  case 1 from not0 show ?case unfolding dbm-le-def by auto
next
  case (2 u c)
  show ?case
  proof (cases v c = a, goal-cases)
    case False then show ?thesis by auto
  next
    case True
    with 2 clock-numbering C ‹a ≤ n› have - u c < d by metis
    with C show ?thesis by auto
  qed
next
  case (3 u) with not0 show ?case by auto
next
  case (4 u' y z)
  from clock-numbering(1) have 0 < v y by auto
  then show ?case by auto
qed
qed

```

**lemma** len-int-closed:

```

  ∀ i j. (M i j :: real) ∈ ℤ ⇒ len M i j xs ∈ ℤ
by (induction xs arbitrary: i) auto

```

**lemma** get-const-distr:

```

  a ≠ ∞ ⇒ b ≠ ∞ ⇒ get-const (a + b) = get-const a + get-const b
by (cases a) (cases b, auto simp: add)+

```

**lemma** len-int-dbm-closed:

```

  ∀ (i, j) ∈ set (arcs i j xs). (get-const (M i j) :: real) ∈ ℤ ∧ M i j ≠ ∞
  ⇒ get-const (len M i j xs) ∈ ℤ ∧ len M i j xs ≠ ∞
by (induction xs arbitrary: i) (auto simp: get-const-distr, simp add: dbm-add-not-inf
add)

```

**lemma** zone-diag-le-3:

```

  assumes a ≤ n and C: v c = a and not0: a > 0
  shows [(λ i j. if i = 0 ∧ j = a then Le d else ∞)]v,n = {u. - u c ≤ d}
unfolding DBM-zone-repr-def DBM-val-bounded-def
proof (rule, goal-cases)
  case 1
  then show ?case using ‹a ≤ n› C by fastforce

```

```

next
  case 2
  then show ?case
  proof (safe, goal-cases)
    case 1 from not0 show ?case unfolding dbm-le-def by auto
  next
  case (2 u c)
  show ?case
  proof (cases v c = a)
    case False then show ?thesis by auto
  next
  case True
  with 2 clock-numbering C ⟨a ≤ n⟩ have - u c ≤ d bymetis
  with C show ?thesis by auto
  qed
next
  case (3 u) with not0 show ?case by auto
next
  case (4 u' y z)
  from clock-numbering(1) have 0 < v y by auto
  then show ?case by auto
  qed
qed

```

**lemma dbm-lt':**

```

assumes [M]v,n ⊆ V M a b ≤ Lt d a ≤ n b ≤ n v c1 = a v c2 = b a >
0 b > 0
shows [M]v,n ⊆ {u ∈ V. u c1 - u c2 < d}
proof -
  from assms have [M]v,n ⊆ [(λ i j. if i = a ∧ j = b then Lt d else ∞)]v,n
  apply safe
  apply (rule DBM-le-subset)
  unfolding less-eq dbm-le-def by auto
  moreover from zone-diag-lt[OF ⟨a ≤ n⟩ ⟨b ≤ n⟩ assms(5-)]
  have [(λ i j. if i = a ∧ j = b then Lt d else ∞)]v,n = {u. u c1 - u c2 <
d} by blast
  moreover from assms have [M]v,n ⊆ V by auto
  ultimately show ?thesis by auto
qed

```

**lemma dbm-lt'2:**

```

assumes [M]v,n ⊆ V M a 0 ≤ Lt d a ≤ n v c1 = a a > 0
shows [M]v,n ⊆ {u ∈ V. u c1 < d}
proof -

```

**from** *assms*(2) **have**  $[M]_{v,n} \subseteq [(\lambda i j. \text{if } i = a \wedge j = 0 \text{ then } Lt \ d \ \text{else } \infty)]_{v,n}$   
**apply** *safe*  
**apply** (*rule DBM-le-subset*)  
**unfolding** *less-eq dbm-le-def* **by** *auto*  
**moreover from** *zone-diag-lt-2*[*OF*  $\langle a \leq n \rangle$  *assms*(4,5)]  
**have**  $[(\lambda i j. \text{if } i = a \wedge j = 0 \text{ then } Lt \ d \ \text{else } \infty)]_{v,n} = \{u. u \ c1 < d\}$  **by**  
*blast*  
**ultimately show** *?thesis* **using** *assms*(1) **by** *auto*  
**qed**

**lemma** *dbm-lt'3*:

**assumes**  $[M]_{v,n} \subseteq V \ M \ 0 \ a \leq Lt \ d \ a \leq n \ v \ c1 = a \ a > 0$   
**shows**  $[M]_{v,n} \subseteq \{u \in V. - \ u \ c1 < d\}$   
**proof** –  
**from** *assms*(2) **have**  $[M]_{v,n} \subseteq [(\lambda i j. \text{if } i = 0 \wedge j = a \text{ then } Lt \ d \ \text{else } \infty)]_{v,n}$   
**apply** *safe*  
**apply** (*rule DBM-le-subset*)  
**unfolding** *less-eq dbm-le-def* **by** *auto*  
**moreover from** *zone-diag-lt-3*[*OF*  $\langle a \leq n \rangle$  *assms*(4,5)]  
**have**  $[(\lambda i j. \text{if } i = 0 \wedge j = a \text{ then } Lt \ d \ \text{else } \infty)]_{v,n} = \{u. - \ u \ c1 < d\}$   
**by** *blast*  
**ultimately show** *?thesis* **using** *assms*(1) **by** *auto*  
**qed**

**lemma** *dbm-le'*:

**assumes**  $[M]_{v,n} \subseteq V \ M \ a \ b \leq Le \ d \ a \leq n \ b \leq n \ v \ c1 = a \ v \ c2 = b \ a > 0 \ b > 0$   
**shows**  $[M]_{v,n} \subseteq \{u \in V. u \ c1 - u \ c2 \leq d\}$   
**proof** –  
**from** *assms* **have**  $[M]_{v,n} \subseteq [(\lambda i j. \text{if } i = a \wedge j = b \text{ then } Le \ d \ \text{else } \infty)]_{v,n}$   
**apply** *safe*  
**apply** (*rule DBM-le-subset*)  
**unfolding** *less-eq dbm-le-def* **by** *auto*  
**moreover from** *zone-diag-le*[*OF*  $\langle a \leq n \rangle \langle b \leq n \rangle$  *assms*(5–)]  
**have**  $[(\lambda i j. \text{if } i = a \wedge j = b \text{ then } Le \ d \ \text{else } \infty)]_{v,n} = \{u. u \ c1 - u \ c2 \leq d\}$  **by** *blast*  
**moreover from** *assms* **have**  $[M]_{v,n} \subseteq V$  **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *dbm-le'2*:

**assumes**  $[M]_{v,n} \subseteq V \ M \ a \ 0 \leq Le \ d \ a \leq n \ v \ c1 = a \ a > 0$

**shows**  $[M]_{v,n} \subseteq \{u \in V. u \text{ c1} \leq d\}$   
**proof** –  
**from** *assms*(2) **have**  $[M]_{v,n} \subseteq [(\lambda i j. \text{if } i = a \wedge j = 0 \text{ then } Le \ d \text{ else } \infty)]_{v,n}$   
**apply** *safe*  
**apply** (*rule DBM-le-subset*)  
**unfolding** *less-eq dbm-le-def* **by** *auto*  
**moreover from** *zone-diag-le-2*[*OF*  $\langle a \leq n \rangle$  *assms*(4,5)]  
**have**  $[(\lambda i j. \text{if } i = a \wedge j = 0 \text{ then } Le \ d \text{ else } \infty)]_{v,n} = \{u. u \text{ c1} \leq d\}$  **by**  
*blast*  
**ultimately show** *?thesis* **using** *assms*(1) **by** *auto*  
**qed**

**lemma** *dbm-le'3*:

**assumes**  $[M]_{v,n} \subseteq V \ M \ 0 \ a \leq Le \ d \ a \leq n \ v \ \text{c1} = a \ a > 0$   
**shows**  $[M]_{v,n} \subseteq \{u \in V. - \ u \text{ c1} \leq d\}$   
**proof** –  
**from** *assms*(2) **have**  $[M]_{v,n} \subseteq [(\lambda i j. \text{if } i = 0 \wedge j = a \text{ then } Le \ d \text{ else } \infty)]_{v,n}$   
**apply** *safe*  
**apply** (*rule DBM-le-subset*)  
**unfolding** *less-eq dbm-le-def* **by** *auto*  
**moreover from** *zone-diag-le-3*[*OF*  $\langle a \leq n \rangle$  *assms*(4,5)]  
**have**  $[(\lambda i j. \text{if } i = 0 \wedge j = a \text{ then } Le \ d \text{ else } \infty)]_{v,n} = \{u. - \ u \text{ c1} \leq d\}$   
**by** *blast*  
**ultimately show** *?thesis* **using** *assms*(1) **by** *auto*  
**qed**

**lemma** *int-zone-dbm*:

**assumes**  $\forall (-,d) \in \text{collect-clock-pairs } cc. d \in \mathbb{Z} \ \forall c \in \text{collect-clks } cc. v \ c \leq n$   
**obtains**  $M$  **where**  $\{u. u \vdash cc\} = [M]_{v,n}$  **and** *dbm-int*  $M \ n$   
**using** *int-zone-dbm*[*OF* - *assms*] *clock-numbering*(1) **by** *auto*

**lemma** *non-empty-dbm-diag-set'*:

**assumes** *clock-numbering'*  $v \ n \ \forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow \text{get-const } (M \ i \ j) \in \mathbb{Z}$   
 $[M]_{v,n} \neq \{\}$   
**obtains**  $M'$  **where**  $[M]_{v,n} = [M']_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M' \ i \ j \neq \infty \longrightarrow \text{get-const } (M' \ i \ j) \in \mathbb{Z})$   
 $\wedge (\forall i \leq n. M' \ i \ i = 0)$

**proof** –

**let**  $?M = \lambda i j. \text{if } i = j \text{ then } 0 \text{ else } M \ i \ j$   
**from** *non-empty-dbm-diag-set*[*OF* *assms*(1,3)] **have**  $[M]_{v,n} = [?M]_{v,n}$  **by**

*auto*

**moreover from** *assms(2)* **have**  $\forall i \leq n. \forall j \leq n. ?M\ i\ j \neq \infty \longrightarrow \text{get-const } (?M\ i\ j) \in \mathbf{Z}$

**unfolding** *neutral by auto*

**moreover have**  $\forall i \leq n. ?M\ i\ i = 0$  **by** *auto*

**ultimately show** *?thesis by (auto intro: that)*

**qed**

**lemma** *dbm-entry-int:*

$(x :: t\ DBMEntry) \neq \infty \implies \text{get-const } x \in \mathbf{Z} \implies \exists d :: \text{int. } x = Le\ d \vee x = Lt\ d$

**apply** (*cases x*) **using** *Ints-cases by auto*

## 6.2 Bouyer's Main Theorem

**theorem** *region-zone-intersect-empty-approx-correct:*

**assumes**  $R \in \mathcal{R}\ Z \subseteq V\ R \cap Z = \{\}$  *vabstr*  $Z\ M$

**shows**  $R \cap Approx_\beta\ Z = \{\}$

**proof** –

**define**  $v'$  **where**  $v' \equiv \lambda i. THE\ c. c \in X \wedge v\ c = i$

**from** *region-dbm[OF assms(1)]* **obtain**  $M_R$  **where**  $M_R:$

$[M_R]_{v,n} = R\ \forall i \leq n. \forall j \leq n. M_R\ i\ 0 = \infty \wedge 0 < j \wedge i \neq j \longrightarrow M_R\ i\ j = \infty \wedge M_R\ j\ i = \infty$

$\forall i \leq n. M_R\ i\ i = Le\ 0$

$\forall i \leq n. \forall j \leq n. 0 < i \wedge 0 < j \wedge M_R\ i\ 0 \neq \infty \wedge M_R\ j\ 0 \neq \infty \longrightarrow$

$(\exists d. -\ \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = j)) \leq d \wedge d \leq \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = i)))$

$\wedge M_R\ i\ j = Le\ d \wedge M_R\ j\ i = Le\ (\text{real-of-int}\ (-\ d))$

$\vee -\ \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = j)) \leq d - 1 \wedge d \leq \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = i))$

$\wedge M_R\ i\ j = Lt\ d \wedge M_R\ j\ i = Lt\ (\text{real-of-int}\ (-\ d + 1))$

$\forall i \leq n. 0 < i \wedge M_R\ i\ 0 \neq \infty \longrightarrow (\exists d \leq \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = i))).\ d \geq 0 \wedge$

$(M_R\ i\ 0 = Le\ d \wedge M_R\ 0\ i = Le\ (\text{real-of-int}\ (-\ d)) \vee M_R\ i\ 0 = Lt\ d \wedge M_R\ 0\ i = Lt\ (\text{real-of-int}\ (-\ d + 1)))$

$\forall i \leq n. 0 < i \longrightarrow (\exists d \geq -\ \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = i))).\ d \leq 0 \wedge (M_R\ 0\ i = Le\ d \vee M_R\ 0\ i = Lt\ d)$

$\forall i\ j. M_R\ i\ j \neq \infty \longrightarrow \text{get-const}\ (M_R\ i\ j) \in \mathbf{Z}$

$\forall i \leq n. \forall j \leq n. M_R\ i\ j \neq \infty \wedge 0 < i \wedge 0 < j \longrightarrow (\exists d. (M_R\ i\ j = Le\ d \vee M_R\ i\ j = Lt\ d)$

$\wedge -\ \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = j)) \leq d \wedge d \leq \text{int}\ (k\ (THE\ c. c \in X \wedge v\ c = i)))$

**show** *?thesis*

**proof** (*cases*  $R = \{\}$ )  
    **case** *True* **then show** *?thesis* **by** *auto*  
**next**  
    **case** *False*  
    **from** *clock-numbering(2)* **have** *cn-weak*:  $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k)$  **by** *auto*

**show** *?thesis*  
**proof** (*cases*  $Z = \{\}$ )  
    **case** *True*  
    **then show** *?thesis* **using** *beta-interp.apx-empty* **by** *blast*  
**next**  
    **case** *False*  
    **from** *assms(4)* **have**  
         $Z = [M]_{v,n} \forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z}$   
    **by** *auto*  
    **from** *this(1)* *non-empty-dbm-diag-set*<sup>[OF *clock-numbering(1)* *this(2)*]</sup>  
     $\langle Z \neq \{\} \rangle$  **obtain** *M* **where** *M*:  
         $Z = [M]_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z})$   
         $\wedge (\forall i \leq n. M i i = 0)$   
    **by** *auto*  
    **with** *not-empty-cyc-free*<sup>[OF *cn-weak*]</sup> *False* **have** *cyc-free M n* **by** *auto*  
    **then have** *cycle-free M n* **using** *cycle-free-diag-equiv* **by** *auto*  
    **from** *M* **have**  $Z = [FW M n]_{v,n}$  **unfolding** *neutral* **by** (*auto intro!*:  
*FW-zone-equiv*<sup>[OF *cn-weak*]</sup>)  
    **moreover from** *fw-canonical*<sup>[OF  $\langle \text{cyc-free } M \rightarrow \rangle$ ]</sup> *M* **have** *canonical*  
     $(FW M n) n$   
    **unfolding** *neutral* **by** *auto*  
    **moreover from** *FW-int-preservation M* **have**  
         $\forall i \leq n. \forall j \leq n. FW M n i j \neq \infty \longrightarrow \text{get-const } (FW M n i j) \in \mathbb{Z}$   
    **by** *auto*  
    **ultimately obtain** *M* **where** *M*:  
         $[M]_{v,n} = Z$  *canonical M n*  $\forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const}$   
         $(M i j) \in \mathbb{Z}$   
    **by** *blast*  
    **let**  $?M = \lambda i j. \min (M i j) (M_R i j)$   
    **from** *M(1)* *M<sub>R</sub>(1)* *assms* **have**  $[M]_{v,n} \cap [M_R]_{v,n} = \{\}$  **by** *auto*  
    **moreover from** *DBM-le-subset*<sup>[folded *less-eq*, of *n* *?M M*]</sup> **have**  $[?M]_{v,n}$   
     $\subseteq [M]_{v,n}$  **by** *auto*  
    **moreover from** *DBM-le-subset*<sup>[folded *less-eq*, of *n* *?M M<sub>R</sub>]*</sup> **have**  
     $[?M]_{v,n} \subseteq [M_R]_{v,n}$  **by** *auto*  
    **ultimately have**  $[?M]_{v,n} = \{\}$  **by** *blast*  
    **then have**  $\neg \text{cyc-free } ?M n$  **using** *cyc-free-not-empty*<sup>[of *n* *?M v*]</sup>  
    *clock-numbering(1)* **by** *auto*

**then obtain**  $i$   $xs$  **where**  $xs: i \leq n$   $set\ xs \subseteq \{0..n\}$   $len\ ?M\ i\ i\ xs < 0$   
**by** *auto*  
**from**  $this(1,2)$  *canonical-shorten-rotate-neg-cycle*[*OF*  $M(2)$   $this(2,1,3)$ ]  
**obtain**  $i$   $ys$  **where**  $ys$ :  
 $len\ ?M\ i\ i\ ys < 0$   
 $set\ ys \subseteq \{0..n\}$  *successive*  $(\lambda(a, b). ?M\ a\ b = M\ a\ b)$   $(arcs\ i\ i\ ys)\ i$   
 $\leq n$   
**and** *distinct*:  $distinct\ ys\ i \notin set\ ys$   
**and** *cycle-closes*:  $ys \neq [] \longrightarrow ?M\ i\ (hd\ ys) \neq M\ i\ (hd\ ys) \vee ?M\ (last\ ys)\ i \neq M\ (last\ ys)\ i$   
**by** *fastforce*  
  
**have** *one-M-aux*:  
 $len\ ?M\ i\ j\ ys = len\ M_R\ i\ j\ ys$  **if**  $\forall (a,b) \in set\ (arcs\ i\ j\ ys). M\ a\ b \geq M_R\ a\ b$  **for**  $j$   
**using** *that* **by** (*induction*  $ys$  *arbitrary*:  $i$ ) (*auto* *simp*: *min-def*)  
**have** *one-M*:  $\exists (a,b) \in set\ (arcs\ i\ i\ ys). M\ a\ b < M_R\ a\ b$   
**proof** (*rule* *ccontr*, *goal-cases*)  
**case**  $1$   
**then** **have**  $\forall (a, b) \in set\ (arcs\ i\ i\ ys). M_R\ a\ b \leq M\ a\ b$  **by** *auto*  
**from** *one-M-aux*[*OF*  $this$ ] **have**  $len\ ?M\ i\ i\ ys = len\ M_R\ i\ i\ ys$  .  
**with**  $Nil\ ys(1)\ xs(3)$  **have**  $len\ M_R\ i\ i\ ys < 0$  **by** *simp*  
**from** *DBM-val-bounded-neg-cycle*[*OF* -  $\langle i \leq n \rangle \langle set\ ys \subseteq \rightarrow this\ cn-weak \rangle$ ]  
**have**  $[M_R]_{v,n} = \{\}$  **unfolding** *DBM-zone-repr-def* **by** *auto*  
**with**  $\langle R \neq \{\} \rangle M_R(1)$  **show** *False* **by** *auto*  
**qed**  
**have** *one-M-R-aux*:  
 $len\ ?M\ i\ j\ ys = len\ M\ i\ j\ ys$  **if**  $\forall (a,b) \in set\ (arcs\ i\ j\ ys). M\ a\ b \leq M_R\ a\ b$  **for**  $j$   
**using** *that* **by** (*induction*  $ys$  *arbitrary*:  $i$ ) (*auto* *simp*: *min-def*)  
**have** *one-M-R*:  $\exists (a,b) \in set\ (arcs\ i\ i\ ys). M\ a\ b > M_R\ a\ b$   
**proof** (*rule* *ccontr*, *goal-cases*)  
**case**  $1$   
**then** **have**  $\forall (a, b) \in set\ (arcs\ i\ i\ ys). M_R\ a\ b \geq M\ a\ b$  **by** *auto*  
**from** *one-M-R-aux*[*OF*  $this$ ] **have**  $len\ ?M\ i\ i\ ys = len\ M\ i\ i\ ys$  .  
**with**  $Nil\ ys(1)\ xs(3)$  **have**  $len\ M\ i\ i\ ys < 0$  **by** *simp*  
**from** *DBM-val-bounded-neg-cycle*[*OF* -  $\langle i \leq n \rangle \langle set\ ys \subseteq \rightarrow this\ cn-weak \rangle$ ]  
**have**  $[M]_{v,n} = \{\}$  **unfolding** *DBM-zone-repr-def* **by** *auto*  
**with**  $\langle Z \neq \{\} \rangle M(1)$  **show** *False* **by** *auto*  
**qed**  
  
**have**  $0: (0,0) \notin set\ (arcs\ i\ i\ ys)$

```

proof (cases ys = [])
  case False with distinct show ?thesis using arcs-distinct1 by blast
next
  case True with ys(1) have ?M i i < 0 by auto
  then have M i i < 0  $\vee$  MR i i < 0 by (simp add: min-less-iff-disj)
  from one-M one-M-R True show ?thesis by auto
qed

{ fix a b assume A: (a,b)  $\in$  set (arcs i i ys)
  assume not0: a > 0
  from aux1[OF ys(4,4,2) A] have C2: a  $\leq$  n by auto
  then obtain c1 where C: v c1 = a c1  $\in$  X
  using clock-numbering(2) not0 unfolding v'-def by meson
  then have v' a = c1 using clock-numbering C2 not0 unfolding
v'-def by fastforce
  with C C2 have  $\exists$  c  $\in$  X. v c = a  $\wedge$  v' a = c a  $\leq$  n by auto
} note clock-dest-1 = this
{ fix a b assume A: (a,b)  $\in$  set (arcs i i ys)
  assume not0: b > 0
  from aux1[OF ys(4,4,2) A] have C2: b  $\leq$  n by auto
  then obtain c2 where C: v c2 = b c2  $\in$  X
  using clock-numbering(2) not0 unfolding v'-def by meson
  then have v' b = c2 using clock-numbering C2 not0 unfolding
v'-def by fastforce
  with C C2 have  $\exists$  c  $\in$  X. v c = b  $\wedge$  v' b = c b  $\leq$  n by auto
} note clock-dest-2 = this
have clock-dest:
 $\bigwedge$  a b. (a,b)  $\in$  set (arcs i i ys)  $\implies$  a > 0  $\implies$  b > 0  $\implies$ 
 $\exists$  c1  $\in$  X.  $\exists$  c2  $\in$  X. v c1 = a  $\wedge$  v c2 = b  $\wedge$  v' a = c1  $\wedge$  v' b =
c2  $\&\&\&$  a  $\leq$  n  $\&\&\&$  b  $\leq$  n
using clock-dest-1 clock-dest-2 by (auto) presburger

{ fix a assume A: (a,0)  $\in$  set (arcs i i ys)
  assume not0: a > 0
  assume bounded: MR a 0  $\neq$   $\infty$ 
  assume lt: M a 0 < MR a 0
  from clock-dest-1[OF A not0] obtain c1 where C:
    v c1 = a c1  $\in$  X v' a = c1 and C2: a  $\leq$  n
  by blast
  from C2 not0 bounded MR(5) obtain d :: int where *:
    d  $\leq$  int (k (v' a))
    MR a 0 = Le d  $\wedge$  MR 0 a = Le (- d)  $\vee$  MR a 0 = Lt d  $\wedge$  MR 0
a = Lt (- d + 1)
  unfolding v'-def by auto

```

```

with  $C$  have  $**$ :  $d \leq \text{int } (k \ c1)$  by auto
from  $*(2)$  have ?thesis
proof (standard, goal-cases)
  case 1
    with  $lt$  have  $M \ a \ 0 < Le \ d$  by auto
    then have  $M \ a \ 0 \leq Lt \ d$  unfolding less less-eq dbm-le-def by
(fastforce elim!: dbm-lt.cases)
    from  $\text{dbm-lt}'2[OF \ \text{assms}(2)[\text{folded } M(1)] \ \text{this } C2 \ C(1) \ \text{not}0]$  have
 $[M]_{v,n} \subseteq \{u \in V. u \ c1 < d\}$ 
    by auto
    from  $\text{beta-interp.}\beta\text{-boundedness-lt}'[OF \ ** \ C(2) \ \text{this}, \ \text{unfolded}$ 
 $\mathcal{R}_\beta\text{-def}]$  have
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \ c1 < d\}$ 
    .
    moreover
    { fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
      with  $C \ C2$  have
         $\text{dbm-entry-val } u \ (\text{Some } c1) \ \text{None} \ (M_R \ a \ 0) \ \text{dbm-entry-val } u$ 
 $\text{None} \ (\text{Some } c1) \ (M_R \ 0 \ a)$ 
        unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
        then have  $u \ c1 = d$  using 1 by auto
        then have  $u \notin \{u \in V. u \ c1 < d\}$  by auto
      }
    ultimately show ?thesis using  $M_R(1) \ M(1)$  by auto
next
  case 2
    from 2  $lt$  have  $M \ a \ 0 \neq \infty$  by auto
    with  $\text{dbm-entry-int}[OF \ \text{this}] \ M(3) \ \langle a \leq n \rangle$ 
    obtain  $d' :: \text{int}$  where  $d': M \ a \ 0 = Le \ d' \vee M \ a \ 0 = Lt \ d'$  by auto
    then have  $M \ a \ 0 \leq Le \ (d - 1)$  using  $lt \ 2$ 
    apply (auto simp: less-eq dbm-le-def less)
    apply (cases rule: dbm-lt.cases)
    apply auto
    apply rule
    apply (cases rule: dbm-lt.cases)
    by auto
    with  $lt$  have  $M \ a \ 0 \leq Le \ (d - 1)$  by auto
    from  $\text{dbm-le}'2[OF \ \text{assms}(2)[\text{folded } M(1)] \ \text{this } C2 \ C(1) \ \text{not}0]$  have
 $[M]_{v,n} \subseteq \{u \in V. u \ c1 \leq d - 1\}$ 
    by auto
    from  $\text{beta-interp.}\beta\text{-boundedness-le}'[OF \ - \ C(2) \ \text{this}] \ **$  have
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \ c1 \leq d - 1\}$ 
    by auto
    moreover

```

```

{ fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
  with  $C$   $C2$  have
     $dbm\text{-entry}\text{-val } u \text{ None } (Some\ c1) (M_R\ 0\ a)$ 
    unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 
    then have  $u\ c1 > d - 1$  using  $2$  by  $auto$ 
    then have  $u \notin \{u \in V. u\ c1 \leq d - 1\}$  by  $auto$ 
  }
  ultimately show  $?thesis$  using  $M_R(1)\ M(1)$  by  $auto$ 
qed
} note  $bounded\text{-zero}\text{-1} = this$ 

{ fix  $a$  assume  $A: (0, a) \in set\ (arcs\ i\ i\ ys)$ 
  assume  $not0: a > 0$ 
  assume  $bounded: M_R\ a\ 0 \neq \infty$ 
  assume  $lt: M\ 0\ a < M_R\ 0\ a$ 
  from  $clock\text{-dest}\text{-2}[OF\ A\ not0]$  obtain  $c1$  where  $C:$ 
     $v\ c1 = a\ c1 \in X\ v'\ a = c1$  and  $C2: a \leq n$ 
  by  $blast$ 
  from  $C2\ not0\ bounded\ M_R(5)$  obtain  $d :: int$  where  $*$ :
     $d \leq int\ (k\ (v'\ a))$ 
     $M_R\ a\ 0 = Le\ d \wedge M_R\ 0\ a = Le\ (-\ d) \vee M_R\ a\ 0 = Lt\ d \wedge M_R\ 0$ 
 $a = Lt\ (-\ d + 1)$ 
  unfolding  $v'\text{-def}$  by  $auto$ 
  with  $C$  have  $**:$   $- int\ (k\ c1) \leq - d$  by  $auto$ 
  from  $*(2)$  have  $?thesis$ 
  proof ( $standard, goal\text{-cases}$ )
    case  $1$ 
    with  $lt$  have  $M\ 0\ a < Le\ (-d)$  by  $auto$ 
    then have  $M\ 0\ a \leq Lt\ (-d)$  unfolding  $less\ less\text{-eq}\ dbm\text{-le}\text{-def}$  by
( $fastforce\ elim!: dbm\text{-lt}\text{-cases}$ )
    from  $dbm\text{-lt}'3[OF\ assms(2)[folded\ M(1)]\ this\ C2\ C(1)\ not0]$  have
 $[M]_{v,n} \subseteq \{u \in V. d < u\ c1\}$ 
    by  $auto$ 
    from  $beta\text{-interp}.\beta\text{-boundedness}\text{-gt}'[OF\ -\ C(2)\ this]$   $**$  have
 $Approx_\beta\ ([M]_{v,n}) \subseteq \{u \in V. - u\ c1 < -d\}$ 
    by  $auto$ 
    moreover
    { fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
      with  $C\ C2$  have
         $dbm\text{-entry}\text{-val } u\ (Some\ c1)\ None\ (M_R\ a\ 0)\ dbm\text{-entry}\text{-val } u$ 
 $None\ (Some\ c1)\ (M_R\ 0\ a)$ 
        unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 
        with  $1$  have  $u \notin \{u \in V. - u\ c1 < -d\}$  by  $auto$ 
      }
  }

```

```

ultimately show ?thesis using  $M_R(1)$   $M(1)$  by auto
next
case 2
from 2 lt have  $M \ 0 \ a \neq \infty$  by auto
with dbm-entry-int[OF this]  $M(3) \ \langle a \leq n \rangle$ 
obtain  $d' :: int$  where  $d': M \ 0 \ a = Le \ d' \vee M \ 0 \ a = Lt \ d'$  by auto
then have  $M \ 0 \ a \leq Le \ (-d)$  using lt 2
apply (auto simp: less-eq dbm-le-def less)
apply (cases rule: dbm-lt.cases)
apply auto
apply rule
apply (metis get-const.simps(2) 2 of-int-less-iff of-int-minus
zless-add1-eq)
apply (cases rule: dbm-lt.cases)
apply auto
apply (rule dbm-lt.intros(5))
by (simp add: int-lt-Suc-le)
from dbm-le'3[OF assms(2)[folded  $M(1)$ ] this C2 C(1) not0] have
 $[M]_{v,n} \subseteq \{u \in V. d \leq u \ c1\}$ 
by auto
from beta-interp. $\beta$ -boundedness-ge'[OF - C(2) this] ** have
 $Approx_\beta ([M]_{v,n}) \subseteq \{u \in V. - u \ c1 \leq -d\}$ 
by auto
moreover
{ fix u assume u:  $u \in [M_R]_{v,n}$ 
with C C2 have
dbm-entry-val u (Some c1) None ( $M_R \ a \ 0$ )
unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
with 2 have  $u \notin \{u \in V. - u \ c1 \leq -d\}$  by auto
}
ultimately show ?thesis using  $M_R(1)$   $M(1)$  by auto
qed
} note bounded-zero-2 = this

{ fix a b c c1 c2 assume A:  $(a,b) \in set \ (arcs \ i \ i \ ys)$ 
assume not0:  $a > 0 \ b > 0$ 
assume lt:  $M \ a \ b = Lt \ c$ 
assume neg:  $M \ a \ b + M_R \ b \ a < 0$ 
assume C:  $v \ c1 = a \ v \ c2 = b \ c1 \in X \ c2 \in X$  and C2:  $a \leq n \ b \leq n$ 
assume valid:  $-k \ c2 \leq -get-const \ (M_R \ b \ a) -get-const \ (M_R \ b \ a)$ 
 $\leq k \ c1$ 
from neg have  $M_R \ b \ a \neq \infty$  by auto
then obtain d where *:  $M_R \ b \ a = Le \ d \vee M_R \ b \ a = Lt \ d$  by (cases
 $M_R \ b \ a, auto$ )+

```

**with**  $M_R(\gamma) \langle - - - \neq \infty \rangle$  **have**  $d \in \mathbb{Z}$  **by** *fastforce*  
**with**  $*$  **obtain**  $d :: \text{int}$  **where**  $*$ :  $M_R \ b \ a = \text{Le } d \vee M_R \ b \ a = \text{Lt } d$   
**using** *Ints-cases by auto*  
**with** *valid* **have** *valid*:  $- \ k \ c2 \leq -d - d \leq k \ c1$  **by** *auto*  
**from**  $*$  *neg lt* **have**  $M \ a \ b \leq \text{Lt } (-d)$  **unfolding** *less-eq dbm-le-def*  
*add neutral less*  
**by** *(auto elim!: dbm-lt.cases)*  
**from** *dbm-lt'[OF assms(2)[folded M(1)] this C2 C(1,2) not0]* **have**  
 $[M]_{v,n} \subseteq \{u \in V. u \ c1 - u \ c2 < -d\}$   
 $\cdot$   
**from** *beta-interp. $\beta$ -boundedness-diag-lt'[OF valid C(3,4) this]* **have**  
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \ c1 - u \ c2 < -d\}$   
 $\cdot$   
**moreover**  
**{** **fix**  $u$  **assume**  $u: u \in [M_R]_{v,n}$   
**with**  $C \ C2$  **have**  
 $\text{dbm-entry-val } u \ (\text{Some } c2) \ (\text{Some } c1) \ (M_R \ b \ a)$   
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**with**  $*$  **have**  $u \notin \{u \in V. u \ c1 - u \ c2 < -d\}$  **by** *auto*  
**}**  
**ultimately have** *?thesis* **using**  $M_R(1) \ M(1)$  **by** *auto*  
**}** **note** *neg-sum-lt = this*

**{** **fix**  $a \ b$  **assume**  $A: (a,b) \in \text{set } (\text{arcs } i \ i \ ys)$   
**assume** *not0*:  $a > 0 \ b > 0$   
**assume** *neg*:  $M \ a \ b + M_R \ b \ a < 0$   
**from** *clock-dest[OF A not0]* **obtain**  $c1 \ c2$  **where**  
 $C: v \ c1 = a \ v \ c2 = b \ c1 \in X \ c2 \in X$  **and**  $C2: a \leq n \ b \leq n$   
**by** *blast*  
**then have**  $C3: v' \ a = c1 \ v' \ b = c2$  **unfolding** *v'-def* **using**  
*clock-numbering(1)* **by** *auto*  
**from** *neg* **have** *inf*:  $M \ a \ b \neq \infty \ M_R \ b \ a \neq \infty$  **by** *auto*  
**from**  $M_R(8)$  *inf not0*  $C(3,4) \ C2 \ C3$  **obtain**  $d :: \text{int}$  **where**  $d$ :  
 $M_R \ b \ a = \text{Le } d \vee M_R \ b \ a = \text{Lt } d - \text{int } (k \ c1) \leq d \ d \leq \text{int } (k \ c2)$   
**unfolding** *v'-def* **by** *auto*  
**from** *inf* **obtain**  $c$  **where**  $c: M \ a \ b = \text{Le } c \vee M \ a \ b = \text{Lt } c$  **by** *(cases*  
 $M \ a \ b)$  *auto*  
**{** **assume**  $**$ :  $M \ a \ b \leq \text{Lt } (-d)$   
**from** *dbm-lt'[OF assms(2)[folded M(1)] this C2 C(1,2) not0]* **have**  
 $[M]_{v,n} \subseteq \{u \in V. u \ c1 - u \ c2 < (-d)\}$   
 $\cdot$   
**from** *beta-interp. $\beta$ -boundedness-diag-lt'[OF - - C(3,4) this]*  $d$  **have**  
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \ c1 - u \ c2 < -d\}$   
**by** *auto*

```

moreover
{ fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
  with  $C$   $C2$  have
     $dbm\text{-entry}\text{-val } u (Some\ c2) (Some\ c1) (M_R\ b\ a)$ 
    unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 
    with  $d$  have  $u \notin \{u \in V. u\ c1 - u\ c2 < -d\}$  by  $auto$ 
  }
ultimately have  $?thesis$  using  $M_R(1)$   $M(1)$  by  $auto$ 
} note  $aux = this$ 
from  $c$  have  $?thesis$ 
proof ( $standard, goal\text{-cases}$ )
  case 2
    with  $neg\ d$  have  $M\ a\ b \leq Lt\ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
    add neutral less
    by ( $auto\ elim!: dbm\text{-lt}\text{-cases}$ )
    with  $aux$  show  $?thesis$  .
  next
    case 1
    note  $A = this$ 
    from  $d(1)$  show  $?thesis$ 
    proof ( $standard, goal\text{-cases}$ )
      case 1
        with  $A\ neg\ d$  have  $M\ a\ b \leq Lt\ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
        add neutral less
        by ( $auto\ elim!: dbm\text{-lt}\text{-cases}$ )
        with  $aux$  show  $?thesis$  .
      next
        case 2
        with  $A\ neg\ d$  have  $M\ a\ b \leq Le\ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
        add neutral less
        by ( $auto\ elim!: dbm\text{-lt}\text{-cases}$ )
        from  $dbm\text{-le}'[OF\ assms(2)][folded\ M(1)]\ this\ C2\ C(1,2)\ not0]$ 
have
 $[M]_{v,n} \subseteq \{u \in V. u\ c1 - u\ c2 \leq -d\}$ 
.
from  $beta\text{-interp}.\beta\text{-boundedness}\text{-diag}\text{-le}'[OF\ -\ -\ C(3,4)\ this]\ d$ 
have
 $Approx_\beta ([M]_{v,n}) \subseteq \{u \in V. u\ c1 - u\ c2 \leq -d\}$ 
by  $auto$ 
moreover
{ fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
  with  $C$   $C2$  have
     $dbm\text{-entry}\text{-val } u (Some\ c2) (Some\ c1) (M_R\ b\ a)$ 
    unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 

```

```

    with A 2 have  $u \notin \{u \in V. u \text{ c1} - u \text{ c2} \leq -d\}$  by auto
  }
  ultimately show ?thesis using  $M_R(1) M(1)$  by auto
qed
qed
} note neg-sum-1 = this

{ fix a b assume A:  $(a, 0) \in \text{set}(\text{arcs } i \text{ i } ys)$ 
  assume not0:  $a > 0$ 
  assume neg:  $M \ a \ 0 + M_R \ 0 \ a < 0$ 
  from clock-dest-1[OF A not0] obtain c1 where C:  $v \text{ c1} = a \text{ c1} \in$ 
X and C2:  $a \leq n$  by blast
  with clock-numbering(1) have C3:  $v' \ a = c1$  unfolding v'-def by
auto
  from neg have inf:  $M \ a \ 0 \neq \infty \ M_R \ 0 \ a \neq \infty$  by auto
  from  $M_R(6)$  not0 C2 C3 obtain d :: int where d:
     $M_R \ 0 \ a = Le \ d \vee M_R \ 0 \ a = Lt \ d - \text{int}(k \text{ c1}) \leq d \ d \leq 0$ 
  unfolding v'-def by auto
  from inf obtain c where c:  $M \ a \ 0 = Le \ c \vee M \ a \ 0 = Lt \ c$  by
(cases  $M \ a \ 0$ ) auto
  { assume  $M \ a \ 0 \leq Lt \ (-d)$ 
    from dbm-lt'2[OF assms(2)[folded  $M(1)$ ] this C2 C(1) not0] have
       $[M]_{v,n} \subseteq \{u \in V. u \text{ c1} < -d\}$ 
    .
    from beta-interp. $\beta$ -boundedness-lt'[OF - C(2) this] d have
       $Approx_\beta([M]_{v,n}) \subseteq \{u \in V. u \text{ c1} < -d\}$ 
    by auto
    moreover
    { fix u assume u:  $u \in [M_R]_{v,n}$ 
      with C C2 have
        dbm-entry-val u None (Some c1) ( $M_R \ 0 \ a$ )
        unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
        with d have  $u \notin \{u \in V. u \text{ c1} < -d\}$  by auto
      }
    ultimately have ?thesis using  $M_R(1) M(1)$  by auto
  }
} note aux = this
from c have ?thesis
proof (standard, goal-cases)
  case 2
  with neg d have  $M \ a \ 0 \leq Lt \ (-d)$  unfolding less-eq dbm-le-def
add neutral less
  by (auto elim!: dbm-lt.cases)
  with aux show ?thesis .
next

```

```

    case 1
    note A = this
    from d(1) show ?thesis
    proof (standard, goal-cases)
      case 1
      with A neg d have M a 0 ≤ Lt (-d) unfolding less-eq dbm-le-def
    add neutral less
      by (auto elim!: dbm-lt.cases)
      with aux show ?thesis .
    next
      case 2
      with A neg d have M a 0 ≤ Le (-d) unfolding less-eq dbm-le-def
    add neutral less
      by (auto elim!: dbm-lt.cases)
      from dbm-le'2[OF assms(2)][folded M(1)] this C2 C(1) not0]
  have
    [M]v,n ⊆ {u ∈ V. u c1 ≤ - d}
    .
    from beta-interp.β-boundedness-le'[OF - C(2) this] d have
      Approxβ ([M]v,n) ⊆ {u ∈ V. u c1 ≤ -d}
    by auto
    moreover
    { fix u assume u: u ∈ [M]Rv,n
      with C C2 have
        dbm-entry-val u None (Some c1) (MR 0 a)
        unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
        with A 2 have u ∉ {u ∈ V. u c1 ≤ -d} by auto
      }
    ultimately show ?thesis using MR(1) M(1) by auto
  qed
  qed
} note neg-sum-1' = this

{ fix a b assume A: (0,b) ∈ set (arcs i i ys)
  assume not0: b > 0
  assume neg: M 0 b + MR b 0 < 0
  from clock-dest-2[OF A not0] obtain c2 where
    C: v c2 = b c2 ∈ X and C2: b ≤ n
  by blast
  with clock-numbering(1) have C3: v' b = c2 unfolding v'-def by
auto
  from neg have M 0 b ≠ ∞ MR b 0 ≠ ∞ by auto
  with MR(5) not0 C2 C3 obtain d :: int where d:
    MR b 0 = Le d ∨ MR b 0 = Lt d d ≤ k c2

```

**unfolding**  $v'$ -def **by** *fastforce*  
**from**  $\langle M \ 0 \ b \neq \infty \rangle$  **obtain**  $c$  **where**  $c: M \ 0 \ b = Le \ c \vee M \ 0 \ b = Lt$   
 $c$  **by** (*cases*  $M \ 0 \ b$ ) *auto*  
{ **assume**  $M \ 0 \ b \leq Lt \ (-d)$   
**from**  $dbm\text{-}lt'\exists[OF \ assms(2)[folded \ M(1)] \ this \ C2 \ C(1) \ not0]$  **have**  
 $[M]_{v,n} \subseteq \{u \in V. u \ c2 > d\}$   
**by** *simp*  
**from**  $beta\text{-interp}.\beta\text{-boundedness-gt}'[OF - C(2) \ this] \ d$  **have**  
 $Approx_{\beta} ([M]_{v,n}) \subseteq \{u \in V. - u \ c2 < -d\}$   
**by** *auto*  
**moreover**  
{ **fix**  $u$  **assume**  $u: u \in [M_R]_{v,n}$   
**with**  $C \ C2$  **have**  
 $dbm\text{-}entry\text{-}val \ u \ (Some \ c2) \ None \ (M_R \ b \ 0)$   
**unfolding**  $DBM\text{-}zone\text{-}repr\text{-}def \ DBM\text{-}val\text{-}bounded\text{-}def$  **by** *auto*  
**with**  $d$  **have**  $u \notin \{u \in V. - u \ c2 < -d\}$  **by** *auto*  
}
**ultimately have** *?thesis* **using**  $M_R(1) \ M(1)$  **by** *auto*  
} **note**  $aux = this$   
**from**  $c$  **have** *?thesis*  
**proof** (*standard, goal-cases*)  
**case**  $2$   
**with**  $neg \ d$  **have**  $M \ 0 \ b \leq Lt \ (-d)$  **unfolding**  $less\text{-}eq \ dbm\text{-}le\text{-}def$   
*add neutral less*  
**by** (*auto elim!:*  $dbm\text{-}lt.cases$ )  
**with**  $aux$  **show** *?thesis* .  
**next**  
**case**  $A: 1$   
**from**  $d(1)$  **show** *?thesis*  
**proof** (*standard, goal-cases*)  
**case**  $1$   
**with**  $A \ neg$  **have**  $M \ 0 \ b \leq Lt \ (-d)$  **unfolding**  $less\text{-}eq \ dbm\text{-}le\text{-}def$   
*add neutral less*  
**by** (*auto elim!:*  $dbm\text{-}lt.cases$ )  
**with**  $aux$  **show** *?thesis* .  
**next**  
**case**  $2$   
**with**  $A \ neg \ c$  **have**  $M \ 0 \ b \leq Le \ (-d)$  **unfolding**  $less\text{-}eq \ dbm\text{-}le\text{-}def$   
*add neutral less*  
**by** (*auto elim!:*  $dbm\text{-}lt.cases$ )  
**from**  $dbm\text{-}le'\exists[OF \ assms(2)[folded \ M(1)] \ this \ C2 \ C(1) \ not0]$

**have**  
 $[M]_{v,n} \subseteq \{u \in V. u \ c2 \geq d\}$   
**by** *simp*

**from** *beta-interp.* $\beta$ -*boundedness-ge'*[*OF - C(2) this*] *d(2)* **have**  
*Approx <sub>$\beta$</sub>  ([M]<sub>v,n</sub>)*  $\subseteq \{u \in V. - u \ c2 \leq -d\}$   
**by** *auto*  
**moreover**  
**{** **fix** *u* **assume** *u: u*  $\in [M_R]_{v,n}$   
**with** *C C2* **have**  
*dbm-entry-val u (Some c2) None (M\_R b 0)*  
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**with** *A 2* **have**  $u \notin \{u \in V. - u \ c2 \leq -d\}$  **by** *auto*  
**}**  
**ultimately show** *?thesis* **using** *M\_R(1) M(1)* **by** *auto*  
**qed**  
**qed**  
**}** **note** *neg-sum-1'' = this*

**{** **fix** *a b* **assume** *A: (a,b)  $\in$  set (arcs i i ys)*  
**assume** *not0: b > 0 a > 0*  
**assume** *neg: M\_R a b + M b a < 0*  
**from** *clock-dest[OF A not0(2,1)]* **obtain** *c1 c2* **where**  
*C: v c1 = a v c2 = b c1  $\in$  X c2  $\in$  X and C2: a  $\leq$  n b  $\leq$  n*  
**by** *blast*  
**then have** *C3: v' a = c1 v' b = c2* **unfolding** *v'-def* **using**  
*clock-numbering(1)* **by** *auto*  
**from** *neg* **have** *inf: M b a  $\neq$   $\infty$  M\_R a b  $\neq$   $\infty$*  **by** *auto*  
**with** *M\_R(8) not0 C(3,4) C2 C3* **obtain** *d :: int* **where** *d:*  
*M\_R a b = Le d  $\vee$  M\_R a b = Lt d d  $\geq$  -int (k c2) d  $\leq$  int (k c1)*  
**unfolding** *v'-def* **by** *blast*  
**from** *inf* **obtain** *c* **where** *c: M b a = Le c  $\vee$  M b a = Lt c* **by** (*cases*  
*M b a*) *auto*  
**{** **assume** *M b a  $\leq$  Lt (-d)*  
**from** *dbm-lt'[OF assms(2)][folded M(1)] this C2(2,1) C(2,1) not0*  
**have**  
 $[M]_{v,n} \subseteq \{u \in V. u \ c2 - u \ c1 < -d\}$   
**}**  
**from** *beta-interp.* $\beta$ -*boundedness-diag-lt'*[*OF - - C(4,3) this*] *d*  
**have** *Approx <sub>$\beta$</sub>  ([M]<sub>v,n</sub>)*  $\subseteq \{u \in V. u \ c2 - u \ c1 < -d\}$  **by** *auto*  
**moreover**  
**{** **fix** *u* **assume** *u: u*  $\in [M_R]_{v,n}$   
**with** *C C2* **have**  
*dbm-entry-val u (Some c1) (Some c2) (M\_R a b)*  
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**with** *d* **have**  $u \notin \{u \in V. u \ c2 - u \ c1 < -d\}$  **by** *auto*  
**}**  
**ultimately have** *?thesis* **using** *M\_R(1) M(1)* **by** *auto*

```

} note aux = this
from c have ?thesis
proof (standard, goal-cases)
  case 2
  with neg d have M b a ≤ Lt (-d) unfolding less-eq dbm-le-def
add neutral less
  by (auto elim!: dbm-lt.cases)
  with aux show ?thesis .
next
case A: 1
from d(1) show ?thesis
proof (standard, goal-cases)
  case 1
  with A neg d have M b a ≤ Lt (-d) unfolding less-eq dbm-le-def
add neutral less
  by (auto elim!: dbm-lt.cases)
  with aux show ?thesis .
next
case 2
with A neg d have M b a ≤ Le (-d) unfolding less-eq dbm-le-def
add neutral less
  by (auto elim!: dbm-lt.cases)
  from dbm-le'[OF assms(2)] [folded M(1)] this C2(2,1) C(2,1)
not0] have
  [M]v,n ⊆ {u ∈ V. u c2 - u c1 ≤ -d}
  .
  from beta-interp.β-boundedness-diag-le'[OF - - C(4,3) this] d
  have Approxβ ([M]v,n) ⊆ {u ∈ V. u c2 - u c1 ≤ -d} by auto
  moreover
  { fix u assume u: u ∈ [M]Rv,n
  with C C2 have
    dbm-entry-val u (Some c1) (Some c2) (MR a b)
    unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
    with A 2 have u ∉ {u ∈ V. u c2 - u c1 ≤ -d} by auto
  }
  ultimately show ?thesis using MR(1) M(1) by auto
qed
qed
} note neg-sum-2 = this

{ fix a b assume A: (a,0) ∈ set (arcs i i ys)
  assume not0: a > 0
  assume neg: MR a 0 + M 0 a < 0
  from clock-dest-1[OF A not0] obtain c1 where C: v c1 = a c1 ∈

```

**X and C2:  $a \leq n$  by blast**  
**with clock-numbering(1) have C3:  $v' a = c1$  unfolding v'-def by auto**  
**from neg have inf:  $M 0 a \neq \infty M_R a 0 \neq \infty$  by auto**  
**with  $M_R(5)$  not0 C2 C3 obtain  $d :: int$  where  $d$ :**  
 $M_R a 0 = Le d \vee M_R a 0 = Lt d d \leq int (k c1) d \geq 0$   
**unfolding v'-def by auto**  
**from inf obtain  $c$  where  $c: M 0 a = Le c \vee M 0 a = Lt c$  by**  
*(cases M 0 a) auto*  
**{ assume  $M 0 a \leq Lt (-d)$**   
**from dbm-lt'3[OF assms(2)[folded M(1)] this C2 C(1) not0] have**  
 $[M]_{v,n} \subseteq \{u \in V. u c1 > d\}$   
**by simp**  
**from beta-interp. $\beta$ -boundedness-gt'[OF - C(2) this] d have**  
 $Approx_\beta ([M]_{v,n}) \subseteq \{u \in V. u c1 > d\}$   
**by auto**  
**moreover**  
**{ fix  $u$  assume  $u: u \in [M_R]_{v,n}$**   
**with C C2 have**  
 $dbm-entry-val u (Some c1) None (M_R a 0)$   
**unfolding DBM-zone-repr-def DBM-val-bounded-def by auto**  
**with  $d$  have  $u \notin \{u \in V. u c1 > d\}$  by auto**  
**}**  
**ultimately have ?thesis using  $M_R(1) M(1)$  by auto**  
**} note  $aux = this$**   
**from  $c$  have ?thesis**  
**proof (standard, goal-cases)**  
**case 2**  
**with neg  $d$  have  $M 0 a \leq Lt (-d)$  unfolding less-eq dbm-le-def**  
*add neutral less*  
**by (auto elim!: dbm-lt.cases)**  
**with  $aux$  show ?thesis .**  
**next**  
**case A: 1**  
**from  $d(1)$  show ?thesis**  
**proof (standard, goal-cases)**  
**case 1**  
**with A neg  $d$  have  $M 0 a \leq Lt (-d)$  unfolding less-eq dbm-le-def**  
*add neutral less*  
**by (auto elim!: dbm-lt.cases)**  
**with  $aux$  show ?thesis .**  
**next**  
**case 2**  
**with A neg  $d$  have  $M 0 a \leq Le (-d)$  unfolding less-eq dbm-le-def**

*add neutral less*  
**by** (*auto elim!*: *dbm-lt.cases*)  
**from** *dbm-le'3*[*OF assms(2)*][*folded M(1)*] *this C2 C(1) not0*  
**have**  
 $[M]_{v,n} \subseteq \{u \in V. u \text{ c1} \geq d\}$   
**by** *simp*  
**from** *beta-interp.beta-boundedness-ge'*[*OF - C(2) this*] *d have*  
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \text{ c1} \geq d\}$   
**by** *auto*  
**moreover**  
**{ fix u assume u: u ∈ [M<sub>R</sub>]<sub>v,n</sub>**  
**with C C2 have**  
*dbm-entry-val u (Some c1) None (M<sub>R</sub> a 0)*  
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**with A 2 have**  $u \notin \{u \in V. u \text{ c1} \geq d\}$  **by** *auto*  
**}**  
**ultimately show** *?thesis* **using** *M<sub>R</sub>(1) M(1)* **by** *auto*  
**qed**  
**qed**  
**}** *note neg-sum-2' = this*  
  
**{ fix a b assume A: (0,b) ∈ set (arcs i i ys)**  
**assume** *not0: b > 0*  
**assume** *neg: M<sub>R</sub> 0 b + M b 0 < 0*  
**from** *clock-dest-2*[*OF A not0*] **obtain** *c2* **where**  
*C: v c2 = b c2 ∈ X and C2: b ≤ n*  
**by** *blast*  
**with** *clock-numbering(1)* **have** *C3: v' b = c2* **unfolding** *v'-def* **by**  
*auto*  
**from** *neg* **have**  $M \text{ b } 0 \neq \infty \text{ } M_R \text{ 0 } b \neq \infty$  **by** *auto*  
**with** *M<sub>R</sub>(6) not0 C2 C3* **obtain** *d :: int* **where** *d:*  
 $M_R \text{ 0 } b = \text{Le } d \vee M_R \text{ 0 } b = \text{Lt } d \text{ } -d \leq k \text{ } c2$   
**unfolding** *v'-def* **by** *fastforce*  
**from**  $\langle M \text{ b } 0 \neq \infty \rangle$  **obtain** *c* **where** *c: M b 0 = Le c ∨ M b 0 =*  
*Lt c* **by** (*cases M b 0*) *auto*  
**{ assume**  $M \text{ b } 0 \leq \text{Lt } (-d)$   
**from** *dbm-lt'2*[*OF assms(2)*][*folded M(1)*] *this C2 C(1) not0*] **have**  
 $[M]_{v,n} \subseteq \{u \in V. u \text{ c2} < -d\}$   
**by** *simp*  
**from** *beta-interp.beta-boundedness-lt'*[*OF - C(2) this*] *d have*  
 $\text{Approx}_\beta ([M]_{v,n}) \subseteq \{u \in V. u \text{ c2} < -d\}$   
**by** *auto*  
**moreover**  
**{ fix u assume u: u ∈ [M<sub>R</sub>]<sub>v,n</sub>**

```

with  $C$   $C2$  have
   $dbm\text{-entry}\text{-val } u \text{ None (Some } c2) (M_R \ 0 \ b)$ 
unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 
with  $d$  have  $u \notin \{u \in V. u \ c2 < -d\}$  by  $auto$ 
}
ultimately have  $?thesis$  using  $M_R(1) \ M(1)$  by  $auto$ 
} note  $aux = this$ 
from  $c$  have  $?thesis$ 
proof ( $standard, \ goal\text{-cases}$ )
  case  $2$ 
    with  $neg \ d$  have  $M \ b \ 0 \leq Lt \ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
    add neutral less
    by ( $auto \ elim! : dbm\text{-lt}\text{-cases}$ )
    with  $aux$  show  $?thesis$  .
  next
    case  $1$ 
    note  $A = this$ 
    from  $d(1)$  show  $?thesis$ 
    proof ( $standard, \ goal\text{-cases}$ )
      case  $1$ 
        with  $A \ neg$  have  $M \ b \ 0 \leq Lt \ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
        add neutral less
        by ( $auto \ elim! : dbm\text{-lt}\text{-cases}$ )
        with  $aux$  show  $?thesis$  .
      next
        case  $2$ 
        with  $A \ neg \ c$  have  $M \ b \ 0 \leq Le \ (-d)$  unfolding  $less\text{-eq } dbm\text{-le}\text{-def}$ 
        add neutral less
        by ( $auto \ elim! : dbm\text{-lt}\text{-cases}$ )
        from  $dbm\text{-le}'2[OF \ assms(2)][folded \ M(1)] \ this \ C2 \ C(1) \ not0$ 
have
   $[M]_{v,n} \subseteq \{u \in V. u \ c2 \leq -d\}$ 
by  $simp$ 
from  $beta\text{-interp}.\beta\text{-boundedness}\text{-le}'[OF - C(2) \ this] \ d(2)$  have
   $Approx_\beta ([M]_{v,n}) \subseteq \{u \in V. u \ c2 \leq -d\}$ 
by  $auto$ 
moreover
  { fix  $u$  assume  $u : u \in [M_R]_{v,n}$ 
    with  $C \ C2$  have
       $dbm\text{-entry}\text{-val } u \text{ None (Some } c2) (M_R \ 0 \ b)$ 
unfolding  $DBM\text{-zone}\text{-repr}\text{-def } DBM\text{-val}\text{-bounded}\text{-def}$  by  $auto$ 
with  $A \ 2$  have  $u \notin \{u \in V. u \ c2 \leq -d\}$  by  $auto$ 
    }
ultimately show  $?thesis$  using  $M_R(1) \ M(1)$  by  $auto$ 

```

```

    qed
  qed
} note neg-sum-2'' = this

{ fix a b assume A: (a,b) ∈ set (arcs i i ys)
  assume not0: a > 0 b > 0
  assume bounded: MR a 0 ≠ ∞ MR b 0 ≠ ∞
  assume lt: M a b < MR a b
  from clock-dest[OF A not0] obtain c1 c2 where
    C: v c1 = a v c2 = b c1 ∈ X c2 ∈ X and C2: a ≤ n b ≤ n
  by blast
  from C C2 clock-numbering(1,3) have C3: v' b = c2 v' a = c1
  unfolding v'-def by blast+
  with C C2 not0 bounded MR(4) obtain d :: int where *:
    - int (k c2) ≤ d ∧ d ≤ int (k c1) ∧ MR a b = Le d ∧ MR b a =
    Le (- d)
    ∨ - int (k c2) ≤ d - 1 ∧ d ≤ int (k c1) ∧ MR a b = Lt d ∧ MR
    b a = Lt (- d + 1)
  unfolding v'-def by force
  from * have ?thesis
  proof (standard, goal-cases)
  case 1
  with lt have M a b < Le d by auto
  then have M a b ≤ Lt d unfolding less less-eq dbm-le-def by
  (fastforce elim!: dbm-lt.cases)
  from dbm-lt'[OF assms(2)[folded M(1)] this C2 C(1,2) not0] have
    [M]v,n ⊆ {u ∈ V. u c1 - u c2 < d}
  .
  from beta-interp.β-boundedness-diag-lt'[OF - - C(3,4) this] 1
  have Approxβ ([M]v,n) ⊆ {u ∈ V. u c1 - u c2 < d} by auto
  moreover
  { fix u assume u: u ∈ [M]R v,n
    with C C2 have
      dbm-entry-val u (Some c1) (Some c2) (MR a b) dbm-entry-val
      u (Some c2) (Some c1) (MR b a)
    unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
    with 1 have u ∉ {u ∈ V. u c1 - u c2 < d} by auto
  }
  ultimately show ?thesis using MR(1) M(1) by auto
next
case 2
with lt have M a b ≠ ∞ by auto
with dbm-entry-int[OF this] M(3) ⟨a ≤ n⟩ ⟨b ≤ n⟩
obtain d' :: int where d': M a b = Le d' ∨ M a b = Lt d' by auto

```

```

then have  $M a b \leq Le (d - 1)$  using  $lt \ 2$ 
  apply (auto simp: less-eq dbm-le-def less)
  apply (cases rule: dbm-lt.cases)
    apply auto
    apply (rule dbm-lt.intros)
    apply (cases rule: dbm-lt.cases)
  by auto
with  $lt$  have  $M a b \leq Le (d - 1)$  by auto
from dbm-le'[OF assms(2)] [folded M(1)] this C2 C(1,2) not0] have
   $[M]_{v,n} \subseteq \{u \in V. u \ c1 - u \ c2 \leq d - 1\}$ 
.
from beta-interp. $\beta$ -boundedness-diag-le'[OF - - C(3,4) this] 2
have  $Approx_{\beta} ([M]_{v,n}) \subseteq \{u \in V. u \ c1 - u \ c2 \leq d - 1\}$  by auto
moreover
{ fix  $u$  assume  $u: u \in [M_R]_{v,n}$ 
  with C C2 have
    dbm-entry-val  $u$  (Some  $c2$ ) (Some  $c1$ ) ( $M_R \ b \ a$ )
    unfolding DBM-zone-repr-def DBM-val-bounded-def by auto
    with 2 have  $u \notin \{u \in V. u \ c1 - u \ c2 \leq d - 1\}$  by auto
  }
ultimately show ?thesis using  $M_R(1) \ M(1)$  by auto
qed
} note bounded = this

{ assume not-bounded:  $\forall (a,b) \in set \ (arcs \ i \ i \ ys). \ M \ a \ b < M_R \ a \ b$ 
 $\rightarrow M_R \ a \ 0 = \infty \vee M_R \ b \ 0 = \infty$ 
  have  $\exists \ y \ z \ zs. \ set \ zs \cup \{0, y, z\} = set \ (i \ \# \ ys) \wedge \ len \ ?M \ 0 \ 0 \ (y \ \#$ 
 $z \ \# \ zs) < Le \ 0 \wedge$ 
     $(\forall (a,b) \in set \ (arcs \ 0 \ 0 \ (y \ \# \ z \ \# \ zs)). \ M \ a \ b < M_R \ a \ b$ 
 $\rightarrow a = y \wedge b = z)$ 
     $\wedge M \ y \ z < M_R \ y \ z \wedge distinct \ (0 \ \# \ y \ \# \ z \ \# \ zs) \vee ?thesis$ 
  proof (cases  $ys$ )
    case Nil
      show ?thesis
    proof (cases  $M \ i \ i < M_R \ i \ i$ )
      case True
        then have  $?M \ i \ i = M \ i \ i$  by simp
        with Nil  $ys(1) \ xs(3)$  have *:  $M \ i \ i < 0$  by simp
        with neg-cycle-empty[OF cn-weak -  $\langle i \leq n \rangle$ , of [] M] have  $[M]_{v,n}$ 
= {} by auto
        with  $\langle Z \neq \{\} \rangle \ M(1)$  show ?thesis by auto
      next
        case False
          then have  $?M \ i \ i = M_R \ i \ i$  by (simp add: min-absorb2)

```

```

    with Nil ys(1) xs(3) have  $M_R i i < 0$  by simp
      with neg-cycle-empty[OF cn-weak -  $\langle i \leq n \rangle$ , of []  $M_R$ ] have
 $[M_R]_{v,n} = \{\}$  by auto
      with  $\langle R \neq \{\} \rangle M_R(1)$  show ?thesis by auto
    qed
  next
    case (Cons w ws)
    note ws = this
    show ?thesis
    proof (cases ws)
      case Nil
      with ws ys xs(3) have *:
 $?M i w + ?M w i < 0 \ ?M w i = M w i \longrightarrow ?M i w \neq M i w$ 
      (i, w)  $\in$  set (arcs i i ys)
      by auto
      have  $R \cap Approx_\beta Z = \{\}$ 
      proof (cases  $?M w i = M w i$ )
        case True
        with *(2) have  $?M i w = M_R i w$  unfolding min-def by auto
        with *(1) True have neg:  $M_R i w + M w i < 0$  by auto
        show ?thesis
        proof (cases  $i = 0$ )
          case True
          show ?thesis
          proof (cases  $w = 0$ )
            case True with  $0 \langle i = 0 \rangle *(3)$  show ?thesis by auto
          next
            case False with  $\langle i = 0 \rangle$  neg-sum-2'' *(3) neg show ?thesis
          by blast
        qed
      next
        case False
        show ?thesis
        proof (cases  $w = 0$ )
          case True with  $\langle i \neq 0 \rangle$  neg-sum-2' *(3) neg show ?thesis
          by blast
        next
          case False with  $\langle i \neq 0 \rangle$  neg-sum-2 *(3) neg show ?thesis
          by blast
        qed
      qed
    next
      case False
      show ?thesis
      proof (cases  $w = 0$ )
        case True with  $\langle i \neq 0 \rangle$  neg-sum-2' *(3) neg show ?thesis
        by blast
      next
        case False with  $\langle i \neq 0 \rangle$  neg-sum-2 *(3) neg show ?thesis
        by blast
      qed
    qed
  next
    case False
    have  $M_R w i < M w i$ 

```

```

proof (rule ccontr, goal-cases)
  case 1
  then have  $M_R w i \geq M w i$  by auto
  with False show False unfolding min-def by auto
qed
with one-M ws Nil have  $M i w < M_R i w$  by auto
then have  $?M i w = M i w$  unfolding min-def by auto
moreover from False  $*(2)$  have  $?M w i = M_R w i$  unfolding
min-def by auto
  ultimately have  $neg: M i w + M_R w i < 0$  using  $*(1)$  by
  auto

show ?thesis
proof (cases  $i = 0$ )
  case True
  show ?thesis
  proof (cases  $w = 0$ )
    case True with  $\langle i = 0 \rangle$   $*(3)$  show ?thesis by auto
  next
    case False with  $\langle i = 0 \rangle$  neg-sum-1''  $*(3)$  neg show ?thesis
by blast

  qed
next
  case False
  show ?thesis
  proof (cases  $w = 0$ )
    case True with  $\langle i \neq 0 \rangle$  neg-sum-1'  $*(3)$  neg show ?thesis
by blast

  next
    case False with  $\langle i \neq 0 \rangle$  neg-sum-1  $*(3)$  neg show ?thesis
by blast

  qed
qed
then show ?thesis by simp
next
  case zs: (Cons z zs)
  from one-M obtain  $a b$  where  $*$ :
     $(a,b) \in set (arcs i i ys)$   $M a b < M_R a b$ 
  by fastforce
  from cycle-rotate-3 [OF -  $*(1)$  ys(3)] ws cycle-closes obtain  $ws'$ 
where  $ws'$ :
   $len ?M i i ys = len ?M a a (b \# ws')$   $set (a \# b \# ws') = set$ 
 $(i \# ys)$ 
   $1 + length ws' = length ys$   $set (arcs i i ys) = set (arcs a a (b \#$ 

```

$ws')$ )  
**and** *successive*: *successive*  $(\lambda(a, b). ?M a b = M a b)$  (*arcs*  $a a$   
 $(b \# ws')$   $@ [(a, b)]$ )  
**by** *blast*  
**from** *successive* **have** *successive-arcs*:  
*successive*  $(\lambda(a, b). ?M a b = M a b)$  (*arcs*  $a b (b \# ws' @ [a])$ )  
**using** *arcs-decomp-tail* **by** *auto*  
**from**  $ws'(4)$  *one-M-R*  $*(2)$  **obtain**  $c d$  **where** **\*\***:  
 $(c, d) \in \text{set } (\text{arcs } a a (b \# ws'))$   $M c d > M_R c d$   $(a, b) \neq (c, d)$   
**by** *fastforce*  
**from** *card-distinct*[*of*  $a \# b \# ws'$ ] *distinct-card*[*of*  $i \# ys$ ]  $ws'(2, 3)$   
*distinct*  
**have** *distinct*: *distinct*  $(a \# b \# ws')$  **by** *simp*  
**from**  $ws' ws'(3)$  **have**  $ws' \neq []$  **by** *auto*  
**then** **obtain**  $z$  **where**  $z: ws' = zs @ [z]$  **by** (*metis ap-*  
*pend-butlast-last-id*)  
**then** **have**  $b \# ws' = (b \# zs) @ [z]$  **by** *simp*  
**with** *len-decomp*[*OF* *this*, *of*  $?M a a$ ] *arcs-decomp-tail* **have**  
*rotated*:  
 $\text{len } ?M a a (b \# ws') = \text{len } ?M z z (a \# b \# zs)$   
 $\text{set } (\text{arcs } a a (b \# ws')) = \text{set } (\text{arcs } z z (a \# b \# zs))$   
**by** (*auto simp add: comm*)  
**from**  $ys(1)$   $xs(3)$   $ws'(1)$  **have**  $\text{len } ?M a a (b \# ws') < 0$  **by** *auto*  
**from**  $ws'(2)$   $ys(2)$   $\langle i \leq n \rangle z$  **have** *n-bounds*:  $a \leq n$   $b \leq n$   $\text{set } ws'$   
 $\subseteq \{0..n\}$   $z \leq n$  **by** *auto*  
**from**  $*$  **have** *a-b*:  $?M a b = M a b$  **by** *simp*  
**from** *successive* *successive-split*[*of* - *arcs*  $a z (b \# zs)$   $[(z, a), (a, b)]$ ]  
**have** *first*: *successive*  $(\lambda(a, b). ?M a b = M a b)$  (*arcs*  $a z (b \#$   
 $zs)$ ) **and**  
*last-two*: *successive*  $(\lambda(a, b). ?M a b = M a b)$   $[(z, a), (a, b)]$   
**using** *arcs-decomp-tail*  $z$  **by** *auto*  
**from**  $*$  *not-bounded* **have** *not-bounded'*:  $M_R a 0 = \infty \vee M_R b 0$   
 $= \infty$  **by** *auto*  
**from** *this(1)* **have**  $z = 0$   
**proof**  
**assume** *inf*:  $M_R b 0 = \infty$   
**from** *a-b* *successive* **obtain**  $z$  **where**  $z: (b, z) \in \text{set } (\text{arcs } b a$   
 $ws')$   $?M b z \neq M b z$   
**by** (*cases*  $ws'$ ) *auto*  
**then** **have**  $?M b z = M_R b z$  **by** (*meson min-def*)  
**from** *arcs-distinct2*[*OF* - - -  $z(1)$ ] *distinct* **have**  $b \neq z$  **by** *auto*  
**from**  $z$  *n-bounds* **have**  $z \leq n$   
**apply** (*induction*  $ws'$  *arbitrary*:  $b$ )  
**apply** *auto*[]

```

    apply (rename-tac ws' b)
    apply (case-tac ws')
    apply auto
  done
  have  $M_R b z = \infty$ 
  proof (cases z = 0)
    case True
      with inf show ?thesis by auto
    next
      case False
        with inf  $M_R(2)$   $\langle b \neq z \rangle \langle z \leq n \rangle \langle b \leq n \rangle$  show ?thesis by
blast
  qed
  with  $\langle ?M b z = M_R b z \rangle$  have len  $?M b a ws' = \infty$  by (auto
intro: len-inf-elem[OF z(1)])
  then have  $\infty = \text{len } ?M a a (b \# ws')$  by simp
  with  $\langle \text{len } ?M a a - < 0 \rangle$  show ?thesis by auto
next
  assume inf:  $M_R a 0 = \infty$ 
  show z = 0
  proof (rule ccontr)
    assume z  $\neq 0$ 
    with last-two a-b have  $?M z a = M_R z a$  by (auto simp:
min-def)
    from distinct z have  $a \neq z$  by auto
    with  $\langle z \neq 0 \rangle \langle a \leq n \rangle \langle z \leq n \rangle M_R(2)$  inf have  $M_R z a = \infty$ 
by blast
    with  $\langle ?M z a = M_R z a \rangle$  have len  $?M z z (a \# b \# zs) = \infty$ 
by (auto intro: len-inf-elem)
    with  $\langle \text{len } ?M a a - < 0 \rangle$  rotated show False by auto
  qed
  qed
  { fix c d assume A:  $(c, d) \in \text{set } (\text{arcs } 0 0 (a \# b \# zs))$   $M c d$ 
<  $M_R c d$ 
    then have *:  $?M c d = M c d$  by simp
    from rotated(2) A  $\langle z = 0 \rangle$  not-bounded ws'(4) have **:  $M_R c$ 
0 =  $\infty \vee M_R d 0 = \infty$  by auto
    { assume inf:  $M_R c 0 = \infty$ 
      fix x assume x:  $(x, c) \in \text{set } (\text{arcs } a 0 (b \# zs))$   $?M x c \neq M$ 
x c
      from x(2) have  $?M x c = M_R x c$  unfolding min-def by
auto
      from arcs-elem[OF x(1)] z  $\langle z = 0 \rangle$  have
        x  $\in \text{set } (a \# b \# ws')$  c  $\in \text{set } (a \# b \# ws')$ 

```

```

    by auto
    with n-bounds have  $x \leq n$   $c \leq n$  by auto
    have  $x = 0$ 
    proof (rule ccontr)
      assume  $x \neq 0$ 
      from distinct z arcs-distinct1[OF - - - x(1)]  $\langle z = 0 \rangle$  have
x  $\neq c$  by auto
      with  $\langle x \neq 0 \rangle$   $\langle c \leq n \rangle$   $\langle x \leq n \rangle$   $M_R(2)$  inf have  $M_R x c =$ 
 $\infty$  by blast
      with  $\langle ?M x c = M_R x c \rangle$  have
        len  $?M a 0$  ( $b \# zs$ ) =  $\infty$ 
      by (fastforce intro: len-inf-elem[OF x(1)])
      with  $\langle z = 0 \rangle$  have len  $?M z z$  ( $a \# b \# zs$ ) =  $\infty$  by auto
      with  $\langle len ?M a a - < 0 \rangle$  rotated show False by auto
    qed
    with arcs-distinct-dest1[OF - x(1), of z] z distinct x  $\langle z = 0 \rangle$ 
have False by auto
  } note c-0-inf = this
  have  $a = c \wedge b = d$ 
  proof (cases (c, d) = (0, a))
    case True
      with last-two  $\langle z = 0 \rangle$  * a-b have False by auto
      then show ?thesis by simp
  next
    case False
      show ?thesis
      proof (rule ccontr, goal-cases)
        case 1
          with False A(1) have ***:  $(c, d) \in set (arcs b 0 zs)$  by auto
          from successive z  $\langle z = 0 \rangle$  have
            successive  $(\lambda(a, b). ?M a b = M a b)$  ([[a, b]] @ arcs b 0 zs
@ [(0, a), (a, b)])
          by (simp add: arcs-decomp)
          then have ****: successive  $(\lambda(a, b). ?M a b = M a b)$  (arcs
b 0 zs)
          using successive-split[of - [(a, b)] arcs b 0 zs @ [(0, a), (a,
b)]]
            successive-split[of - arcs b 0 zs [(0, a), (a, b)]]
          by auto
          from successive-predecessor[OF *** - this] successive z
          obtain x where  $x: (x, c) \in set (arcs a 0 (b \# zs))$   $?M x c$ 
 $\neq M x c$ 
          proof (cases c = b)
            case False

```

```

then have  $zs \neq []$  using *** by auto
from successive-predecessor[OF *** False **** - this] *
obtain  $x$  where  $x$ :
  ( $zs = [c] \wedge x = b \vee (\exists ys. zs = c \# d \# ys \wedge x = b)$ 
    $\vee (\exists ys. zs = ys @ [x, c] \wedge d = 0) \vee (\exists ys ws. zs = ys$ 
@  $x \# c \# d \# ws)$ )
  ? $M$   $x$   $c \neq M$   $x$   $c$ 
by blast+
from this(1) have  $(x, c) \in \text{set } (\text{arcs } a \ 0 \ (b \# zs))$  using
arcs-decomp by auto
with  $x$ (2) show ?thesis by (auto intro: that)
next
case True
have ****: successive  $(\lambda(a, b). ?M \ a \ b = M \ a \ b)$   $(\text{arcs } a \ 0$ 
( $b \# zs$ ))
using first  $\langle z = 0 \rangle$  arcs-decomp successive-arcs  $z$  by auto
show ?thesis
proof (cases  $zs$ )
  case Nil
    with **** True *** * show ?thesis by (auto intro: that)
  next
    case (Cons  $u$   $us$ )
      with *** True distinct  $z \ \langle z = 0 \rangle$  have distinct  $(b \# u \#$ 
us @  $[0])$  by auto
      from arcs-distinct-fix[OF this] *** True Cons have  $d =$ 
u by auto
      with **** * Cons True show ?thesis by (auto intro: that)
    qed
  qed
show False
proof (cases  $d = 0$ )
  case True
    from ** show False
  proof
    assume  $M_R \ c \ 0 = \infty$  from c-0-inf[OF this  $x$ ] show
False .
  next
    assume  $M_R \ d \ 0 = \infty$  with  $\langle d = 0 \rangle$   $M_R$ (3) show False
by auto
  qed
next
  case False with *** have  $zs \neq []$  by auto
  from successive-successor[OF  $\langle (c, d) \in \text{set } (\text{arcs } b \ 0 \ zs) \rangle$ 
False **** - this] *

```

```

obtain  $e$  where
  ( $zs = [d] \wedge e = 0 \vee (\exists ys. zs = d \# e \# ys) \vee (\exists ys. zs$ 
 $= ys @ [c, d] \wedge e = 0)$ 
   $\vee (\exists ys ws. zs = ys @ c \# d \# e \# ws))$   $?M d e \neq M d e$ 
by blast
  then have  $e: (d, e) \in set (arcs b 0 zs)$   $?M d e \neq M d e$ 
using arcs-decomp by auto
from ** show False
proof
  assume  $inf: M_R d 0 = \infty$ 
  from  $e$  have  $?M d e = M_R d e$  by (meson min-def)
  from arcs-distinct2[OF - - - e(1)]  $z \langle z = 0 \rangle$  distinct
have  $d \neq e$  by auto
  from  $z$  n-bounds have  $set zs \subseteq \{0..n\}$  by auto
  with  $e$  have  $e \leq n$ 
  apply (induction zs arbitrary: d)
  apply auto
  apply (case-tac zs)
  apply auto
done
from n-bounds z arcs-elem(2)[OF A(1)] have  $d \leq n$  by
auto
  have  $M_R d e = \infty$ 
  proof (cases e = 0)
    case True
      with  $inf$  show  $?thesis$  by auto
    next
      case False
      with  $inf M_R(2)$   $\langle d \neq e \rangle \langle e \leq n \rangle \langle d \leq n \rangle$  show  $?thesis$ 
by blast
  qed
  with  $\langle ?M d e = M_R d e \rangle$  have  $len ?M b 0 zs = \infty$  by
(auto intro: len-inf-elim[OF e(1)])
  with  $\langle z = 0 \rangle$  rotated have  $\infty = len ?M a a (b \# ws')$ 
by simp
  with  $\langle len ?M a a - < 0 \rangle$  show  $?thesis$  by auto
next
  assume  $M_R c 0 = \infty$  from c-0-inf[OF this x] show
False .
  qed
qed
qed
qed
}

```

**then have**  $\forall (c, d) \in \text{set } (\text{arcs } 0 \ 0 \ (a \ \# \ b \ \# \ zs)). \ M \ c \ d < M_R \ c$   
 $d \longrightarrow c = a \wedge d = b$   
**by** *blast*  
**moreover from**  $ys(1) \ xs(3)$  **have**  $\text{len } ?M \ i \ i \ ys < Le \ 0$  **unfolding**  
*neutral* **by** *auto*  
**moreover with** *rotated*  $ws'(1)$  **have**  $\text{len } ?M \ z \ z \ (a \ \# \ b \ \# \ zs) <$   
 $Le \ 0$  **by** *auto*  
**moreover from**  $\langle z = 0 \rangle \ z \ ws'(2)$  **have**  $\text{set } zs \cup \{0, a, b\} = \text{set}$   
 $(i \ \# \ ys)$  **by** *auto*  
**moreover from**  $\langle z = 0 \rangle \ \text{distinct } z$  **have**  $\text{distinct } (0 \ \# \ a \ \# \ b \ \#$   
 $zs)$  **by** *auto*  
**ultimately show** *thesis* **using**  $\langle z = 0 \rangle \ \langle M \ a \ b < M_R \ a \ b \rangle$  **by**  
*blast*  
**qed**  
**qed note**  $* = \text{this}$   
**{ assume**  $\neg ?thesis$   
**with**  $*$  **obtain**  $y \ z \ zs$  **where**  $*$ :  
 $\text{set } zs \cup \{0, y, z\} = \text{set } (i \ \# \ ys) \ \text{len } ?M \ 0 \ 0 \ (y \ \# \ z \ \# \ zs) < Le \ 0$   
 $\forall (a, b) \in \text{set } (\text{arcs } 0 \ 0 \ (y \ \# \ z \ \# \ zs)). \ M \ a \ b < M_R \ a \ b \longrightarrow a = y$   
 $\wedge b = z \ M \ y \ z < M_R \ y \ z$   
**and**  $\text{distinct}' : \text{distinct } (0 \ \# \ y \ \# \ z \ \# \ zs)$   
**by** *blast*  
**then have**  $y \neq 0 \ z \neq 0$  **by** *auto*  
**let**  $?r = \text{len } M_R \ z \ 0 \ zs$   
**have**  $\forall (a, b) \in \text{set } (\text{arcs } z \ 0 \ zs). \ ?M \ a \ b = M_R \ a \ b$   
**proof** (*safe, goal-cases*)  
**case**  $A : (1 \ a \ b)$   
**have**  $M_R \ a \ b \leq M \ a \ b$   
**proof** (*rule ccontr, goal-cases*)  
**case**  $1$   
**with**  $*(3) \ A$  **have**  $a = y \ b = z$  **by** *auto*  
**with**  $A \ \text{distinct}' \ \text{arcs-distinct3}[OF - A, \text{of } y]$  **show** *False* **by**  
*auto*  
**qed**  
**then show**  $?case$  **by** (*simp add: min-def*)  
**qed**  
**then have**  $r : \text{len } ?M \ z \ 0 \ zs = ?r$  **by** (*induction zs arbitrary: z*)  
*auto*  
**with**  $*(2)$  **have**  $** : ?M \ 0 \ y + (?M \ y \ z + ?r) < Le \ 0$  **by** *simp*  
**from**  $M_R(1) \ \langle R \neq \{\} \rangle$  **obtain**  $u$  **where**  $u : \text{DBM-val-bounded } v \ u$   
 $M_R \ n$   
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**from**  $*(1) \ \langle i \leq n \rangle \ \langle \text{set } ys \subseteq \cdot \rangle$  **have**  $y \leq n \ z \leq n$  **by** *fastforce+*  
**from**  $*(1) \ ys(2,4)$  **have**  $\text{set } zs \subseteq \{0 \ ..n\}$  **by** *auto*

**from**  $\langle y \leq n \rangle \langle z \leq n \rangle$  *clock-numbering*(2)  $\langle y \neq 0 \rangle \langle z \neq 0 \rangle$  **obtain**  
*c1 c2* **where** *C*:  
 $c1 \in X \ c2 \in X \ v \ c1 = y \ v \ c2 = z$   
**by** *blast+*  
**with** *clock-numbering*(1,3) **have** *C2*:  $v' \ y = c1 \ v' \ z = c2$  **unfolding**  
*v'-def* **by** *auto*  
**with** *C* **have**  $v \ (v' \ z) = z$  **by** *auto*  
**with** *DBM-val-bounded-len'1*[*OF* *u*, *of* *zs* *v' z*] **have** *dbm-entry-val*  
*u* (*Some* (*v' z*)) *None* *?r*  
**using**  $\langle z \leq n \rangle$  *clock-numbering*(2)  $\langle \text{set } zs \subseteq \rightarrow \text{distinct}' \rangle$  **by** *force*  
**from** *len-inf-elem* \*\* **have** *tl-not-inf*:  $\forall (a, b) \in \text{set} \ (\text{arcs } z \ 0 \ zs)$ .  $M_R$   
 $a \ b \neq \infty$  **by** *fastforce*  
**with**  $M_R(7)$  *len-int-dbm-closed* **have** *get-const*  $?r \in \mathbb{Z} \wedge ?r \neq \infty$   
**by** *blast*  
**then obtain**  $r :: \text{int}$  **where**  $r' : ?r = \text{Le } r \vee ?r = \text{Lt } r$  **using**  
*Ints-cases* **by** (*cases* *?r*) *auto*  
**from**  $r' \ \langle \text{dbm-entry-val} \ - \ - \ - \rightarrow \ C \ C2 \rangle$  **have**  $le : u \ (v' \ z) \leq r$  **by**  
*fastforce*  
**from** *arcs-ex-head* **obtain**  $z'$  **where**  $(z, z') \in \text{set} \ (\text{arcs } z \ 0 \ zs)$  **by**  
*blast*  
**then have**  $z'$ :  
 $(z, z') \in \text{set} \ (\text{arcs } 0 \ 0 \ (y \ \# \ z \ \# \ zs)) \ (z, z') \in \text{set} \ (\text{arcs } z \ 0 \ zs)$   
**by** *auto*  
**have**  $M_R \ z \ 0 \neq \infty$   
**proof** (*rule ccontr*, *goal-cases*)  
**case** 1  
**then have** *inf*:  $M_R \ z \ 0 = \infty$  **by** *auto*  
**have**  $M_R \ z \ z' = \infty$   
**proof** (*cases*  $z' = 0$ )  
**case** *True*  
**with** 1 **show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**from** *arcs-elem*[*OF*  $z'(1)$ ]  $*(1) \ \langle i \leq n \rangle \ \langle \text{set } ys \subseteq \rightarrow \rangle$  **have**  $z' \leq$   
*n* **by** *fastforce*  
**moreover from** *distinct'*  $*(1)$  *arcs-distinct1*[*OF*  $z'(1)$ ]  
**have**  $z \neq z'$  **by** *auto*  
**ultimately show** *?thesis* **using**  $M_R(2) \ \langle z \leq n \rangle$  *False inf* **by**  
*blast*  
**qed**  
**with** *tl-not-inf*  $z'(2)$  **show** *False* **by** *auto*  
**qed**  
**with**  $M_R(5) \ \langle z \neq 0 \rangle \ \langle z \leq n \rangle$  **obtain**  $d :: \text{int}$  **where** *d*:  
 $M_R \ z \ 0 = \text{Le } d \wedge M_R \ 0 \ z = \text{Le } (-d) \vee M_R \ z \ 0 = \text{Lt } d \wedge M_R \ 0$

$z = Lt (-d + 1)$   
 $d \leq k (v' z) \ 0 \leq d$   
**unfolding**  $v'$ -def by auto

Needs property that len of integral dbm entries is integral and definition of  $M-R$

**from** this (1) **have**  $rr: ?r \geq M_R z \ 0$   
**proof** (standard, goal-cases)  
**case** A: 1  
**with**  $u \langle z \leq n \rangle \ C \ C2$  **have**  $*$ :  $- u (v' z) \leq -d$  **unfolding**  
*DBM-val-bounded-def* **by** fastforce  
**from**  $r'$  **show** ?case  
**proof** (standard, goal-cases)  
**case** 1  
**with**  $le \ * \ A$  **show** ?case **unfolding** less-eq dbm-le-def **by**  
*fastforce*  
**next**  
**case** 2  
**with**  $\langle dbm-entry-val \ - \ - \ - \ \rangle \ C \ C2$  **have**  $u (v' z) < r$  **by** fastforce  
**with**  $*$  **have**  $r > d$  **by** auto  
**with** A 2 **show** ?case **unfolding** less-eq dbm-le-def **by** fastforce  
**qed**  
**next**  
**case** A: 2  
**with**  $u \langle z \leq n \rangle \ C \ C2$  **have**  $*$ :  $- u (v' z) < -d + 1$  **unfolding**  
*DBM-val-bounded-def* **by** fastforce  
**from**  $r'$  **show** ?case  
**proof** (standard, goal-cases)  
**case** 1  
**with**  $le \ * \ A$  **show** ?case **unfolding** less-eq dbm-le-def **by**  
*fastforce*  
**next**  
**case** 2  
**with**  $\langle dbm-entry-val \ - \ - \ - \ \rangle \ C \ C2$  **have**  $u (v' z) \leq r$  **by** fastforce  
**with**  $*$  **have**  $r \geq d$  **by** auto  
**with** A 2 **show** ?case **unfolding** less-eq dbm-le-def **by** fastforce  
**qed**  
**qed**  
**with**  $*(3) \ \langle y \neq 0 \rangle$  **have**  $M \ 0 \ y \geq M_R \ 0 \ y$  **by** fastforce  
**then** **have**  $?M \ 0 \ y = M_R \ 0 \ y$  **by** (simp add: min.absorb2)  
**moreover** **from**  $*(4)$  **have**  $?M \ y \ z = M \ y \ z$  **unfolding** min-def  
**by** auto  
**ultimately** **have**  $**$ :  $M_R \ 0 \ y + (M \ y \ z + M_R \ z \ 0) < Le \ 0$   
**using**  $**$  add-mono-right[OF add-mono-right[OF rr], of  $M_R \ 0 \ y \ M$

$y z]$  **by** *simp*  
**from** **\*\* have** *not-inf*:  $M_R 0 y \neq \infty \ M y z \neq \infty \ M_R z 0 \neq \infty$  **by**  
*auto*  
**from**  $M_R(6) \langle y \neq 0 \rangle \langle y \leq n \rangle$  **obtain**  $c :: int$  **where**  $c$ :  
 $M_R 0 y = Le\ c \vee M_R 0 y = Lt\ c - k\ (v'\ y) \leq c \ c \leq 0$   
**unfolding** *v'-def* **by** *auto*  
**have** *?thesis*  
**proof** (*cases*  $M_R 0 y + M_R z 0 = Lt\ (c + d)$ )  
**case** *True*  
**from** **\*\* have**  $(M_R 0 y + M_R z 0) + M y z < Le\ 0$  **using** *comm*  
*add.assoc* **by** *metis*  
**with** *True* **have** **\*\***:  $Lt\ (c + d) + M y z < Le\ 0$  **by** *simp*  
**then** **have**  $M y z \leq Le\ (-\ (c + d))$  **unfolding** *less less-eq*  
*dbm-le-def add*  
**by** (*cases*  $M y z$ ) (*fastforce elim!*: *dbm-lt.cases*) +  
**from** *dbm-le'[OF assms(2)]folded M(1)* *this*  $\langle y \leq n \rangle \langle z \leq n \rangle$   
 $C(3,4)] \langle y \neq 0 \rangle \langle z \neq 0 \rangle M$   
**have** *subs*:  $Z \subseteq \{u \in V. u\ c1 - u\ c2 \leq -\ (c + d)\}$  **by** *blast*  
**with**  $c\ d$  **have**  $-k\ (v'\ z) \leq -\ (c + d) - (c + d) \leq k\ (v'\ y)$  **by**  
*auto*  
**with** *beta-interp.β-boundedness-diag-le'[OF - - C(1,2) subs]*  $C2$   
**have**  
 $Approx_\beta\ Z \subseteq \{u \in V. u\ c1 - u\ c2 \leq -\ (c + d)\}$   
**by** *auto*  
**moreover**  
**{** **fix**  $u$  **assume**  $u: u \in R$   
**with**  $C \langle y \leq n \rangle \langle z \leq n \rangle M_R(1)$  **have**  
 $dbm-entry-val\ u\ (Some\ c2)\ None\ (M_R\ z\ 0)\ dbm-entry-val\ u$   
 $None\ (Some\ c1)\ (M_R\ 0\ y)$   
**unfolding** *DBM-zone-repr-def DBM-val-bounded-def* **by** *auto*  
**with** *True c d(1)* **have**  $u \notin \{u \in V. u\ c1 - u\ c2 \leq -\ (c +$   
 $d)\}$  **unfolding** *add* **by** *auto*  
**}**  
**ultimately show** *?thesis* **by** *blast*  
**next**  
**case** *False*  
**with**  $c\ d$  **have**  $M_R 0 y + M_R z 0 = Le\ (c + d)$  **unfolding** *add*  
**by** *fastforce*  
**moreover from** **\*\* have**  $(M_R 0 y + M_R z 0) + M y z < Le\ 0$   
**using** *comm add.assoc* **by** *metis*  
**ultimately have** **\*\***:  $Le\ (c + d) + M y z < Le\ 0$  **by** *simp*  
**then** **have**  $M y z \leq Lt\ (-\ (c + d))$  **unfolding** *less less-eq*  
*dbm-le-def add*  
**by** (*cases*  $M y z$ ) (*fastforce elim!*: *dbm-lt.cases*) +

**from**  $dbm-lt'[OF\ assms(2)]\text{folded } M(1)$  **this**  $\langle y \leq n \rangle \langle z \leq n \rangle$   
 $C(3,4)$   $\langle y \neq 0 \rangle \langle z \neq 0 \rangle M$   
**have**  $subs: Z \subseteq \{u \in V. u\ c1 - u\ c2 < - (c + d)\}$  **by** *auto*  
**from**  $c\ d(2-)\ C2$  **have**  $-k\ c2 \leq - (c + d) - (c + d) \leq k\ c1$   
**by** *auto*  
**from**  $beta\text{-interp.}\beta\text{-boundedness-diag-lt}'[OF\ this\ C(1,2)\ subs]$  **have**  
 $Approx_\beta Z \subseteq \{u \in V. u\ c1 - u\ c2 < - (c + d)\}$   
 $\cdot$   
**moreover**  
**{** **fix**  $u$  **assume**  $u: u \in R$   
**with**  $C\ \langle y \leq n \rangle \langle z \leq n \rangle M_R(1)$  **have**  
 $dbm\text{-entry-}val\ u\ (Some\ c2)\ None\ (M_R\ z\ 0)\ dbm\text{-entry-}val\ u$   
 $None\ (Some\ c1)\ (M_R\ 0\ y)$   
**unfolding**  $DBM\text{-zone-repr-def}\ DBM\text{-val-bounded-def}$  **by** *auto*  
**with**  $c\ d(1)$  **have**  $u \notin \{u \in V. u\ c1 - u\ c2 < - (c + d)\}$  **by**  
*auto*  
**}**  
**ultimately show** *?thesis* **by** *auto*  
**qed**  
**}** **then have** *?thesis* **by** *auto*  
**}**  
**with**  $bounded\ 0\ bounded\text{-zero-}1\ bounded\text{-zero-}2$  **show** *?thesis* **by** *blast*  
**qed**  
**qed**  
**qed**

### 6.3 Nice Corollaries of Bouyer's Theorem

**lemma**  $\mathcal{R}\text{-}V: \bigcup \mathcal{R} = V$  **unfolding**  $V\text{-def}\ \mathcal{R}\text{-def}$  **using**  $region\text{-cover}[of\ X - k]$  **by** *auto*

**lemma**  $regions\text{-beta-}V: R \in \mathcal{R}_\beta \implies R \subseteq V$  **unfolding**  $V\text{-def}\ \mathcal{R}_\beta\text{-def}$  **by** *auto*

**lemma**  $apx\text{-}V: Z \subseteq V \implies Approx_\beta Z \subseteq V$

**proof** (*goal-cases*)

**case**  $1$

**from**  $beta\text{-interp.apx-in}[OF\ 1]$  **obtain**  $U$  **where**  $Approx_\beta Z = \bigcup U$   $U \subseteq \mathcal{R}_\beta$  **by** *auto*

**with**  $regions\text{-beta-}V$  **show** *?thesis* **by** *auto*

**qed**

**corollary**  $approx\text{-}\beta\text{-closure-}\alpha:$

**assumes**  $Z \subseteq V$  **vabstr**  $Z\ M$

**shows**  $Approx_\beta Z \subseteq Closure_\alpha Z$   
**proof** –  
**note**  $T = region\text{-}zone\text{-}intersect\text{-}empty\text{-}approx\text{-}correct[OF - assms(1) - assms(2-)]$   
**have**  $-\bigcup\{R \in \mathcal{R}. R \cap Z \neq \{\}\} = \bigcup\{R \in \mathcal{R}. R \cap Z = \{\}\} \cup -V$   
**proof** (*safe, goal-cases*)  
**case 1 with  $\mathcal{R}\text{-}V$  show *False* by *fast***  
**next**  
**case 2 then show ?case using *alpha-interp.valid-regions-distinct-spec***  
**by *fastforce***  
**next**  
**case 3 then show ?case using *\mathcal{R}\text{-}V* unfolding *V-def* by *blast***  
**qed**  
**with  $T$  *apx-V[OF assms(1)]* have  $Approx_\beta Z \cap -\bigcup\{R \in \mathcal{R}. R \cap Z \neq \{\}\} = \{\}$  by *auto***  
**then show ?thesis unfolding *alpha-interp.cla-def* by *blast***  
**qed**

**corollary *approx-beta-closure-alpha'*:**  $Z \in V' \implies Approx_\beta Z \subseteq Closure_\alpha Z$   
**using *approx-beta-closure-alpha* unfolding *V'-def* by *auto***

We could prove this more directly too (without using  $Closure_\alpha Z$ ), obviously

**lemma *apx-empty-iff*:**  
**assumes**  $Z \subseteq V$  **vabstr**  $Z M$   
**shows**  $Z = \{\} \longleftrightarrow Approx_\beta Z = \{\}$   
**using *alpha-interp.cla-empty-iff[OF assms(1)] approx-beta-closure-alpha[OF assms] beta-interp.apx-subset***  
**by *auto***

**lemma *apx-empty-iff'*:**  
**assumes**  $Z \in V'$  **shows**  $Z = \{\} \longleftrightarrow Approx_\beta Z = \{\}$   
**using *apx-empty-iff assms* unfolding *V'-def* by *force***

**lemma *apx-V'*:**  
**assumes**  $Z \subseteq V$  **shows**  $Approx_\beta Z \in V'$   
**proof** (*cases  $Z = \{\}$* )  
**case *True***  
**with *beta-interp.apx-empty beta-interp.empty-zone-dbm* show ?thesis unfolding *V'-def* *neutral* by *auto***  
**next**  
**case *False***  
**then have *non-empty*:  $Approx_\beta Z \neq \{\}$  using *beta-interp.apx-subset* by *blast***  
**from *beta-interp.apx-in[OF assms]* obtain  $U M$  where  $*$ :**

$Approx_\beta Z = \bigcup U \ U \subseteq \mathcal{R}_\beta Z \subseteq Approx_\beta Z \text{ vabstr } (Approx_\beta Z) M$   
 by *blast*  
 moreover from *\* beta-interp.ℛ-union* have  $\bigcup U \subseteq V$  by *blast*  
 ultimately show *?thesis* using *\*(1,4) unfolding V'-def* by *auto*  
 qed

end

**lemma** *valid-abstraction-pairsD*:

$\forall (x, m) \in \text{Timed-Automata.clkp-set } A. x \in X \wedge m \in \mathbb{N}$  if *valid-abstraction*  
 $A \ X \ k$

using *that*

apply *cases*

unfolding *clkp-set-def Timed-Automata.clkp-set-def*

unfolding *collect-clki-def Timed-Automata.collect-clki-def*

unfolding *collect-clkt-def Timed-Automata.collect-clkt-def*

by *blast*

## 6.4 A New Zone Semantics Abstracting with $Approx_\beta$

**locale** *Regions* =

*Regions-defs*  $X \ v \ n$  for  $X$  and  $v :: 'c \Rightarrow \text{nat}$  and  $n :: \text{nat} +$

fixes  $k :: 's \Rightarrow 'c \Rightarrow \text{nat}$  and *not-in-X*

assumes *finite*: *finite X*

assumes *clock-numbering*:

$\text{clock-numbering}' \ v \ n \ \forall k \leq n. k > 0 \longrightarrow (\exists c \in X. v \ c = k) \ \forall c \in X. v$   
 $c \leq n$

assumes *not-in-X*: *not-in-X*  $\notin X$

assumes *non-empty*:  $X \neq \{\}$

**begin**

**definition** *ℛ-def*:  $\mathcal{R} \ l \equiv \{\text{Regions.region } X \ I \ r \mid I \ r. \text{Regions.valid-region } X \ (k \ l) \ I \ r\}$

**definition** *ℛ<sub>β</sub>-def*:

$\mathcal{R}_\beta \ l \equiv \{\text{Regions-Beta.region } X \ I \ J \ r \mid I \ J \ r. \text{Regions-Beta.valid-region } X \ (k \ l) \ I \ J \ r\}$

**sublocale**

*AlphaClosure X k ℛ* by (*unfold-locales*) (*auto simp: finite ℛ-def V-def*)

**abbreviation**  $Approx_\beta \ l \ Z \equiv \text{Beta-Regions}'.Approx_\beta \ X \ (k \ l) \ v \ n \ \text{not-in-X}$   
 $Z$

### 6.4.1 Single Step

**inductive** *step-z-beta* ::

$(\text{'a}, \text{'c}, t, \text{'s}) \text{ta} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{'a} \text{action} \Rightarrow \text{'s} \Rightarrow (\text{'c}, t) \text{zone} \Rightarrow \text{bool}$

$(\langle \cdot \vdash \langle \cdot, \cdot \rangle \rightsquigarrow_{\beta(\cdot)} \langle \cdot, \cdot \rangle \rangle [61,61,61,61] \ 61)$

**where**

*step-beta*:  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \Longrightarrow A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', \text{Approx}_{\beta} l' Z' \rangle$

**inductive-cases**[*elim!*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{\beta(a)} \langle l', u' \rangle$

**declare** *step-z-beta.intros*[*intro*]

**context**

**fixes**  $l' :: \text{'s}$

**begin**

**interpretation** *regions*: *Regions-global* - - -  $k \ l'$

**by** *standard* (*rule finite clock-numbering not-in-X non-empty*) $+$

**lemma** *step-z-V'*:

**assumes**  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle$  *valid-abstraction*  $A \ X \ k \ \forall c \in \text{clk-set } A. \ v \ c \leq n \ Z \in V'$

**shows**  $Z' \in V'$

**proof** –

**from** *assms(3) clock-numbering have numbering: global-clock-numbering*  
 $A \ v \ n$  **by** *metis*

**from** *assms(4) obtain M where M:*

$Z \subseteq V \ Z = [M]_{v,n} \text{dbm-int } M \ n$

**unfolding** *V'-def by auto*

**from** *valid-abstraction-pairsD[OF assms(2)] have*  $\forall (x, m) \in \text{Timed-Automata.clkp-set}$   
 $A. \ m \in \mathbb{N}$

**by** *blast*

**from** *step-z-V[OF assms(1) M(1)] M(2) assms(1) step-z-dbm-DBM[OF*  
*- numbering]*

*step-z-dbm-preserves-int[OF - numbering this M(3)]*

**obtain**  $M'$  **where**  $M': Z' \subseteq V \ Z' = [M']_{v,n} \text{dbm-int } M' \ n$  **by** *metis*

**then show** *?thesis unfolding V'-def by blast*

**qed**

**lemma** *step-z-alpha-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z' \rangle \Longrightarrow \text{valid-abstraction } A \ X \ k \Longrightarrow \forall c \in \text{clk-set } A. \ v$

$c \leq n \implies Z \in V'$   
 $\implies Z' \neq \{\} \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z'' \rangle \wedge Z'' \neq \{\}$   
**apply** (*induction*  $l' \equiv l'$   $Z'$  *rule: step-z-beta.induct*)  
**apply** (*frule* *step-z-V'*)  
**apply** *assumption+*  
**apply** (*rotate-tac* 5)  
**apply** (*drule* *regions.apx-empty-iff'*)  
**by** *blast*

**lemma** *step-z-alpha-complete:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A. v \ c$   
 $\leq n \implies Z \in V'$   
 $\implies Z' \neq \{\} \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$   
**apply** (*frule* *step-z-V'*)  
**apply** *assumption+*  
**apply** (*rotate-tac* 4)  
**apply** (*drule* *regions.apx-empty-iff'*)  
**by** *blast*

**lemma** *alpha-beta-step:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A.$   
 $v \ c \leq n \implies Z \in V'$   
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$   
**apply** (*induction*  $l' \equiv l'$   $Z'$  *rule: step-z-beta.induct*)  
**apply** (*frule* *step-z-V'*)  
**apply** *assumption+*  
**apply** (*rotate-tac* 4)  
**apply** (*drule* *regions.approx-beta-closure-alpha'*)  
**apply** *auto*  
**done**

**lemma** *alpha-beta-step':*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A.$   
 $v \ c \leq n \implies Z \in V' \implies W \subseteq V$   
 $\implies Z \subseteq W \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\alpha(a)} \langle l', W' \rangle \wedge Z' \subseteq W'$   
**proof** (*induction*  $l' \equiv l'$   $Z'$  *rule: step-z-beta.induct*)  
**case** (*step-beta*  $A \ l \ Z \ a \ Z'$ )  
**from** *step-z-mono*[*OF* *step-beta*(1,6)] **obtain**  $W'$  **where**  $W'$ :  
 $A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle \wedge Z' \subseteq W'$   
**by** *blast*  
**from** *regions.approx-beta-closure-alpha'*[*OF* *step-z-V'*[*OF* *step-beta*(1-4)]]  
*regions.alpha-interp.cla-mono*[*OF* *this*(2)] *this*(1)  
**show** ?*case* **by** *auto*

qed

**lemma** *apx-mono*:

$$Z' \subseteq V \implies Z \subseteq Z' \implies \text{Approx}_\beta l' Z \subseteq \text{Approx}_\beta l' Z'$$

**proof** (*goal-cases*)

**case** 1

**with** *regions.beta-interp.apx-in* **have**

$$\begin{aligned} \text{regions.Approx}_\beta Z' \in \{S. \exists U M. S = \bigcup U \wedge U \subseteq \text{regions.R}_\beta \wedge Z' \subseteq \\ S \wedge \text{regions.beta-interp.vabstr } S M \\ \wedge \text{regions.beta-interp.normalized } M\} \end{aligned}$$

**by** *auto*

**with** 1 **obtain**  $U M$  **where**

$$\begin{aligned} \text{regions.Approx}_\beta Z' = \bigcup U \quad U \subseteq \text{regions.R}_\beta \quad Z \subseteq \text{regions.Approx}_\beta Z' \\ \text{regions.beta-interp.vabstr } (\text{regions.Approx}_\beta Z') M \\ \text{regions.beta-interp.normalized } M \end{aligned}$$

**by** *auto*

**with** *regions.beta-interp.apx-min* **show** *?thesis* **by** *auto*

qed

end

**lemma** *step-z'-V'*:

**assumes**  $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle$  *valid-abstraction*  $A \ X \ k \ \forall c \in \text{clk-set } A. v \ c \leq n \ Z \in V'$

**shows**  $Z' \in V'$

**using** *assms* **unfolding** *step-z'-def* **by** (*auto elim: step-z-V'*)

**lemma** *steps-z-V'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A. v \ c \leq n \implies Z \in V' \implies Z' \in V'$

**by** (*induction rule: rtranclp-induct2; blast intro: step-z'-V'*)

## 6.4.2 Multi step

**definition**

*step-z-beta'* ::  $(l'a, l'c, t, l's) \text{ ta} \Rightarrow l's \Rightarrow (l'c, t) \text{ zone} \Rightarrow l's \Rightarrow (l'c, t) \text{ zone} \Rightarrow \text{bool}$

$(\langle - \vdash \langle -, - \rangle \rightsquigarrow_\beta \langle -, - \rangle) [61, 61, 61] \ 61)$

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z'' \rangle = (\exists Z' a. A \vdash \langle l, Z \rangle \rightsquigarrow_\tau \langle l, Z' \rangle \wedge A \vdash \langle l, Z' \rangle \rightsquigarrow_{\beta(1a)} \langle l', Z'' \rangle)$

**abbreviation**

*steps-z-beta* ::  $(l'a, l'c, t, l's) \text{ ta} \Rightarrow l's \Rightarrow (l'c, t) \text{ zone} \Rightarrow l's \Rightarrow (l'c, t) \text{ zone} \Rightarrow$

*bool*

$(\langle \cdot \vdash \langle \cdot, \cdot \rangle \rightsquigarrow_{\beta^*} \langle \cdot, \cdot \rangle) [61,61,61] 61)$

**where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z'' \rangle \equiv (\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z'' \rangle)^{**}$   
 $(l, Z) (l', Z'')$

**lemma**  $V'-V$ :  $Z \in V' \implies Z \subseteq V$  **unfolding**  $V'$ -def **by** *auto*

**context**

**fixes**  $A :: ('a, 'c, t, 's) ta$

**assumes** *valid-ta*: *valid-abstraction*  $A X k \forall c \in \text{clk-set } A. v c \leq n$

**begin**

**interpretation** *alpha*: *AlphaClosure-global* -  $k l' \mathcal{R} l'$  **by** *standard* (*rule finite*)

**lemma** [*simp*]:  $\text{alpha.cla } l' = \text{cla } l'$  **unfolding** *alpha.cla-def* *cla-def* ..

**lemma** *step-z-alpha'-V*:

$Z' \subseteq V$  **if**  $Z \subseteq V$   $A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z' \rangle$

**using** *that alpha.closure-V[simplified]* **unfolding** *step-z-alpha'-def* **by** *blast*

**lemma** *step-z-beta'-V'*:

$Z' \in V'$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle$   $Z \in V'$

**proof** –

**interpret** *regions*: *Regions-global* - - -  $k l'$

**by** *standard* (*rule finite clock-numbering not-in-X non-empty*)+

**from** *that valid-ta* **show** *?thesis*

**unfolding** *step-z-beta'-def* **by** (*blast intro: step-z-V' regions.apx-V'[OF V'-V]*)

**qed**

**lemma** *steps-z-beta-V'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies Z \in V' \implies Z' \in V'$

**by** (*induction rule: rtranclp-induct2*; *blast intro: step-z-beta'-V'*)

**Soundness lemma** *alpha'-beta'-step*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \implies Z \in V' \implies W \subseteq V \implies Z \subseteq W \implies \exists W'$   
 $A \vdash \langle l, W \rangle \rightsquigarrow_{\alpha} \langle l', W' \rangle \wedge Z' \subseteq W'$

**unfolding** *step-z-beta'-def* *step-z-alpha'-def*

**apply** (*elim exE conjE*)

**apply** (*frule step-z-mono, assumption*)

**apply** (*elim exE conjE*)

**apply** (*frule alpha-beta-step'*[*OF - valid-ta*])  
**prefer** 3  
**using** *valid-ta* **by** (*blast intro: step-z-V' dest: step-z-V*)**+**

**lemma** *alpha-beta-sim:*

*Simulation-Invariant*

$(\lambda(l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z'' \rangle)$

$(\lambda(l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle)$

$(\lambda(l, Z) (l', Z'). l = l' \wedge Z \subseteq Z') (\lambda(-, Z). Z \in V') (\lambda(-, Z). Z \subseteq V)$

**by** *standard* (*auto elim: alpha'-beta'-step step-z-beta'-V' dest: step-z-alpha'-V*)

**interpretation**

*Simulation-Invariant*

$\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z'' \rangle$

$\lambda (l, Z) (l', Z''). A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle$

$\lambda (l, Z) (l', Z'). l = l' \wedge Z \subseteq Z'$

$\lambda (-, Z). Z \in V' \lambda (-, Z). Z \subseteq V$

**by** (*fact alpha-beta-sim*)

**lemma** *alpha-beta-steps:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies Z \in V' \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z'' \rangle$   
 $\wedge Z' \subseteq Z''$

**using** *simulation-reaches*[*of (l, Z) (l', Z') (l, Z)*] **by** (*auto dest: V'-V*)

**end**

**Completeness lemma** *step-z-beta-mono:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle$   
 $\rightsquigarrow_{\beta(a)} \langle l', W' \rangle \wedge Z' \subseteq W'$

**proof** (*goal-cases*)

**case** 1

**then obtain**  $Z''$  **where**  $*$ :  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z'' \rangle$   $Z' = \text{Approx}_{\beta} l' Z''$  **by**  
*auto*

**from** *step-z-mono*[*OF this(1) 1(2)*] **obtain**  $W'$  **where**

$A \vdash \langle l, W \rangle \rightsquigarrow_a \langle l', W' \rangle$   $Z'' \subseteq W'$

**by** *auto*

**moreover with**  $*(2)$  *apx-mono*[*OF step-z-V*]  $\langle W \subseteq V \rangle$  **have**

$Z' \subseteq \text{Approx}_{\beta} l' W'$

**by** *metis*

**ultimately show** *?case* **by** *blast*

**qed**

**lemma** *step-z-beta'-V*:

$Z' \subseteq V$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle$   $Z \subseteq V$

**proof** –

**interpret** *regions*: *Regions-global - - k l'*

**by** *standard (rule finite clock-numbering not-in-X non-empty)+*

**from** *that* **show** *?thesis unfolding step-z-beta'-def*

**by** *(auto intro: regions.apx-V dest: step-z-V del: subsetI)*

**qed**

**lemma** *steps-z-beta-V*:

$Z' \subseteq V$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle$   $Z \subseteq V$

**using** *that* **by** *(induction rule: rtranclp-induct2; blast intro: step-z-beta'-V del: subsetI)*

**lemma** *step-z-beta'-mono*:

$\exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\beta} \langle l', W' \rangle \wedge Z' \subseteq W' \wedge W' \subseteq V$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle$   $Z \subseteq V$

**using** *that* **unfolding** *step-z-beta'-def*

**apply** *(elim exE conjE)*

**apply** *(frule step-z-mono, assumption)*

**apply** *(elim exE conjE)*

**apply** *(drule step-z-beta-mono, assumption)*

**apply** *(auto dest: step-z-V)*

**done**

**lemma** *steps-z-beta-mono*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\beta^*} \langle l', W' \rangle \wedge Z' \subseteq W'$

**apply** *(induction rule: rtranclp-induct2)*

**apply** *blast*

**apply** *(clarsimp; drule step-z-beta'-mono;*

*blast intro: rtranclp.intros(2) steps-z-beta-V del: subsetI)*

**done**

**end**

**end**

**theory** *Simulation-Graphs*

```

imports
  library/CTL
  library/More-List1
begin

lemmas [simp] = holds.simps

```

## 7 Simulation Graphs

### 7.1 Simulation Graphs

```

locale Simulation-Graph-Defs = Graph-Defs C for C :: 'a ⇒ 'a ⇒ bool +
  fixes A :: 'a set ⇒ 'a set ⇒ bool
begin

```

```

sublocale Steps: Graph-Defs A .

```

```

abbreviation Steps ≡ Steps.steps
abbreviation Run ≡ Steps.run

```

```

lemmas Steps-appendD1 = Steps.steps-appendD1

```

```

lemmas Steps-appendD2 = Steps.steps-appendD2

```

```

lemmas steps-alt-induct = Steps.steps-alt-induct

```

```

lemmas Steps-appendI = Steps.steps-appendI

```

```

lemmas Steps-cases = Steps.steps.cases

```

```

end

```

```

locale Simulation-Graph-Poststable = Simulation-Graph-Defs +
  assumes poststable: A S T ⇒ ∀ s' ∈ T. ∃ s ∈ S. C s s'

```

```

locale Simulation-Graph-Prestable = Simulation-Graph-Defs +
  assumes prestable: A S T ⇒ ∀ s ∈ S. ∃ s' ∈ T. C s s'

```

```

locale Double-Simulation-Defs =
  fixes C :: 'a ⇒ 'a ⇒ bool — Concrete step relation
  and A1 :: 'a set ⇒ 'a set ⇒ bool — Step relation for the first abstraction
  layer
  and P1 :: 'a set ⇒ bool — Valid states of the first abstraction layer
  and A2 :: 'a set ⇒ 'a set ⇒ bool — Step relation for the second

```

abstraction layer  
**and**  $P2 :: 'a \text{ set} \Rightarrow \text{bool}$  — Valid states of the second abstraction layer  
**begin**

**sublocale** *Simulation-Graph-Defs*  $C \ A2$  .

**sublocale** *pre-defs: Simulation-Graph-Defs*  $C \ A1$  .

**definition** *closure*  $a = \{x. P1 \ x \wedge a \cap x \neq \{\}\}$

**definition**  $A2' \ a \ b \equiv \exists \ x \ y. a = \text{closure } x \wedge b = \text{closure } y \wedge A2 \ x \ y$

**sublocale** *post-defs: Simulation-Graph-Defs*  $A1 \ A2'$  .

**lemma** *closure-mono*:  
*closure*  $a \subseteq \text{closure } b$  **if**  $a \subseteq b$   
**using** *that* **unfolding** *closure-def* **by** *auto*

**lemma** *closure-intD*:  
 $x \in \text{closure } a \wedge x \in \text{closure } b$  **if**  $x \in \text{closure } (a \cap b)$   
**using** *that* *closure-mono* **by** *blast*

**end**

**locale** *Double-Simulation* = *Double-Simulation-Defs* +  
**assumes** *prestable*:  $A1 \ S \ T \Longrightarrow \forall \ s \in S. \exists \ s' \in T. C \ s \ s'$   
**and** *closure-poststable*:  $s' \in \text{closure } y \Longrightarrow A2 \ x \ y \Longrightarrow \exists \ s \in \text{closure } x. A1 \ s \ s'$   
**and** *P1-distinct*:  $P1 \ x \Longrightarrow P1 \ y \Longrightarrow x \neq y \Longrightarrow x \cap y = \{\}$   
**and** *P1-finite*: *finite*  $\{x. P1 \ x\}$   
**and** *P2-cover*:  $P2 \ a \Longrightarrow \exists \ x. P1 \ x \wedge x \cap a \neq \{\}$   
**begin**

**sublocale** *post: Simulation-Graph-Poststable*  $A1 \ A2'$   
**unfolding** *A2'-def* **by** *standard* (*auto dest: closure-poststable*)

**sublocale** *pre: Simulation-Graph-Prestable*  $C \ A1$   
**by** *standard* (*rule prestable*)

**end**

**locale** *Finite-Graph* = *Graph-Defs* +  
**fixes**  $x_0$   
**assumes** *finite-reachable*: *finite*  $\{x. E^{**} \ x_0 \ x\}$

```

locale Simulation-Graph-Complete-Defs =
  Simulation-Graph-Defs C A for C :: 'a ⇒ 'a ⇒ bool and A :: 'a set ⇒ 'a
  set ⇒ bool +
  fixes P :: 'a set ⇒ bool — well-formed abstractions

locale Simulation-Graph-Complete = Simulation-Graph-Complete-Defs +
  simulation: Simulation-Invariant C A (∈) λ -. True P
begin

lemmas complete = simulation.A-B-step
lemmas P-invariant = simulation.B-invariant

end

locale Simulation-Graph-Finite-Complete = Simulation-Graph-Complete +
  fixes a0
  assumes finite-abstract-reachable: finite {a. A** a0 a}
begin

sublocale Steps-finite: Finite-Graph A a0
  by standard (rule finite-abstract-reachable)

end

locale Double-Simulation-Complete = Double-Simulation +
  fixes a0
  assumes complete: C x y ⇒ x ∈ S ⇒ P2 S ⇒ ∃ T. A2 S T ∧ y ∈ T
  assumes P2-invariant: P2 a ⇒ A2 a a' ⇒ P2 a'
  and P2-a0: P2 a0
begin

sublocale Simulation-Graph-Complete C A2 P2
  by standard (blast intro: complete P2-invariant) +

sublocale P2-invariant: Graph-Invariant-Start A2 a0 P2
  by (standard; blast intro: P2-invariant P2-a0)

end

locale Double-Simulation-Finite-Complete = Double-Simulation-Complete
+
  assumes finite-abstract-reachable: finite {a. A2** a0 a}
begin

```

**sublocale** *Simulation-Graph-Finite-Complete*  $C A2 P2 a_0$   
**by standard** (blast intro: complete finite-abstract-reachable P2-invariant)+  
**end**

**locale** *Simulation-Graph-Complete-Prestable* = *Simulation-Graph-Complete*  
+ *Simulation-Graph-Prestable*  
**begin**

**sublocale** *Graph-Invariant*  $A P$  **by standard** (rule P-invariant)  
**end**

**locale** *Double-Simulation-Complete-Bisim* = *Double-Simulation-Complete*  
+  
**assumes** *A1-complete*:  $C x y \implies P1 S \implies x \in S \implies \exists T. A1 S T \wedge y \in T$   
**and** *P1-invariant*:  $P1 S \implies A1 S T \implies P1 T$   
**begin**

**sublocale** *bisim*: *Simulation-Graph-Complete-Prestable*  $C A1 P1$   
**by standard** (blast intro: A1-complete P1-invariant)+  
**end**

**locale** *Double-Simulation-Finite-Complete-Bisim* =  
*Double-Simulation-Finite-Complete* + *Double-Simulation-Complete-Bisim*

**locale** *Double-Simulation-Complete-Bisim-Cover* = *Double-Simulation-Complete-Bisim*  
+  
**assumes** *P2-P1-cover*:  $P2 a \implies x \in a \implies \exists a'. a \cap a' \neq \{\} \wedge P1 a' \wedge x \in a'$

**locale** *Double-Simulation-Finite-Complete-Bisim-Cover* =  
*Double-Simulation-Finite-Complete-Bisim* + *Double-Simulation-Complete-Bisim-Cover*

**locale** *Double-Simulation-Complete-Abstraction-Prop* =  
*Double-Simulation-Complete* +  
**fixes**  $\varphi :: 'a \Rightarrow bool$  — The property we want to check  
**assumes**  $\varphi$ -A1-compatible:  $A1 a b \implies b \subseteq \{x. \varphi x\} \vee b \cap \{x. \varphi x\} = \{\}$   
**and**  $\varphi$ -P2-compatible:  $P2 a \implies a \cap \{x. \varphi x\} \neq \{\} \implies P2 (a \cap \{x. \varphi x\})$   
**and**  $\varphi$ -A2-compatible:  $A2^{**} a_0 a \implies a \cap \{x. \varphi x\} \neq \{\} \implies A2^{**} a_0$

$(a \cap \{x. \varphi x\})$   
**and** *P2-non-empty*:  $P2 a \implies a \neq \{\}$

**locale** *Double-Simulation-Complete-Abstraction-Prop-Bisim* =  
*Double-Simulation-Complete-Abstraction-Prop* + *Double-Simulation-Complete-Bisim*

**locale** *Double-Simulation-Finite-Complete-Abstraction-Prop* =  
*Double-Simulation-Complete-Abstraction-Prop* + *Double-Simulation-Finite-Complete*

**locale** *Double-Simulation-Finite-Complete-Abstraction-Prop-Bisim* =  
*Double-Simulation-Finite-Complete-Abstraction-Prop* + *Double-Simulation-Finite-Complete-Bisim*

## 7.2 Poststability

**context** *Simulation-Graph-Poststable*  
**begin**

**lemma** *Steps-poststable*:

$\exists xs. \text{steps } xs \wedge \text{list-all2 } (\in) xs \text{ as} \wedge \text{last } xs = x$  **if** *Steps as x*  $\in$  *last as*  
**using that**

**proof** *induction*

**case** (*Single a*)

**then show** *?case* **by** *auto*

**next**

**case** (*Cons a b as*)

**then obtain** *xs* **where** *A a b steps xs list-all2*  $(\in) xs (b \# as) x = \text{last } xs$

**by** *clarsimp*

**then have** *hd xs*  $\in$  *b* **by** (*cases xs*) *auto*

**with** *poststable[OF*  $\langle A a b \rangle$  **obtain** *y* **where** *y*  $\in$  *a C y* (*hd xs*) **by** *auto*

**with**  $\langle \text{list-all2 } - - \rightarrow \rangle \langle \text{steps } - \rightarrow \rangle \langle x = - \rightarrow \rangle$  **show** *?case* **by** (*cases xs*) *auto*

**qed**

**lemma** *reaches-poststable*:

$\exists x \in a. \text{reaches } x y$  **if** *Steps.reaches a b y*  $\in$  *b*

**using that** **unfolding** *reaches-steps-iff* *Steps.reaches-steps-iff*

**apply** *clarify*

**apply** (*drule* *Steps-poststable*, *assumption*)

**apply** *clarify*

**subgoal for** *as xs*

**apply** (*cases xs* =  $\square$ )

**apply** *force*

**apply** (*rule* *bexI*[**where** *x* = *hd xs*])

**using** *list.rel-sel* **by** (*auto* *dest*: *Graph-Defs.steps-non-empty'*)

done

**lemma** *Steps-steps-cycle:*

$\exists x xs. \text{steps } (x \# xs @ [x]) \wedge (\forall x \in \text{set } xs. \exists a \in \text{set } as \cup \{a\}. x \in a)$   
 $\wedge x \in a$

**if** *assms: Steps (a # as @ [a]) finite a a ≠ {}*

**proof** –

**define** *E* **where**

$E x y = (\exists xs. \text{steps } (x \# xs @ [y]) \wedge (\forall x \in \text{set } xs \cup \{x, y\}. \exists a \in \text{set } as \cup \{a\}. x \in a))$

**for** *x y*

**from** *assms(2-)* **have**  $\exists x. E x y \wedge x \in a$  **if**  $y \in a$  **for** *y*

**using** *that unfolding E-def*

**apply** *simp*

**apply** (*drule Steps-poststable[OF assms(1), simplified]*)

**apply** *clarify*

**subgoal for** *xs*

**apply** (*inst-existentials hd xs tl (butlast xs)*)

**subgoal by** (*cases xs*) *auto*

**subgoal by** (*auto elim: steps.cases dest!: list-all2-set1*)

**subgoal by** (*drule list-all2-set1*) (*cases xs, auto dest: in-set-butlastD*)

**by** (*cases xs*) *auto*

done

**with**  $\langle \text{finite } a \rangle \langle a \neq \{\} \rangle$  **obtain** *x y* **where** *cycle: E x y E\*\* y x x ∈ a*

**by** (*force dest!: Graph-Defs.directed-graph-indegree-ge-1-cycle*)

**have** *trans[intro]: E x z if E x y E y z for x y z*

**using** *that unfolding E-def*

**apply** *safe*

**subgoal for** *xs ys*

**apply** (*inst-existentials xs @ y # ys*)

**apply** (*drule steps-append, assumption; simp; fail*)

**by** (*cases ys, auto dest: list.set-sel(2)[rotated] elim: steps.cases*)

done

**have** *E x z if E\*\* y z E x y x ∈ a for x y z*

**using** *that proof induction*

**case** *base*

**then show** *?case unfolding E-def by force*

**next**

**case** (*step y z*)

**then show** *?case by auto*

**qed**

**with** *cycle* **have** *E x x by blast*

**with**  $\langle x \in a \rangle$  **show** *?thesis unfolding E-def by auto*

**qed**

end

### 7.3 Prestability

**context** *Simulation-Graph-Prestable*  
**begin**

**lemma** *Steps-prestable:*

$\exists xs. \text{steps } (x \# xs) \wedge \text{list-all2 } (\in) (x \# xs) \text{ as}$  **if** *Steps*.as  $x \in \text{hd as}$   
**using that**

**proof** (*induction arbitrary: x*)

**case** (*Single a*)

**then show** *?case* **by auto**

**next**

**case** (*Cons a b as*)

**from** *prestable*[*OF*  $\langle A \ a \ b \rangle$ ]  $\langle x \in \rightarrow$  **obtain** *y* **where**  $y \in b \ C \ x \ y$  **by auto**

**with** *Cons.IH*[*of y*] **obtain** *xs* **where**  $y \in b \ C \ x \ y \ \text{steps } (y \# xs) \ \text{list-all2 } (\in) \ xs \ \text{as}$

**by** *clarsimp*

**with**  $\langle x \in \rightarrow$  **show** *?case* **by auto**

**qed**

**lemma** *reaches-prestable:*

$\exists y. \text{reaches } x \ y \wedge y \in b$  **if** *Steps*.reaches *a b x*  $x \in a$

**using that** **unfolding** *reaches-steps-iff* *Steps*.reaches-steps-iff

**by** (*force simp: hd-map last-map dest: list-all2-last dest!: Steps-prestable*)

Abstract cycles lead to concrete infinite runs.

**lemma** *Steps-run-cycle-buechi:*

$\exists xs. \text{run } (x \#\# xs) \wedge \text{stream-all2 } (\in) \ xs \ (\text{cycle } (as \ @ \ [a]))$

**if** *assms: Steps* (*a # as @ [a]*)  $x \in a$

**proof** –

**note**  $C = \text{Steps-prestable}[\text{OF } \text{assms}(1), \text{simplified}]$

**define** *P* **where**  $P \equiv \lambda x \ xs. \ \text{steps } (\text{last } x \# xs) \wedge \text{list-all2 } (\in) \ xs \ (as \ @ \ [a])$

**define** *f* **where**  $f \equiv \lambda x. \ \text{SOME } xs. \ P \ x \ xs$

**from** *Steps-prestable*[*OF assms(1)*]  $\langle x \in a \rangle$  **obtain** *ys* **where** *ys*:

$\text{steps } (x \# ys) \ \text{list-all2 } (\in) (x \# ys) \ (a \# as \ @ \ [a])$

**by auto**

**define** *xs* **where**  $xs = \text{flat } (\text{siterate } f \ ys)$

**from** *ys* **have**  $P \ [x] \ ys$  **unfolding** *P-def* **by auto**

**from**  $\langle P \ - \ \rightarrow \rangle$  **have**  $*$ :  $\exists xs. \ P \ xs \ ys$  **by blast**

**have**  $P-1[\text{intro}]: ys \neq []$  **if**  $P \ xs \ ys$  **for** *xs ys* **using that** **unfolding** *P-def*

**by** (*cases ys*) *auto*  
**have**  $P-2$ [*intro*]: *last ys*  $\in a$  **if**  $P$  *xs ys* **for** *xs ys*  
**using** *that P-1*[*OF that*] **unfolding**  $P$ -*def* **by** (*auto dest: list-all2-last*)  
**from** \* **have** *stream-all2* ( $\in$ ) *xs* (*cycle* (*as* @ [*a*]))  
**unfolding** *xs-def* **proof** (*coinduction arbitrary: ys rule: stream-rel-coinduct-shift*)  
**case** *prems: stream-rel*  
**then** **have** *ys*  $\neq []$  *last ys*  $\in a$  **by** (*blast dest: P-1 P-2*)+  
**from**  $\langle ys \neq [] \rangle$   $C$ [*OF*  $\langle last\ ys \in a \rangle$ ] **have**  $\exists xs. P\ ys\ xs$  **unfolding**  $P$ -*def*  
**by** *auto*  
**from** *someI-ex*[*OF this*] **have**  $P\ ys\ (f\ ys)$  **unfolding**  $f$ -*def* .  
**with**  $\langle ys \neq [] \rangle$  *prems* **show** ?*case*  
**apply** (*inst-existentials ys flat* (*siterate f* ( $f\ ys$ )) *as* @ [*a*] *cycle* (*as* @ [*a*]))  
**apply** (*subst siterate.ctr; simp; fail*)  
**apply** (*subst cycle-decomp; simp; fail*)  
**by** (*auto simp: P-def*)  
**qed**  
**from** \* **have** *run xs*  
**unfolding** *xs-def* **proof** (*coinduction arbitrary: ys rule: run-flat-coinduct*)  
**case** *prems: (run-shift xs ws xss ys)*  
**then** **have** *ys*  $\neq []$  *last ys*  $\in a$  **by** (*blast dest: P-1 P-2*)+  
**from**  $\langle ys \neq [] \rangle$   $C$ [*OF*  $\langle last\ ys \in a \rangle$ ] **have**  $\exists xs. P\ ys\ xs$  **unfolding**  $P$ -*def*  
**by** *auto*  
**from** *someI-ex*[*OF this*] **have**  $P\ ys\ (f\ ys)$  **unfolding**  $f$ -*def* .  
**with**  $\langle ys \neq [] \rangle$  *prems* **show** ?*case* **by** (*auto elim: steps.cases simp: P-def*)  
**qed**  
**with**  $P-1$ [*OF*  $\langle P - \rightarrow \rangle$ ]  $\langle steps\ (x\ \#\ ys) \rangle$  **have** *run* ( $x\ \#\#\ xs$ )  
**unfolding** *xs-def*  
**by** (*subst siterate.ctr, subst (asm) siterate.ctr*) (*cases ys; auto elim: steps.cases*)  
**with**  $\langle stream-all2 - - \rightarrow \rangle$  **show** ?*thesis* **by** *blast*  
**qed**

**lemma** *Steps-run-cycle-buechi'*:

$\exists xs. run\ (x\ \#\#\ xs) \wedge (\forall x \in sset\ xs. \exists a \in set\ as \cup \{a\}. x \in a) \wedge infs$   
 $(\lambda x. x \in b)\ (x\ \#\#\ xs)$   
**if** *assms: Steps* ( $a\ \#\ as$  @ [*a*])  $x \in a\ b \in set\ (a\ \#\ as$  @ [*a*])  
**using** *Steps-run-cycle-buechi*[*OF that(1,2)*] *that(2,3)*  
**apply** *safe*  
**apply** (*rule exI conjI*)+  
**apply** *assumption*  
**apply** (*subst alw-ev-stl[symmetric]*)  
**by** (*force dest: alw-ev-HLD-cycle[of - - b]*) *stream-all2-sset1*

**lemma** *Steps-run-cycle-buechi'*:

$\exists xs. \text{run } (x \#\# xs) \wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\}. x \in a) \wedge \text{infs}$   
 $(\lambda x. x \in a) (x \#\# xs)$   
**if** *assms*:  $\text{Steps } (a \# as @ [a]) x \in a$   
**using** *Steps-run-cycle-buechi''*[*OF that*]  $\langle x \in a \rangle$  **by** *auto*

**lemma** *Steps-run-cycle'*:

$\exists xs. \text{run } (x \#\# xs) \wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\}. x \in a)$   
**if** *assms*:  $\text{Steps } (a \# as @ [a]) x \in a$   
**using** *Steps-run-cycle-buechi''*[*OF assms*] **by** *auto*

**lemma** *Steps-run-cycle*:

$\exists xs. \text{run } xs \wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\}. x \in a) \wedge \text{shd } xs \in a$   
**if** *assms*:  $\text{Steps } (a \# as @ [a]) a \neq \{\}$   
**using** *Steps-run-cycle'*[*OF assms(1)*] *assms(2)* **by** *force*

**Unused lemma** *Steps-cycle-every-prestable'*:

$\exists b y. C x y \wedge y \in b \wedge b \in \text{set } as \cup \{a\}$   
**if** *assms*:  $\text{Steps } (as @ [a]) x \in b b \in \text{set } as$   
**using** *assms*

**proof** (*induction as @ [a] arbitrary: as*)

**case** *Single*

**then show** *?case* **by** *simp*

**next**

**case** (*Cons a c xs*)

**show** *?case*

**proof** (*cases a = b*)

**case** *True*

**with** *prestable*[*OF*  $\langle A a c \rangle$ ]  $\langle x \in b \rangle$  **obtain** *y* **where**  $y \in c C x y$   
**by** *auto*

**with**  $\langle a \# c \# - = \rightarrow \rangle$  **show** *?thesis*

**apply** (*inst-existentials c y*)

**proof** (*assumption+*, *cases as*, *goal-cases*)

**case** (*2 a list*)

**then show** *?case* **by** (*cases list*) *auto*

**qed** *simp*

**next**

**case** *False*

**with** *Cons.hyps(3)*[*of tl as*] *Cons.prem*s *Cons.hyps(1,2,4-)* **show** *?thesis*

**by** (*cases as*) *auto*

**qed**

**qed**

**lemma** *Steps-cycle-first-prestable*:

$\exists b y. C x y \wedge x \in b \wedge b \in \text{set } as \cup \{a\}$  **if** *assms*: *Steps* ( $a \# as @ [a]$ )  $x \in a$

**proof** (*cases as*)

**case** *Nil*

**with** *assms* **show** *?thesis* **by** (*auto elim!*: *Steps-cases dest: prestable*)

**next**

**case** (*Cons b as*)

**with** *assms* **show** *?thesis* **by** (*auto 4 4 elim: Steps-cases dest: prestable*)

**qed**

**lemma** *Steps-cycle-every-prestable*:

$\exists b y. C x y \wedge y \in b \wedge b \in \text{set } as \cup \{a\}$

**if** *assms*: *Steps* ( $a \# as @ [a]$ )  $x \in b \wedge b \in \text{set } as \cup \{a\}$

**using** *assms* *Steps-cycle-every-prestable'*[*of a # as a*] *Steps-cycle-first-prestable*  
**by** *auto*

**end**

## 7.4 Double Simulation

**context** *Double-Simulation*

**begin**

**lemma** *closure-involutive*:

$\text{closure } (\bigcup (\text{closure } x)) = \text{closure } x$

**unfolding** *closure-def* **by** (*auto dest: P1-distinct*)

**lemma** *closure-finite*:

*finite* (*closure x*)

**using** *P1-finite* **unfolding** *closure-def* **by** *auto*

**lemma** *closure-non-empty*:

$\text{closure } x \neq \{\}$  **if** *P2 x*

**using** *that* **unfolding** *closure-def* **by** (*auto dest!: P2-cover*)

**lemma** *P1-closure-id*:

$\text{closure } R = \{R\}$  **if** *P1 R R*  $R \neq \{\}$

**unfolding** *closure-def* **using** *that P1-distinct* **by** *blast*

**lemma** *A2'-A2-closure*:

*A2'* (*closure x*) (*closure y*) **if** *A2 x y*

**using** *that* **unfolding** *A2'-def* **by** *auto*

**lemma** *Steps-Union*:  
*post-defs.Steps (map closure xs) if Steps xs*  
**using that proof** (*induction xs rule: rev-induct*)  
  **case** *Nil*  
  **then show** *?case by auto*  
**next**  
  **case** (*snoc y xs*)  
  **show** *?case*  
  **proof** (*cases xs rule: rev-cases*)  
    **case** *Nil*  
    **then show** *?thesis by auto*  
  **next**  
    **case** (*snoc ys z*)  
    **with** *Steps-appendD1[OF <Steps (xs @ [y])>]* **have** *Steps xs by simp*  
    **then have** *\*: post-defs.Steps (map closure xs) by (rule snoc.IH)*  
    **with** *<xs = -> snoc.premis* **have** *A2 z y*  
    **by** (*metis Steps.steps-appendD3 append-Cons append-assoc append-self-conv2*)  
    **with** *<A2 z y>* **have** *A2' (closure z) (closure y) by (auto dest!: A2'-A2-closure)*  
    **with** *\* post-defs.Steps-appendI* **show** *?thesis*  
    **by** (*simp add: <xs = ->*)  
  **qed**  
**qed**

**lemma** *closure-reaches*:  
*post-defs.Steps.reaches (closure x) (closure y) if Steps.reaches x y*  
**using that**  
**unfolding** *Steps.reaches-steps-iff post-defs.Steps.reaches-steps-iff*  
**apply** *clarify*  
**apply** (*drule Steps-Union*)  
**subgoal for** *xs*  
  **by** (*cases xs = []; force simp: hd-map last-map*)  
**done**

**lemma** *post-Steps-non-empty*:  
*x ≠ {} if post-defs.Steps (a # as) x ∈ b b ∈ set as*  
**using that**  
**proof** (*induction a # as arbitrary: a as*)  
  **case** *Single*  
  **then show** *?case by auto*  
**next**  
  **case** (*Cons a c as*)  
  **then show** *?case by (auto simp: A2'-def closure-def)*  
**qed**

**lemma** *Steps-run-cycle'*:  
 $\exists xs. \text{run } xs \wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\}. x \in \bigcup a) \wedge \text{shd } xs \in \bigcup a$   
**if** *assms*: *post-defs.Steps* (*a* # *as* @ [*a*]) *finite a a* ≠ {}  
**proof** –  
**from** *post.Steps-steps-cycle*[*OF assms*] **obtain** *a1 as1* **where** *guessed*:  
*pre-defs.Steps* (*a1* # *as1* @ [*a1*])  
 $\forall x \in \text{set } as1. \exists a \in \text{set } as \cup \{a\}. x \in a$   
*a1* ∈ *a*  
**by** *atomize-elim*  
**from** *assms*(1) <*a1* ∈ *a*> **have** *a1* ≠ {} **by** (*auto dest!*: *post-Steps-non-empty*)  
**with** *guessed pre.Steps-run-cycle*[*of a1 as1*] **obtain** *xs* **where**  
 $\text{run } xs \forall x \in \text{sset } xs. \exists a \in \text{set } as1 \cup \{a1\}. x \in a \text{ shd } xs \in a1$   
**by** *atomize-elim auto*  
**with** *guessed*(2,3) **show** *?thesis*  
**by** (*inst-existentials xs*) (*metis Un-iff UnionI empty-iff insert-iff*)+  
**qed**

**lemma** *Steps-run-cycle*:  
 $\exists xs. \text{run } xs \wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\}. x \in \bigcup (\text{closure } a)) \wedge \text{shd } xs \in \bigcup (\text{closure } a)$   
**if** *assms*: *Steps* (*a* # *as* @ [*a*]) *P2 a*  
**proof** –  
**from** *Steps-Union*[*OF assms*(1)] **have** *post-defs.Steps* (*closure a* # *map closure as* @ [*closure a*])  
**by** *simp*  
**from** *Steps-run-cycle'*[*OF this closure-finite closure-non-empty*[*OF* <*P2 a*>]]  
**show** *?thesis* **by** (*force dest: list-all2-set2*)  
**qed**

**lemma** *Steps-run-cycle2*:  
 $\exists x xs. \text{run } (x \#\# xs) \wedge x \in \bigcup (\text{closure } a_0)$   
 $\wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\} \cup \text{set } bs. x \in \bigcup a)$   
 $\wedge \text{infs } (\lambda x. x \in \bigcup a) (x \#\# xs)$   
**if** *assms*: *post-defs.Steps* (*closure a<sub>0</sub>* # *as* @ *a* # *bs* @ [*a*]) *a* ≠ {}  
**proof** –  
**note** *as1* = *assms*  
**from**  
*post-defs.Steps.steps-decomp*[*of closure a<sub>0</sub>* # *as a* # *bs* @ [*a*]]  
*as1*(1)[*unfolded this*]  
**have** \*:  
*post-defs.Steps* (*closure a<sub>0</sub>* # *as*)  
*post-defs.Steps* (*a* # *bs* @ [*a*])

```

    A2' (last (closure a0 # as)) (a)
  by (simp split: if-split-asm add: last-map)+
  then have finite a
    unfolding A2'-def by (metis closure-finite)
  from post.Steps-steps-cycle[OF *(2) ⟨finite a⟩ ⟨a ≠ {}⟩] obtain a1 as1
where as1:
  pre-defs.Steps (a1 # as1 @ [a1])
  ∀x∈set as1. ∃a∈set bs ∪ {a}. x ∈ a
  a1 ∈ a
  by atomize-elim
with post.poststable[OF *(3)] obtain a2 where a2 ∈ last (closure a0 #
as) A1 a2 a1
  by auto
with post.Steps-poststable[OF *(1), of a2] obtain as2 where as2:
  pre-defs.Steps as2 list-all2 (∈) as2 (closure a0 # as) last as2 = a2
  by (auto split: if-split-asm simp: last-map)
from as2(2) have hd as2 ∈ closure a0 by (cases as2) auto
then have hd as2 ≠ {} unfolding closure-def by auto
then obtain x0 where x0 ∈ hd as2 by auto
from pre.Steps-prestable[OF as2(1) ⟨x0 ∈ -⟩] obtain xs where xs:
  steps (x0 # xs) list-all2 (∈) (x0 # xs) as2
  by auto
with ⟨last as2 = a2⟩ have last (x0 # xs) ∈ a2
  unfolding list-all2-Cons1 by (auto intro: list-all2-last)
with pre.prestable[OF ⟨A1 a2 a1⟩] obtain y where C (last (x0 # xs)) y
y ∈ a1 by auto
from pre.Steps-run-cycle-buechi'[OF as1(1) ⟨y ∈ a1⟩] obtain ys where
ys:
  run (y ## ys) ∀x∈sset ys. ∃a∈set as1 ∪ {a1}. x ∈ a infs (λx. x ∈ a1)
(y ## ys)
  by auto
from ys(3) ⟨a1 ∈ a⟩ have infs (λx. x ∈ ∪ a) (y ## ys)
  by (auto simp: HLD-iff elim!: alw-ev-mono)
from extend-run[OF xs(1) ⟨C - -⟩ ⟨run (y ## ys)⟩] have run ((x0 # xs)
@- y ## ys) by simp
then show ?thesis
  apply (inst-existentials x0 xs @- y ## ys)
  apply (simp; fail)
  using ⟨x0 ∈ -⟩ ⟨hd as2 ∈ -⟩ apply (auto; fail)
  using xs(2) as2(2) *(2) ⟨y ∈ a1⟩ ⟨a1 ∈ -⟩ ys(2) as1(2)
  unfolding list-all2-op-map-iff list-all2-Cons1 list-all2-Cons2
  apply auto
  apply (fastforce dest!: list-all2-set1)
  apply blast

```

**using**  $\langle \text{infs } (\lambda x. x \in \bigcup a) (y \#\# ys) \rangle$   
**by** (*simp add: sdrop-shift*)  
**qed**

**lemma** *Steps-run-cycle''*:

$\exists x xs. \text{run } (x \#\# xs) \wedge x \in \bigcup (\text{closure } a_0)$   
 $\wedge (\forall x \in \text{sset } xs. \exists a \in \text{set } as \cup \{a\} \cup \text{set } bs. x \in \bigcup (\text{closure } a))$   
 $\wedge \text{infs } (\lambda x. x \in \bigcup (\text{closure } a)) (x \#\# xs)$   
**if** *assms*: *Steps* ( $a_0 \# as @ a \# bs @ [a]$ ) *P2 a*

**proof** –

**from** *Steps-Union*[*OF assms(1)*] **have** *post-defs.Steps* (*map closure* ( $a_0 \# as @ a \# bs @ [a]$ ))

**by** *simp*

**from** *Steps-run-cycle2*[*OF this[simplified] closure-non-empty[OF <P2 a>]*]

**show** *?thesis*

**by** *clarify (auto simp: image-def intro!: exI conjI)*

**qed**

**Unused lemma** *post-Steps-P1*:

*P1 x if post-defs.Steps* ( $a \# as$ )  $x \in b$   $b \in \text{set } as$

**using** *that*

**proof** (*induction a # as arbitrary: a as*)

**case** *Single*

**then show** *?case* **by** *auto*

**next**

**case** (*Cons a c as*)

**then show** *?case* **by** (*auto simp: A2'-def closure-def*)

**qed**

**lemma** *strong-compatibility-impl-weak*:

**fixes**  $\varphi :: 'a \Rightarrow \text{bool}$  — The property we want to check

**assumes**  $\varphi\text{-closure-compatible}$ :  $\bigwedge x a. x \in a \Longrightarrow \varphi x \longleftrightarrow (\forall x \in \bigcup (\text{closure } a). \varphi x)$

**shows**  $\varphi x \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P1 a \Longrightarrow \varphi y$

**by** (*auto simp: closure-def dest:  $\varphi\text{-closure-compatible}$* )

**end**

## 7.5 Finite Graphs

**context** *Finite-Graph*

**begin**

### 7.5.1 Infinite Büchi Runs Correspond to Finite Cycles

**lemma** *run-finite-state-set*:

**assumes**  $run\ (x_0\ \#\#\ xs)$   
**shows**  $finite\ (sset\ (x_0\ \#\#\ xs))$

**proof** –

**let**  $?S = \{x. E^{**}\ x_0\ x\}$

**from**  $run\ reachable[OF\ assms]$  **have**  $sset\ xs \subseteq ?S$  **unfolding**  $stream.pred\ set$   
**by** *auto*

**moreover** **have**  $finite\ ?S$  **using**  $finite\ reachable$  **by** *auto*

**ultimately** **show** *?thesis* **by** (*auto* *intro: finite-subset*)

**qed**

**lemma** *run-finite-state-set-cycle*:

**assumes**  $run\ (x_0\ \#\#\ xs)$

**shows**

$\exists\ ys\ zs. run\ (x_0\ \#\#\ ys\ @-\ cycle\ zs) \wedge set\ ys \cup set\ zs \subseteq \{x_0\} \cup sset\ xs$   
 $\wedge\ zs \neq []$

**proof** –

**from**  $run\ finite\ state\ set[OF\ assms]$  **have**  $finite\ (sset\ (x_0\ \#\#\ xs))$  .

**with**  $sdistinct\ infinite\ sset[of\ x_0\ \#\#\ xs]$   $not\ sdistinct\ decomp[of\ x_0\ \#\#\ xs]$  **obtain**  $x\ ws\ ys\ zs$

**where**  $x_0\ \#\#\ xs = ws\ @-\ x\ \#\#\ ys\ @-\ x\ \#\#\ zs$

**by** *force*

**then** **have**  $decomp: x_0\ \#\#\ xs = (ws\ @\ [x])\ @-\ ys\ @-\ x\ \#\#\ zs$  **by** *simp*

**from**  $run\ decomp[OF\ assms[unfolded\ decomp]]$  **have**  $decomp\ first:$

$steps\ (ws\ @\ [x])$

$run\ (ys\ @-\ x\ \#\#\ zs)$

$x \rightarrow (if\ ys = []\ then\ shd\ (x\ \#\#\ zs)\ else\ hd\ ys)$

**by** *auto*

**from**  $run\ sdrop[OF\ assms, of\ length\ (ws\ @\ [x])]$  **have**  $run\ (sdrop\ (length\ ws)\ xs)$

**by** *simp*

**moreover** **from**  $decomp$  **have**  $sdrop\ (length\ ws)\ xs = ys\ @-\ x\ \#\#\ zs$

**by** (*cases\ ws; simp\ add: sdrop-shift*)

**ultimately** **have**  $run\ ((ys\ @\ [x])\ @-\ zs)$  **by** *simp*

**from**  $run\ decomp[OF\ this]$  **have**  $steps\ (ys\ @\ [x])\ run\ zs\ x \rightarrow shd\ zs$

**by** *auto*

**from**  $run\ cycle[OF\ this(1)]$   $decomp\ first$  **have**

$run\ (cycle\ (ys\ @\ [x]))$

**by** (*force\ split: if-split-asm*)

**with**

$extend\ run[of\ (ws\ @\ [x])\ if\ ys = []\ then\ shd\ (x\ \#\#\ zs)\ else\ hd\ ys\ stl$   
 $(cycle\ (ys\ @\ [x]))]$

*decomp-first*  
**have**  
 run ((ws @ [x]) @- cycle (ys @ [x]))  
**apply** (simp split: if-split-asm)  
**subgoal**  
 using cycle-Cons[of x [], simplified] by auto  
**apply** (cases ys)  
**apply** (simp; fail)  
**by** (simp add: cycle-Cons)  
**with** *decomp* **show** ?thesis  
**apply** (inst-existentials tl (ws @ [x]) (ys @ [x]))  
**by** (cases ws; force)+  
**qed**

**lemma** *buechi-run-finite-state-set-cycle:*

**assumes** run (x<sub>0</sub> ## xs) alw (ev (holds φ)) (x<sub>0</sub> ## xs)

**shows**

∃ ys zs.

run (x<sub>0</sub> ## ys @- cycle zs) ∧ set ys ∪ set zs ⊆ {x<sub>0</sub>} ∪ sset xs  
 ∧ zs ≠ [] ∧ (∃ x ∈ set zs. φ x)

**proof** –

**from** *run-finite-state-set*[OF *assms*(1)] **have** finite (sset (x<sub>0</sub> ## xs)) .

**with** *sset-sfilter*[OF <alw (ev -) ->] **have** finite (sset (sfilter φ (x<sub>0</sub> ## xs)))

**by** (rule finite-subset)

**from** *finite-sset-sfilter-decomp*[OF *this* *assms*(2)] **obtain** x ws ys zs **where**

*decomp*: x<sub>0</sub> ## xs = (ws @ [x]) @- ys @- x ## zs **and** φ x

**by** *simp* *metis*

**from** *run-decomp*[OF *assms*(1)[*unfolded decomp*]] **have** *decomp-first*:

*steps* (ws @ [x])

run (ys @- x ## zs)

x → (if ys = [] then shd (x ## zs) else hd ys)

**by** *auto*

**from** *run-sdrop*[OF *assms*(1), of length (ws @ [x])] **have** run (sdrop (length ws) xs)

**by** *simp*

**moreover from** *decomp* **have** sdrop (length ws) xs = ys @- x ## zs

**by** (cases ws; *simp* add: *sdrop-shift*)

**ultimately have** run ((ys @ [x]) @- zs) **by** *simp*

**from** *run-decomp*[OF *this*] **have** *steps* (ys @ [x]) run zs x → shd zs

**by** *auto*

**from** *run-cycle*[OF *this*(1)] *decomp-first* **have**

run (cycle (ys @ [x]))

**by** (*force split: if-split-asm*)  
**with**  
*extend-run*[*of* ( $ws @ [x]$ ) *if*  $ys = []$  *then* *shd* ( $x \#\# zs$ ) *else* *hd*  $ys$  *stl*  
(*cycle* ( $ys @ [x]$ ))]  
*decomp-first*  
**have**  
*run* ( $(ws @ [x]) @- cycle (ys @ [x])$ )  
**apply** (*simp split: if-split-asm*)  
**subgoal**  
**using** *cycle-Cons*[*of*  $x []$ , *simplified*] **by** *auto*  
**apply** (*cases ys*)  
**apply** (*simp; fail*)  
**by** (*simp add: cycle-Cons*)  
**with** *decomp*  $\langle \varphi x \rangle$  **show** *?thesis*  
**apply** (*inst-existentials tl* ( $ws @ [x]$ ) ( $ys @ [x]$ ))  
**by** (*cases ws; force*)  
**qed**

**lemma** *run-finite-state-set-cycle-steps*:

**assumes** *run* ( $x_0 \#\# xs$ )  
**shows**  $\exists x ys zs. steps (x_0 \# ys @ x \# zs @ [x]) \wedge \{x\} \cup set ys \cup set zs$   
 $\subseteq \{x_0\} \cup sset xs$   
**proof** –  
**from** *run-finite-state-set-cycle*[*OF* *assms*] **obtain**  $ys zs$  **where** *guessed*:  
*run* ( $x_0 \#\# ys @- cycle zs$ )  
 $set ys \cup set zs \subseteq \{x_0\} \cup sset xs$   
 $zs \neq []$   
**by** *auto*  
**from**  $\langle zs \neq [] \rangle$  **have**  $cycle zs = (hd zs \# tl zs @ [hd zs]) @- cycle (tl zs$   
 $@ [hd zs])$   
**apply** (*cases zs*)  
**apply** (*simp; fail*)  
**apply** *simp*  
**apply** (*subst cycle-Cons[symmetric]*)  
**apply** (*subst cycle-decomp*)  
**by** *simp+*  
**from** *guessed*(1)[*unfolded this*] **have**  
*run* ( $(x_0 \# ys @ hd zs \# tl zs @ [hd zs]) @- cycle (tl zs @ [hd zs])$ )  
**by** *simp*  
**from** *run-decomp*[*OF* *this*] *guessed*(2,3) **show** *?thesis*  
**by** (*inst-existentials hd zs ys tl zs*) (*auto dest: list.set-sel*)  
**qed**

**lemma** *buechi-run-finite-state-set-cycle-steps*:  
**assumes**  $run\ (x_0\ \#\#\ xs)\ alw\ (ev\ (holds\ \varphi))\ (x_0\ \#\#\ xs)$   
**shows**  
 $\exists\ x\ ys\ zs.$   
 $steps\ (x_0\ \#\ ys\ @\ x\ \#\ zs\ @\ [x])\ \wedge\ \{x\}\ \cup\ set\ ys\ \cup\ set\ zs\ \subseteq\ \{x_0\}\ \cup\ sset\ xs$   
 $\wedge\ (\exists\ y\ \in\ set\ (x\ \#\ zs).\ \varphi\ y)$   
**proof** –  
**from** *buechi-run-finite-state-set-cycle*[*OF assms*] **obtain**  $ys\ zs\ x$  **where**  
*guessed*:  
 $run\ (x_0\ \#\#\ ys\ @-\ cycle\ zs)$   
 $set\ ys\ \cup\ set\ zs\ \subseteq\ \{x_0\}\ \cup\ sset\ xs$   
 $zs\ \neq\ []$   
 $x\ \in\ set\ zs$   
 $\varphi\ x$   
**by** *safe*  
**from**  $\langle zs\ \neq\ [] \rangle$  **have**  $cycle\ zs = (hd\ zs\ \#\ tl\ zs\ @\ [hd\ zs])\ @-\ cycle\ (tl\ zs\ @\ [hd\ zs])$   
**apply** (*cases zs*)  
**apply** (*simp; fail*)  
**apply** *simp*  
**apply** (*subst cycle-Cons[symmetric]*)  
**apply** (*subst cycle-decomp*)  
**by** *simp+*  
**from** *guessed(1)[unfolded this]* **have**  
 $run\ ((x_0\ \#\ ys\ @\ hd\ zs\ \#\ tl\ zs\ @\ [hd\ zs])\ @-\ cycle\ (tl\ zs\ @\ [hd\ zs]))$   
**by** *simp*  
**from** *run-decomp[OF this] guessed(2,3,4,5)* **show** *?thesis*  
**by** (*inst-existentials hd zs ys tl zs*) (*auto 4 4 dest: list.set-sel*)  
**qed**

**lemma** *cycle-steps-run*:  
**assumes**  $steps\ (x_0\ \#\ ys\ @\ x\ \#\ zs\ @\ [x])$   
**shows**  $\exists\ xs.\ run\ (x_0\ \#\#\ xs)\ \wedge\ sset\ xs = \{x\}\ \cup\ set\ ys\ \cup\ set\ zs$   
**proof** –  
**from** *assms* **have**  $steps\ (x_0\ \#\ ys\ @\ [x])\ steps\ (x\ \#\ zs\ @\ [x])$   
  
**apply** (*metis Graph-Defs.steps-appendD1 append.assoc append-Cons append-Nil snoc-eq-iff-butlast*)  
**by** (*metis Graph-Defs.steps-appendD2 append-Cons assms snoc-eq-iff-butlast*)  
  
**from** *this(2)* **have**  $x\ \rightarrow\ hd\ (zs\ @\ [x])\ steps\ (zs\ @\ [x])$   
**apply** (*metis Graph-Defs.steps-decomp last-snoc list.sel(1) list.sel(3) snoc-eq-iff-butlast steps-ConsD steps-append'*)

```

    by (meson steps-ConsD ⟨steps (x # zs @ [x])⟩ snoc-eq-iff-butlast)
  from run-cycle[OF this(2)] this(1) have run (cycle (zs @ [x])) by auto
  with extend-run[OF ⟨steps (x₀ # ys @ [x])⟩, of hd (zs @ [x]) stl (cycle
(zs @ [x]))] ⟨x → -⟩
  have run (x₀ ## ys @- x ## cycle (zs @ [x]))
    by simp (metis cycle.ctr)
  then show ?thesis
    by auto
qed

```

**lemma** *buechi-run-lasso*:

```

  assumes run (x₀ ## xs) alw (ev (holds φ)) (x₀ ## xs)
  obtains x where reaches x₀ x reaches1 x x φ x
proof -
  from buechi-run-finite-state-set-cycle-steps[OF assms] obtain x ys zs y
  where
    steps (x₀ # ys @ x # zs @ [x]) y ∈ set (x # zs) φ y
    by safe
  from ⟨y ∈ -⟩ consider y = x | as bs where zs = as @ y # bs
    by (meson set-ConsD split-list)
  then have ∃ as bs. steps (x₀ # as @ [y]) ∧ steps (y # bs @ [y])
  proof cases
    case 1

    with ⟨steps -⟩ show ?thesis
      by simp (metis Graph-Defs.steps-appendD2 append.assoc append-Cons
list.distinct(1))
    next
      case 2
      with ⟨steps -⟩ show ?thesis
        by simp (metis (no-types)
reaches1-steps steps-reaches append-Cons last-appendR list.distinct(1)
list.sel(1)
reaches1-reaches-iff2 reaches1-steps-append steps-decomp)
    qed
    with ⟨φ y⟩ show ?thesis
      including graph-automation by (intro that[of y]) (auto intro: steps-reaches1)
    qed
  end
end

```

## 7.6 Complete Simulation Graphs

**context** *Simulation-Graph-Defs*

**begin**

**definition** *abstract-run*  $x\ xs = x\ \#\#\ sscan\ (\lambda\ y\ a.\ SOME\ b.\ A\ a\ b\ \wedge\ y\ \in\ b)\ xs\ x$

**lemma** *abstract-run-ctr*:

*abstract-run*  $x\ xs = x\ \#\#\ abstract-run\ (SOME\ b.\ A\ x\ b\ \wedge\ shd\ xs\ \in\ b)\ (stl\ xs)$

**unfolding** *abstract-run-def* **by** (*subst sscan.ctr*) (*rule HOL.refl*)

**end**

**context** *Simulation-Graph-Complete*

**begin**

**lemma** *steps-complete*:

$\exists\ as.\ Steps\ (a\ \#\ as)\ \wedge\ list-all2\ (\in)\ xs\ as$  **if** *steps*  $(x\ \#\ xs)\ x\ \in\ a\ P\ a$   
**using that** **by** (*induction xs arbitrary: x a*) (*erule steps.cases; fastforce dest!: complete*)**+**

**lemma** *abstract-run-Run*:

*Run*  $(abstract-run\ a\ xs)$  **if** *run*  $(x\ \#\#\ xs)\ x\ \in\ a\ P\ a$

**using that**

**proof** (*coinduction arbitrary: a x xs*)

**case**  $(run\ a\ x\ xs)$

**obtain**  $y\ ys$  **where**  $xs = y\ \#\#\ ys$  **by** (*metis stream.collapse*)

**with** *run* **have**  $C\ x\ y\ run\ (y\ \#\#\ ys)$  **by** (*auto elim: run.cases*)

**from** *complete*[*OF*  $\langle C\ x\ y\rangle - \langle P\ a\rangle \langle x\ \in\ a\rangle$ ] **obtain**  $b$  **where**  $A\ a\ b\ \wedge\ y\ \in\ b$  **by** *auto*

**then have**  $A\ a\ (SOME\ b.\ A\ a\ b\ \wedge\ y\ \in\ b)\ \wedge\ y\ \in\ (SOME\ b.\ A\ a\ b\ \wedge\ y\ \in\ b)$  **by** (*rule someI*)

**moreover with**  $\langle P\ a\rangle$  **have**  $P\ (SOME\ b.\ A\ a\ b\ \wedge\ y\ \in\ b)$  **by** (*blast intro: P-invariant*)

**ultimately show** *?case* **using**  $\langle run\ (y\ \#\#\ ys)\rangle$  **unfolding**  $\langle xs = -\rangle$

**apply** (*subst abstract-run-ctr, simp*)

**apply** (*subst abstract-run-ctr, simp*)

**by** (*auto simp: abstract-run-ctr[symmetric]*)

**qed**

**lemma** *abstract-run-abstract*:

*stream-all2*  $(\in)\ (x\ \#\#\ xs)\ (abstract-run\ a\ xs)$  **if** *run*  $(x\ \#\#\ xs)\ x\ \in\ a\ P\ a$

**using that** **proof** (*coinduction arbitrary: a x xs*)

**case** *run*: (*stream-rel x' u b' v a x xs*)

**obtain**  $y$   $ys$  **where**  $xs = y \#\# ys$  **by** (*metis stream.collapse*)  
**with**  $run$  **have**  $C x y run (y \#\# ys)$  **by** (*auto elim: run.cases*)  
**from**  $complete[OF \langle C x y \rangle - \langle P a \rangle \langle x \in a \rangle]$  **obtain**  $b$  **where**  $A a b \wedge y \in b$  **by** *auto*  
**then have**  $A a (SOME b. A a b \wedge y \in b) \wedge y \in (SOME b. A a b \wedge y \in b)$  **by** (*rule someI*)  
**with**  $\langle run (y \#\# ys) \rangle \langle x \in a \rangle \langle P a \rangle run(1,2) \langle xs = - \rangle$  **show** *?case*  
**by** (*subst (asm) abstract-run-ctr*) (*auto intro: P-invariant*)  
**qed**

**lemma** *run-complete*:

$\exists as. Run (a \#\# as) \wedge stream-all2 (\in) xs as$  **if**  $run (x \#\# xs) x \in a P a$   
**using** *abstract-run-Run[OF that] abstract-run-abstract[OF that]*  
**apply** (*subst (asm) abstract-run-ctr*)  
**apply** (*subst (asm) (2) abstract-run-ctr*)  
**by** *auto*

**end**

### 7.6.1 Runs in Finite Complete Graphs

**context** *Simulation-Graph-Finite-Complete*  
**begin**

**lemma** *run-finite-state-set-cycle-steps*:

**assumes**  $run (x_0 \#\# xs) x_0 \in a_0 P a_0$   
**shows**  $\exists x ys zs.$   
 $Steps (a_0 \# ys @ x \# zs @ [x]) \wedge (\forall a \in \{x\} \cup set ys \cup set zs. \exists x \in \{x_0\} \cup sset xs. x \in a)$   
**using** *run-complete[OF assms]*  
**apply** *safe*  
**apply** (*drule Steps-finite.run-finite-state-set-cycle-steps*)  
**apply** *safe*  
**subgoal for**  $as x ys zs$   
**apply** (*inst-existentials x ys zs*)  
**using** *assms(2) by (auto dest: stream-all2-sset2)*  
**done**

**lemma** *buechi-run-finite-state-set-cycle-steps*:

**assumes**  $run (x_0 \#\# xs) x_0 \in a_0 P a_0 alw (ev (holds \varphi)) (x_0 \#\# xs)$   
**shows**  $\exists x ys zs.$   
 $Steps (a_0 \# ys @ x \# zs @ [x])$   
 $\wedge (\forall a \in \{x\} \cup set ys \cup set zs. \exists x \in \{x_0\} \cup sset xs. x \in a)$   
 $\wedge (\exists y \in set (x \# zs). \exists a \in y. \varphi a)$

```

using run-complete[OF assms(1-3)]
apply safe
apply (drule Steps-finite.buechi-run-finite-state-set-cycle-steps[where  $\varphi$ 
=  $\lambda S. \exists x \in S. \varphi x$ ])
subgoal for as
  using assms(4)
  apply (subst alw-ev-stl[symmetric], simp)
  apply (erule alw-stream-all2-mono[where  $Q = \text{ev } (\text{holds } \varphi)$ ], fastforce)
  by (metis (mono-tags, lifting) ev-holds-sset stream-all2-sset1)
apply safe
subgoal for as x ys zs y a
  apply (inst-existentials x ys zs)
  using assms(2) by (auto dest: stream-all2-sset2)
done

```

**lemma** *buechi-run-finite-state-set-cycle-lasso*:

```

assumes run ( $x_0 \## xs$ )  $x_0 \in a_0$   $P a_0$  alw (ev (holds  $\varphi$ )) ( $x_0 \## xs$ )
shows  $\exists a. \text{Steps.reaches } a_0 a \wedge \text{Steps.reaches1 } a a \wedge (\exists y \in a. \varphi y)$ 

```

**proof** –

```

from buechi-run-finite-state-set-cycle-steps[OF assms] obtain b as bs a y

```

**where** *lasso*:

```

  Steps ( $a_0 \# as @ b \# bs @ [b]$ )  $a \in \text{set } (b \# bs)$   $y \in a$   $\varphi y$ 
  by safe

```

```

from  $\langle a \in \text{set} \rightarrow$  consider  $b = a \mid bs1 \ bs2$  where  $bs = bs1 @ a \# bs2$ 
  using split-list by fastforce

```

```

then have Steps.reaches  $a_0 a \wedge \text{Steps.reaches1 } a a$ 

```

```

  using  $\langle \text{Steps} \rightarrow$ 

```

```

  apply cases

```

```

  apply safe

```

```

  subgoal

```

```

    by (simp add: Steps.steps-reaches')

```

```

  subgoal

```

```

    by (blast dest: Steps.stepsD intro: Steps.steps-reaches1)

```

```

  subgoal for bs1 bs2

```

```

    by (subgoal-tac Steps (( $a_0 \# as @ b \# bs1 @ [a]$ ) @ ( $bs2 @ [b]$ )))
    (drule Steps.stepsD, auto elim: Steps.steps-reaches')

```

```

  subgoal

```

```

    by (metis (no-types)

```

```

      Steps.steps-reaches1 Steps.steps-rotate Steps-appendD2 append-Cons

```

```

append-eq-append-conv2

```

```

      list.distinct(1))

```

```

  done

```

```

with lasso show ?thesis

```

```

  by auto

```

qed

end

## 7.7 Finite Complete Double Simulations

**context** *Double-Simulation*

**begin**

**lemma** *Run-closure:*

*post-defs.Run (smap closure xs) if Run xs*

**using that proof** (*coinduction arbitrary: xs*)

**case** *prems: run*

**then obtain**  $x\ y\ ys$  **where**  $xs = x \#\# y \#\# ys$   $A2\ x\ y\ Run\ (y \#\# ys)$

**by** (*auto elim: Steps.run.cases*)

**with**  $A2'$ - $A2$ -closure[*OF*  $\langle A2\ x\ y \rangle$ ] **show** *?case*

**by** *force*

qed

**lemma** *closure-set-finite:*

*finite (closure ' UNIV) (is finite ?S)*

**proof** –

**have**  $?S \subseteq \{x. x \subseteq \{x. P1\ x}\}$

**unfolding** *closure-def* **by** *auto*

**also have** *finite ...*

**using**  $P1$ -*finite* **by** *auto*

**finally show** *?thesis* .

qed

**lemma**  $A2'$ -*empty-step:*

$b = \{\}$  **if**  $A2'\ a\ b\ a = \{\}$

**using that** *closure-poststable* **unfolding**  $A2'$ -*def* **by** *auto*

**lemma**  $A2'$ -*empty-invariant:*

*Graph-Invariant*  $A2'\ (\lambda x. x = \{\})$

**by** *standard* (*rule*  $A2'$ -*empty-step*)

end

**context** *Double-Simulation-Complete*

**begin**

**lemmas**  $P2$ -*invariant-Steps* =  $P2$ -*invariant.invariant-steps*

**interpretation** *Steps-finite: Finite-Graph A2' closure a<sub>0</sub>*

**proof**

**have**  $\{x. \text{post-defs.Steps.reaches } (\text{closure } a_0) x\} \subseteq \text{closure 'UNIV}$

**by** (*auto 4 3 simp: A2'-def elim: rtranclp.cases*)

**also have** *finite ...*

**by** (*fact closure-set-finite*)

**finally show** *finite*  $\{x. \text{post-defs.Steps.reaches } (\text{closure } a_0) x\}$  .

**qed**

**theorem** *infinite-run-cycle-iff'*:

**assumes**  $\bigwedge x xs. \text{run } (x \## xs) \implies x \in \bigcup (\text{closure } a_0) \implies \exists y ys. y \in a_0 \wedge \text{run } (y \## ys)$

**shows**

$(\exists x_0 xs. x_0 \in \bigcup (\text{closure } a_0) \wedge \text{run } (x_0 \## xs)) \longleftrightarrow$

$(\exists as a bs. \text{post-defs.Steps } (\text{closure } a_0 \# as @ a \# bs @ [a]) \wedge a \neq \{\})$

**proof** (*safe, goal-cases*)

**case** *prems: (1 x<sub>0</sub> X xs)*

**from** *assms[OF prems(1)] prems(2,3)* **obtain** *y ys where*  $y \in a_0 \text{run } (y \## ys)$

**by** *auto*

**from** *run-complete[OF this(2,1) P2-a<sub>0</sub>]* **obtain** *as where*  $\text{Run } (a_0 \## as) \text{stream-all2 } (\in) ys as$

**by** *auto*

**from** *P2-invariant.invariant-run[OF <Run ->]* **have**  $*$ :  $\forall a \in \text{sset } (a_0 \## as). P2 a$

**unfolding** *stream.pred-set* **by** *auto*

**from** *Steps-finite.run-finite-state-set-cycle-steps[OF Run-closure[OF <Run ->, simplified]]* **show** *?case*

**using** *<stream-all2 - - -> <y ∈ -> \* closure-non-empty* **by** *force+*

**next**

**case** *prems: (2 as a bs x)*

**with** *post-defs.Steps.steps-decomp[of closure a<sub>0</sub> # as @ [a] bs @ [a]]* **have**  $\text{post-defs.Steps } (\text{closure } a_0 \# as @ [a]) \text{post-defs.Steps } (bs @ [a]) A2' a$   
 $(\text{hd } (bs @ [a]))$

**by** *auto*

**from** *prems(2,3) Steps-run-cycle2[OF prems(1)]* **show** *?case*

**by** *auto*

**qed**

**corollary** *infinite-run-cycle-iff*:

$(\exists x_0 xs. x_0 \in a_0 \wedge \text{run } (x_0 \## xs)) \longleftrightarrow$

$(\exists as a bs. \text{post-defs.Steps } (\text{closure } a_0 \# as @ a \# bs @ [a]) \wedge a \neq \{\})$

**if**  $\bigcup (\text{closure } a_0) = a_0 P2 a_0$

**by** (*subst <- = a<sub>0</sub>>[symmetric]*) (*rule infinite-run-cycle-iff', auto simp:*

that)

**context**

**fixes**  $\varphi :: 'a \Rightarrow \text{bool}$  — The property we want to check

**assumes**  $\varphi$ -closure-compatible:  $P2\ a \Longrightarrow x \in \bigcup (\text{closure } a) \Longrightarrow \varphi\ x \longleftrightarrow$   
 $(\forall x \in \bigcup (\text{closure } a). \varphi\ x)$

**begin**

We need the condition  $a \neq \{\}$  in the following theorem because we cannot prove a lemma like this:

**lemma**

$\exists bs. \text{Steps } bs \wedge \text{closure } a \# as = \text{map closure } bs \text{ if } \text{post-defs.Steps } (\text{closure } a \# as)$

**using** that

**oops**

One possible fix would be to add the stronger assumption  $A2\ a\ b \Longrightarrow P2\ b$ .

**theorem** *infinite-buechi-run-cycle-iff-closure*:

**assumes**

$\bigwedge x\ xs. \text{run } (x \#\# xs) \Longrightarrow x \in \bigcup (\text{closure } a_0) \Longrightarrow \text{alw } (\text{ev } (\text{holds } \varphi))\ xs$   
 $\Longrightarrow \exists y\ ys. y \in a_0 \wedge \text{run } (y \#\# ys) \wedge \text{alw } (\text{ev } (\text{holds } \varphi))\ ys$   
**and**  $\bigwedge a. P2\ a \Longrightarrow a \subseteq \bigcup (\text{closure } a)$

**shows**

$(\exists x_0\ xs. x_0 \in \bigcup (\text{closure } a_0) \wedge \text{run } (x_0 \#\# xs) \wedge \text{alw } (\text{ev } (\text{holds } \varphi))\ (x_0 \#\# xs))$

$\longleftrightarrow (\exists as\ a\ bs. a \neq \{\} \wedge \text{post-defs.Steps } (\text{closure } a_0 \# as @ a \# bs @ [a]) \wedge (\forall x \in \bigcup a. \varphi\ x))$

**proof** (*safe, goal-cases*)

**case** *prems*: (1  $x_0\ xs$ )

**from** *assms*(1)[*OF* *prems*(3)] *prems*(1,2,4) **obtain**  $y\ ys$  **where**

$y \in a_0 \text{run } (y \#\# ys) \text{alw } (\text{ev } (\text{holds } \varphi))\ ys$

**by** *auto*

**from** *run-complete*[*OF* *this*(2,1) *P2-a<sub>0</sub>*] **obtain**  $as$  **where**  $\text{Run } (a_0 \#\# as) \text{stream-all2 } (\in) ys\ as$

**by** *auto*

**from** *P2-invariant.invariant-run*[*OF*  $\langle \text{Run } \rightarrow \rangle$ ] **have** *pred-stream*  $P2\ (a_0 \#\# as)$

**by** *auto*

**from** *Run-closure*[*OF*  $\langle \text{Run } \rightarrow \rangle$ ] **have** *post-defs.Run*  $(\text{closure } a_0 \#\# \text{smap closure } as)$

**by** *simp*

**from**  $\langle \text{alw } (\text{ev } (\text{holds } \varphi))\ ys \rangle \langle \text{stream-all2 } - - \rightarrow \rangle$  **have**  $\text{alw } (\text{ev } (\text{holds } (\lambda a. \exists x \in a. \varphi\ x)))\ as$

**by** (*rule* *alw-ev-lockstep*) *auto*

```

then have alw (ev (holds ( $\lambda a. \exists x \in \bigcup a. \varphi x$ ))) (closure  $a_0$   $\#\#$  smap
closure as)
  apply -
  apply rule
  apply (rule alw-ev-lockstep[where  $Q = \lambda a b. b = \text{closure } a \wedge P2 a$ ],
assumption)
  subgoal
    using  $\langle \text{Run } (a_0 \#\# as) \rangle$ 
    by - (rule stream-all2-combine[where  $P = \text{eq-onp } P2$  and  $Q = \lambda a$ 
b. b = closure a],
      subst stream.pred-rel[symmetric],
      auto dest: P2-invariant.invariant-run simp: stream.rel-refl eq-onp-def
    )
  subgoal for a x
    by (auto dest!: assms(2))
  done
from Steps-finite.buechi-run-finite-state-set-cycle-steps[OF  $\langle \text{post-defs.Run}$ 
(-  $\#\#$  -) $\rangle$  this]
obtain a ys zs where guessed:
  post-defs.Steps (closure  $a_0 \# ys @ a \# zs @ [a]$ )
   $a = \text{closure } a_0 \vee a \in \text{closure } 'sset as$ 
   $set ys \subseteq \text{insert } (\text{closure } a_0) (\text{closure } 'sset as)$ 
   $set zs \subseteq \text{insert } (\text{closure } a_0) (\text{closure } 'sset as)$ 
   $(\exists y \in a. \exists x \in y. \varphi x) \vee (\exists y \in set zs. \exists y' \in y. \exists x \in y'. \varphi x)$ 
  by clarsimp
from guessed(5) show ?case
proof (standard, goal-cases)
  case prems: 1
    from guessed(1) have post-defs.Steps (closure  $a_0 \# ys @ [a]$ )
      by (metis
        Graph-Defs.graphI(3) Graph-Defs.steps-decomp append.simps(2)
list.sel(1) list.simps(3)
      )
    from  $\langle \text{pred-stream - -} \rangle$  guessed(2) obtain a' where  $a = \text{closure } a' P2$ 
a'
      by (auto simp: stream.pred-set)
    from prems obtain x R where  $x \in R R \in a \varphi x$  by auto
    with  $\langle P2 a' \rangle$  have  $\forall x \in \bigcup a. \varphi x$ 
      unfolding  $\langle a = - \rangle$  by (subst  $\varphi$ -closure-compatible[symmetric]) auto
    with guessed(1,2) show ?case
      using  $\langle R \in a \rangle$  by blast
  next
  case prems: 2
    then obtain R b x where  $*$ :  $x \in R R \in b b \in set zs \varphi x$ 

```

```

    by auto
  from ⟨b ∈ set zs⟩ obtain zs1 zs2 where zs = zs1 @ b # zs2 by (force
simp: split-list)
    with guessed(1) have post-defs.Steps ((closure a₀ # ys @ a # zs1 @
[b]) @ zs2 @ [a])
      by simp
    with guessed(1) have post-defs.Steps (closure a₀ # ys @ a # zs1 @ [b])
      by - (drule Graph-Defs.steps-decomp, auto)
    from ⟨pred-stream -> guessed(4) ⟨zs = ->⟩ obtain b' where b = closure
b' P2 b'
      by (auto simp: stream.pred-set)
    with * have *: ∀ x ∈ ⋃ b. φ x
      unfolding ⟨b = ->⟩ by (subst φ-closure-compatible[symmetric]) auto
    from ⟨zs = -> guessed(1) have post-defs.Steps ((closure a₀ # ys) @ (a
# zs1 @ [b]) @ zs2 @ [a])
      by simp
    then have post-defs.Steps (a # zs1 @ [b]) by (blast dest!: post-defs.Steps.steps-decomp)
    with ⟨zs = -> guessed * show ?case
      using
        ⟨R ∈ b⟩
        post-defs.Steps.steps-append[of closure a₀ # ys @ a # zs1 @ b # zs2
@ [a] a # zs1 @ [b]]
        by (inst-existentials ys @ a # zs1 b zs2 @ a # zs1) auto
    qed
next
  case prems: (2 as a bs x)
  then have a ≠ {}
    by auto
  from prems post-defs.Steps.steps-decomp[of closure a₀ # as @ [a] bs @
[a]] have
    post-defs.Steps (closure a₀ # as @ [a])
      by auto
  with Steps-run-cycle2[OF prems(1) ⟨a ≠ {}⟩] prems show ?case
    unfolding HLD-iff by clarify (drule alw-ev-mono[where ψ = holds φ],
auto)
  qed
end

end

end

context Double-Simulation-Finite-Complete
begin

```

**lemmas** *P2-invariant-Steps = P2-invariant.invariant-steps*

**theorem** *infinite-run-cycle-iff'*:

**assumes**  $P2\ a_0 \wedge x\ xs.\ run\ (x\ \#\#\ xs) \implies x \in \bigcup(\text{closure } a_0) \implies \exists\ y\ ys.\ y \in a_0 \wedge run\ (y\ \#\#\ ys)$

**shows**  $(\exists\ x_0\ xs.\ x_0 \in a_0 \wedge run\ (x_0\ \#\#\ xs)) \longleftrightarrow (\exists\ as\ a\ bs.\ Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**proof** (*safe, goal-cases*)

**case** (1  $x_0\ xs$ )

**from** *run-finite-state-set-cycle-steps[OF this(2,1)]*  $\langle P2\ a_0 \rangle$  **show** *?case by auto*

**next**

**case** *prems: (2 as a bs)*

**with** *Steps.steps-decomp[of a\_0 # as @ [a] bs @ [a]]* **have** *Steps (a\_0 # as @ [a]) by auto*

**from** *P2-invariant-Steps[OF this]* **have** *P2 a by auto*

**from** *Steps-run-cycle''[OF prems this] assms(2)* **show** *?case by auto*

**qed**

**corollary** *infinite-run-cycle-iff*:

$(\exists\ x_0\ xs.\ x_0 \in a_0 \wedge run\ (x_0\ \#\#\ xs)) \longleftrightarrow (\exists\ as\ a\ bs.\ Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**if**  $\bigcup(\text{closure } a_0) = a_0\ P2\ a_0$

**by** (*rule infinite-run-cycle-iff', auto simp: that*)

**context**

**fixes**  $\varphi :: 'a \Rightarrow bool$  — The property we want to check

**assumes**  *$\varphi$ -closure-compatible:  $x \in a \implies \varphi\ x \longleftrightarrow (\forall\ x \in \bigcup(\text{closure } a).\ \varphi\ x)$*

**begin**

**theorem** *infinite-buechi-run-cycle-iff*:

$(\exists\ x_0\ xs.\ x_0 \in a_0 \wedge run\ (x_0\ \#\#\ xs) \wedge alw\ (ev\ (\text{holds } \varphi))\ (x_0\ \#\#\ xs))$

$\longleftrightarrow (\exists\ as\ a\ bs.\ Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]) \wedge (\forall\ x \in \bigcup(\text{closure } a).\ \varphi\ x))$

**if**  $\bigcup(\text{closure } a_0) = a_0$

**proof** (*safe, goal-cases*)

**case** (1  $x_0\ xs$ )

**from** *buechi-run-finite-state-set-cycle-steps[OF this(2,1) P2-a\_0, of  $\varphi$ ] this(3)*

**obtain**  $a\ ys\ zs$

**where**

*infs  $\varphi\ xs$*

*Steps (a\_0 # ys @ a # zs @ [a])*

*$x_0 \in a \vee (\exists\ x \in sset\ xs.\ x \in a)$*

```

     $\forall a \in \text{set } ys \cup \text{set } zs. x_0 \in a \vee (\exists x \in \text{set } xs. x \in a)$ 
     $(\exists x \in a. \varphi x) \vee (\exists y \in \text{set } zs. \exists x \in y. \varphi x)$ 
    by clarsimp
  note guessed = this(2-)
  from guessed(4) show ?case
  proof (standard, goal-cases)
    case 1
    then obtain x where  $x \in a \varphi x$  by auto
    with  $\varphi$ -closure-compatible have  $\forall x \in \bigcup (\text{closure } a). \varphi x$  by blast
    with guessed(1,2) show ?case by auto
  next
    case 2
    then obtain b x where  $x \in b b \in \text{set } zs \varphi x$  by auto
    with  $\varphi$ -closure-compatible have *:  $\forall x \in \bigcup (\text{closure } b). \varphi x$  by blast
    from  $\langle b \in \text{set } zs \rangle$  obtain zs1 zs2 where  $zs = zs1 @ b \# zs2$  by (force simp: split-list)
    with guessed(1) have Steps  $((a_0 \# ys) @ (a \# zs1 @ [b]) @ zs2 @ [a])$ 
  by simp
    then have Steps  $(a \# zs1 @ [b])$  by (blast dest!: Steps.steps-decomp)
    with  $\langle zs = \rightarrow \text{guessed} * \rangle$  show ?case
      apply (inst-existentials ys @ a \# zs1 b zs2 @ a \# zs1)
      using Steps.steps-append[of  $a_0 \# ys @ a \# zs1 @ b \# zs2 @ [a] a \#$ 
 $zs1 @ [b]$ ]
      by auto
    qed
  next
    case prems: (2 as a bs)
    with Steps.steps-decomp[of  $a_0 \# as @ [a] bs @ [a]$ ] have Steps  $(a_0 \# as$ 
 $@ [a])$  by auto
    from P2-invariant-Steps[OF this] have P2 a by auto
    from Steps-run-cycle''[OF prems(1) this] prems this that show ?case
      apply safe
      subgoal for x xs b
        by (inst-existentials x xs) (auto elim!: alw-ev-mono)
      done
    qed
  end
end

```

## 7.8 Encoding of Properties in Runs

This approach only works if we assume strong compatibility of the property. For weak compatibility, encoding in the automaton is likely the right way.

**context** *Double-Simulation-Complete-Abstraction-Prop*  
**begin**

**definition**  $C\text{-}\varphi\ x\ y \equiv C\ x\ y \wedge \varphi\ y$

**definition**  $A1\text{-}\varphi\ a\ b \equiv A1\ a\ b \wedge b \subseteq \{x.\ \varphi\ x\}$

**definition**  $A2\text{-}\varphi\ S\ S' \equiv \exists\ S''.\ A2\ S\ S'' \wedge S'' \cap \{x.\ \varphi\ x\} = S' \wedge S' \neq \{\}$

**lemma** *A2-φ-P2-invariant:*

*P2 a if A2-φ\*\* a<sub>0</sub> a*

**proof** –

**interpret** *invariant: Graph-Invariant-Start A2-φ a<sub>0</sub> P2*

**by** *standard (auto intro: φ-P2-compatible P2-invariant P2-a<sub>0</sub> simp: A2-φ-def)*

**from** *invariant.invariant-reaches[OF that] show ?thesis .*

**qed**

**sublocale** *phi: Double-Simulation-Complete C-φ A1-φ P1 A2-φ P2 a<sub>0</sub>*

**proof** (*standard, goal-cases*)

**case** (*1 S T*)

**then show** *?case unfolding A1-φ-def C-φ-def by (auto 4 4 dest: φ-A1-compatible prestable)*

**next**

**case** (*2 y b a*)

**then obtain** *c where A2 a c c ∩ {x. φ x} = b unfolding A2-φ-def by auto*

**with** *⟨y ∈ -⟩ have y ∈ closure c by (auto dest: closure-intD)*

**moreover have** *y ⊆ {x. φ x}*

**by** (*smt 2(1) φ-A1-compatible ⟨A2 a c⟩ ⟨c ∩ {x. φ x} = b⟩ ⟨y ∈ closure c⟩ closure-def*

*closure-poststable inf-assoc inf-bot-right inf-commute mem-Collect-eq*)

**ultimately show** *?case using ⟨A2 a c⟩ unfolding A1-φ-def A2-φ-def*

**by** (*auto dest: closure-poststable*)

**next**

**case** (*3 x y*)

**then show** *?case by (rule P1-distinct)*

**next**

**case** *4*

**then show** *?case by (rule P1-finite)*

**next**

**case** (*5 a*)

```

  then show ?case by (rule P2-cover)
next
  case (6 x y S)
  then show ?case unfolding C-φ-def A2-φ-def by (auto dest!: complete)
next
  case (7 a a')
  then show ?case unfolding A2-φ-def by (auto intro: P2-invariant φ-P2-compatible)
next
  case 8
  then show ?case by (rule P2-a0)
qed

```

**lemma** *phi-run-iff*:

$phi.run (x \#\# xs) \wedge \varphi x \longleftrightarrow run (x \#\# xs) \wedge pred-stream \varphi (x \#\# xs)$

**proof** –

**have** *phi.run xs if run xs pred-stream φ xs for xs*

**using** *that* **by** (coinduction arbitrary: xs) (auto elim: run.cases simp: C-φ-def)

**moreover have** *run xs if phi.run xs for xs*

**using** *that* **by** (coinduction arbitrary: xs) (auto elim: phi.run.cases simp: C-φ-def)

**moreover have** *pred-stream φ xs if phi.run (x ## xs) φ x*

**using** *that* **by** (coinduction arbitrary: xs x) (auto 4 3 elim: phi.run.cases simp: C-φ-def)

**ultimately show** *?thesis* **by** *auto*

**qed**

**end**

**context** *Double-Simulation-Finite-Complete-Abstraction-Prop*

**begin**

**sublocale** *phi: Double-Simulation-Finite-Complete C-φ A1-φ P1 A2-φ P2*

*a0*

**proof** (*standard, goal-cases*)

**case** *1*

**have**  $\{a. A2-\varphi^{**} a_0 a\} \subseteq \{a. Steps.reaches a_0 a\}$

**apply** *safe*

**subgoal premises** *prems* **for** *x*

**using** *prems*

**proof** (*induction x1 ≡ a0 x rule: rtranclp.induct*)

**case** *rtrancl-refl*

**then show** *?case* **by** *blast*

**next**

**case** *prems*: (*rtrancl-into-rtrancl* *b c*)  
**then have**  $c \neq \{\}$   
**by** – (*rule P2-non-empty, auto intro: A2-φ-P2-invariant*)  
**from**  $\langle A2\text{-}\varphi\ b\ c \rangle$  **obtain**  $S''\ x$  **where**  
 $A2\ b\ S''\ c = S'' \cap \{x. \varphi\ x\}\ x \in S''\ \varphi\ x$   
**unfolding** *A2-φ-def* **by** *auto*  
**with** *prems*  $\langle c \neq \{\} \rangle$  *φ-A2-compatible[of S'']* **show** *?case*  
**including** *graph-automation-aggressive* **by** *auto*  
**qed**  
**done**  
**then show** *?case* (**is finite** *?S*) **using** *finite-abstract-reachable* **by** (*rule finite-subset*)  
**qed**

**corollary** *infinite-run-cycle-iff*:

$(\exists\ x_0\ xs.\ x_0 \in a_0 \wedge run\ (x_0\ \#\#\ xs) \wedge pred\text{-stream}\ \varphi\ (x_0\ \#\#\ xs)) \longleftrightarrow$   
 $(\exists\ as\ a\ bs.\ phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**if**  $\bigcup(closure\ a_0) = a_0\ a_0 \subseteq \{x. \varphi\ x\}$

**unfolding** *phi.infinite-run-cycle-iff[OF that(1) P2-a<sub>0</sub>, symmetric]* *phi-run-iff[symmetric]*  
**using** *that(2)* **by** *auto*

**theorem** *Alw-ev-mc*:

$(\forall\ x_0 \in a_0.\ Alw\text{-}ev\ (Not\ o\ \varphi)\ x_0) \longleftrightarrow \neg (\exists\ as\ a\ bs.\ phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**if**  $\bigcup(closure\ a_0) = a_0\ a_0 \subseteq \{x. \varphi\ x\}$

**unfolding** *Alw-ev alw-holds-pred-stream-iff infinite-run-cycle-iff[OF that, symmetric]*

**by** (*auto simp: comp-def*)

**end**

**context** *Simulation-Graph-Defs*

**begin**

**definition** *represent-run*  $x\ as = x\ \#\#\ sscan\ (\lambda\ b\ x.\ SOME\ y.\ C\ x\ y \wedge y \in b)\ as\ x$

**lemma** *represent-run-ctr*:

$represent\text{-run}\ x\ as = x\ \#\#\ represent\text{-run}\ (SOME\ y.\ C\ x\ y \wedge y \in shd\ as)$   
 $(stl\ as)$

**unfolding** *represent-run-def* **by** (*subst sscan.ctr*) (*rule HOL.refl*)

**end**

**context** *Simulation-Graph-Prestable*  
**begin**

**lemma** *represent-run-Run:*

*run* (*represent-run*  $x$   $as$ ) **if**  $Run$  ( $a \#\# as$ )  $x \in a$   
**using** *that*  
**proof** (*coinduction arbitrary: a x as*)  
  **case** (*run a x as*)  
  **obtain**  $b$   $bs$  **where**  $as = b \#\# bs$  **by** (*metis stream.collapse*)  
  **with** *run* **have**  $A$   $a$   $b$   $Run$  ( $b \#\# bs$ ) **by** (*auto elim: Steps.run.cases*)  
  **from** *prestable*[*OF*  $\langle A$   $a$   $b \rangle$ ]  $\langle x \in a \rangle$  **obtain**  $y$  **where**  $C$   $x$   $y \wedge y \in b$  **by**  
*auto*  
  **then** **have**  $C$   $x$  ( $SOME$   $y. C$   $x$   $y \wedge y \in b$ )  $\wedge$  ( $SOME$   $y. C$   $x$   $y \wedge y \in b$ )  $\in$   
 $b$  **by** (*rule someI*)  
  **then** **show** *?case* **using**  $\langle Run$  ( $b \#\# bs$ )  $\rangle$  **unfolding**  $\langle as = - \rangle$   
  **apply** (*subst represent-run-ctr, simp*)  
  **apply** (*subst represent-run-ctr, simp*)  
  **by** (*auto simp: represent-run-ctr[symmetric]*)  
**qed**

**lemma** *represent-run-represent:*

*stream-all2* ( $\in$ ) (*represent-run*  $x$   $as$ ) ( $a \#\# as$ ) **if**  $Run$  ( $a \#\# as$ )  $x \in a$   
**using** *that*  
**proof** (*coinduction arbitrary: a x as*)  
  **case** (*stream-rel x' xs a' as' a x as*)  
  **obtain**  $b$   $bs$  **where**  $as = b \#\# bs$  **by** (*metis stream.collapse*)  
  **with** *stream-rel* **have**  $A$   $a$   $b$   $Run$  ( $b \#\# bs$ ) **by** (*auto elim: Steps.run.cases*)  
  **from** *prestable*[*OF*  $\langle A$   $a$   $b \rangle$ ]  $\langle x \in a \rangle$  **obtain**  $y$  **where**  $C$   $x$   $y \wedge y \in b$  **by**  
*auto*  
  **then** **have**  $C$   $x$  ( $SOME$   $y. C$   $x$   $y \wedge y \in b$ )  $\wedge$  ( $SOME$   $y. C$   $x$   $y \wedge y \in b$ )  $\in$   
 $b$  **by** (*rule someI*)  
  **with**  $\langle x' \#\# xs = - \rangle$   $\langle a' \#\# as' = - \rangle$   $\langle x \in a \rangle$   $\langle Run$  ( $b \#\# bs$ )  $\rangle$  **show**  
*?case* **unfolding**  $\langle as = - \rangle$   
  **by** (*subst (asm) represent-run-ctr*) *auto*  
**qed**

**end**

**context** *Simulation-Graph-Complete-Prestable*  
**begin**

**lemma** *step-bisim:*

$\exists y'. C$   $x' y' \wedge (\exists a. P$   $a \wedge y \in a \wedge y' \in a)$  **if**  $C$   $x$   $y$   $x \in a$   $x' \in a$   $P$   $a$   
**proof** –

**from** *complete*[*OF*  $\langle C \ x \ y \rangle - \langle P \ a \rangle \langle x \in a \rangle$ ] **obtain**  $b'$  **where**  $A \ a \ b' \ y \in b'$   
**by** *auto*  
**from** *prestable*[*OF*  $\langle A \ a \ b' \rangle \langle x' \in a \rangle$ ] **obtain**  $y'$  **where**  $y' \in b' \ C \ x' \ y'$   
**by** *auto*  
**with**  $\langle P \ a \rangle \langle A \ a \ b' \rangle \langle y \in b' \rangle$  **show** *?thesis*  
**by** *auto*  
**qed**

**sublocale** *steps-bisim*:

*Bisimulation-Invariant*  $C \ C \ \lambda \ x \ y. \exists \ a. P \ a \wedge x \in a \wedge y \in a \ \lambda \ -. \ True \ \lambda \ -. \ True$   
**by** (*standard*; *meson step-bisim*)

**lemma** *runs-bisim*:

$\exists \ ys. \text{run} \ (y \ \#\# \ ys) \wedge \text{stream-all2} \ (\lambda \ x \ y. \exists \ a. x \in a \wedge y \in a \wedge P \ a) \ xs$   
 $ys$   
**if**  $\text{run} \ (x \ \#\# \ xs) \ x \in a \ y \in a \ P \ a$   
**using** *that*  
**by**  $- \ (\text{drule} \ \text{steps-bisim.bisim.A-B.simulation-run}[\text{of} \ - \ - \ y],$   
 $\text{auto} \ \text{elim}!: \ \text{stream-all2-weaken} \ \text{simp:} \ \text{steps-bisim.equiv'-def}$   
 $)$

**lemma** *runs-bisim'*:

$\exists \ ys. \text{run} \ (y \ \#\# \ ys) \ \text{if} \ \text{run} \ (x \ \#\# \ xs) \ x \in a \ y \in a \ P \ a$   
**using** *runs-bisim*[*OF that*] **by** *blast*

**context**

**fixes**  $Q :: 'a \Rightarrow \text{bool}$

**assumes** *compatible*:  $Q \ x \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P \ a \Longrightarrow Q \ y$

**begin**

**lemma** *Alw-ev-compatible'*:

**assumes**  $\forall \ xs. \text{run} \ (x \ \#\# \ xs) \longrightarrow \text{ev} \ (\text{holds} \ Q) \ (x \ \#\# \ xs) \ \text{run} \ (y \ \#\# \ xs) \ x \in a \ y \in a \ P \ a$   
**shows**  $\text{ev} \ (\text{holds} \ Q) \ (y \ \#\# \ xs)$

**proof**  $-$

**from** *assms* **obtain**  $ys$  **where**  $\text{run} \ (x \ \#\# \ ys) \ \text{stream-all2} \ \text{steps-bisim.equiv}' \ xs \ ys$

**by** (*auto*  $\&$   $\exists \ \text{simp:} \ \text{steps-bisim.equiv'-def} \ \text{dest:} \ \text{steps-bisim.bisim.A-B.simulation-run}$ )

**with** *assms*(1) **have**  $\text{ev} \ (\text{holds} \ Q) \ (x \ \#\# \ ys)$

**by** *auto*

**from**  $\langle \text{stream-all2} \ - \ - \ \rangle \ \text{assms}$  **have**  $\text{stream-all2} \ \text{steps-bisim.B-A.equiv}' \ (x \ \#\# \ ys) \ (y \ \#\# \ xs)$

```

by (fastforce
  simp: steps-bisim.equiv'-def steps-bisim.A-B.equiv'-def
  intro: steps-bisim.stream-all2-rotate-2
)
then show ?thesis
  by - (rule steps-bisim.ev-ψ-φ[OF - - ⟨ev - -⟩],
    auto dest: compatible simp: steps-bisim.A-B.equiv'-def
  )
qed

```

**lemma** *Alw-ev-compatible:*

*Alw-ev Q x ↔ Alw-ev Q y* **if**  $x \in a \ y \in a \ P \ a$

**unfolding** *Alw-ev-def* **using** *that* **by** (*auto intro: Alw-ev-compatible*)

**end**

**lemma** *steps-bisim:*

$\exists \ ys. \ steps \ (y \ \# \ ys) \ \wedge \ list\text{-}all2 \ (\lambda \ x \ y. \ \exists \ a. \ x \in a \ \wedge \ y \in a \ \wedge \ P \ a) \ xs \ ys$

**if** *steps*  $(x \ \# \ xs) \ x \in a \ y \in a \ P \ a$

**using** *that*

**by** (*auto 4 4*

*dest: steps-bisim.bisim.A-B.simulation-steps*

*intro: list-all2-mono simp: steps-bisim.equiv'-def*

)

**end**

**context** *Subgraph-Node-Defs*

**begin**

**lemma** *subgraph-runD:*

*run xs* **if** *G'.run xs*

**by** (*metis G'.run.cases run.coinduct subgraph that*)

**lemma** *subgraph-V-all:*

*pred-stream V xs* **if** *G'.run xs*

**by** (*metis (no-types, lifting) G'.run.simps Subgraph-Node-Defs.E'-V1 stream.inject stream-pred-coinduct that*)

**lemma** *subgraph-runI:*

*G'.run xs* **if** *pred-stream V xs run xs*

**using** *that*

**by** (*coinduction arbitrary: xs*) (*metis Subgraph-Node-Defs.E'-def run.cases stream.pred-inject*)

**lemma** *subgraph-run-iff*:

$G'.run\ xs \longleftrightarrow pred-stream\ V\ xs \wedge run\ xs$

**using** *subgraph-V-all subgraph-runD subgraph-runI* **by** *blast*

**end**

**context** *Double-Simulation-Finite-Complete-Abstraction-Prop-Bisim*

**begin**

**sublocale** *sim-complete: Simulation-Graph-Complete-Prestable C-φ A1-φ*  
*P1*

**by** (*standard; force dest: P1-invariant φ-A1-compatible A1-complete simp:*  
*C-φ-def A1-φ-def*)

**lemma** *runs-closure-bisim*:

$\exists y\ ys. y \in a_0 \wedge phi.run\ (y\ \#\#\ ys)$  **if**  $phi.run\ (x\ \#\#\ xs) \ x \in \bigcup (phi.closure\ a_0)$

**using** *that(2) sim-complete.runs-bisim'[OF that(1)]* **unfolding** *phi.closure-def*  
**by** *auto*

**lemma** *infinite-run-cycle-iff'*:

$(\exists x_0\ xs. x_0 \in a_0 \wedge phi.run\ (x_0\ \#\#\ xs)) = (\exists as\ a\ bs. phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**by** (*intro phi.infinite-run-cycle-iff' P2-a0 runs-closure-bisim*)

**corollary** *infinite-run-cycle-iff*:

$(\exists x_0\ xs. x_0 \in a_0 \wedge run\ (x_0\ \#\#\ xs) \wedge pred-stream\ \varphi\ (x_0\ \#\#\ xs)) \longleftrightarrow$   
 $(\exists as\ a\ bs. phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**if**  $a_0 \subseteq \{x. \varphi\ x\}$

**unfolding** *infinite-run-cycle-iff'[symmetric] phi-run-iff[symmetric]* **using**  
*that by auto*

**theorem** *Alw-ev-mc*:

$(\forall x_0 \in a_0. Alw-ev\ (Not\ o\ \varphi)\ x_0) \longleftrightarrow \neg (\exists as\ a\ bs. phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a]))$

**if**  $a_0 \subseteq \{x. \varphi\ x\}$

**unfolding** *Alw-ev alw-holds-pred-stream-iff infinite-run-cycle-iff'[OF that, symmetric]*

**by** (*auto simp: comp-def*)

**lemma** *phi-Steps-Alw-ev*:

$\neg (\exists as\ a\ bs. phi.Steps\ (a_0\ \#\ as\ @\ a\ \#\ bs\ @\ [a])) \longleftrightarrow phi.Steps.Alw-ev$

( $\lambda \ -. \ False$ )  $a_0$   
**unfolding**  $phi.Steps.Alw-ev$   
**by** ( $auto \ 4 \ 3 \ dest$ :  
    $sdrop-wait \ phi.Steps-finite.run-finite-state-set-cycle-steps \ phi.Steps-finite.cycle-steps-run$   
    $simp: \ not-alw-iff \ comp-def$   
 )

**theorem**  $Alw-ev-mc'$ :

$(\forall x_0 \in a_0. \ Alw-ev \ (Not \ o \ \varphi) \ x_0) \longleftrightarrow \ phi.Steps.Alw-ev \ (\lambda \ -. \ False) \ a_0$   
**if**  $a_0 \subseteq \{x. \ \varphi \ x\}$   
**unfolding**  $Alw-ev-mc[OF \ that] \ phi-Steps-Alw-ev[symmetric] \ ..$

**end**

**context**  $Graph-Start-Defs$

**begin**

**interpretation**  $Bisimulation-Invariant \ E \ E \ (=) \ reachable \ reachable$

**including**  $graph-automation$  **by**  $standard \ auto$

**lemma**  $Alw-alw-iff-default$ :

$Alw-alw \ \varphi \ x \longleftrightarrow Alw-alw \ \psi \ x$  **if**  $\bigwedge x. \ reachable \ x \implies \varphi \ x \longleftrightarrow \psi \ x$   
 $reachable \ x$   
**by** ( $rule \ Alw-alw-iff-strong$ ) ( $auto \ simp: \ that \ A-B.equiv'-def$ )

**lemma**  $Alw-ev-iff-default$ :

$Alw-ev \ \varphi \ x \longleftrightarrow Alw-ev \ \psi \ x$  **if**  $\bigwedge x. \ reachable \ x \implies \varphi \ x \longleftrightarrow \psi \ x$   
 $reachable \ x$   
**by** ( $rule \ Alw-ev-iff$ ) ( $auto \ simp: \ that \ A-B.equiv'-def$ )

**end**

**context**  $Double-Simulation-Complete-Bisim-Cover$

**begin**

**lemma**  $P2-closure-subs$ :

$a \subseteq \bigcup (closure \ a)$  **if**  $P2 \ a$   
**using**  $P2-P1-cover[OF \ that]$  **unfolding**  $closure-def$  **by**  $fastforce$

**lemma** (**in**  $Double-Simulation-Complete$ )  $P2-Steps-last$ :

$P2 \ (last \ as)$  **if**  $Steps \ as \ a_0 = hd \ as$   
**using**  $that$  **by**  $- \ (cases \ as, \ auto \ dest!; \ P2-invariant-Steps \ simp: \ list-all-iff \ P2-a_0)$

**lemma** (in *Double-Simulation*) *compatible-closure*:

**assumes** *compatible*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**and**  $\forall x \in a. P x$

**shows**  $\forall x \in \bigcup(\text{closure } a). P x$

**unfolding** *closure-def* **using** *assms(2)* **by** (*auto dest: compatible*)

**lemma** *compatible-closure-all-iff*:

**assumes** *compatible*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$  **and** *P2 a*

**shows**  $(\forall x \in a. P x) \longleftrightarrow (\forall x \in \bigcup(\text{closure } a). P x)$

**using**  $\langle P2 a \rangle$  **by** (*auto dest!: P2-closure-subs dest: compatible simp: closure-def*)

**lemma** *compatible-closure-ex-iff*:

**assumes** *compatible*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$  **and** *P2 a*

**shows**  $(\exists x \in a. P x) \longleftrightarrow (\exists x \in \bigcup(\text{closure } a). P x)$

**using**  $\langle P2 a \rangle$  **by** (*auto 4 3 dest!: P2-closure-subs dest: compatible P2-cover simp: closure-def*)

**lemma** (in *Double-Simulation-Complete-Bisim*) *no-deadlock-closureI*:

$\forall x_0 \in \bigcup(\text{closure } a_0). \neg \text{deadlock } x_0$  **if**  $\forall x_0 \in a_0. \neg \text{deadlock } x_0$

**using that by** – (*rule compatible-closure, simp, rule bisim.steps-bisim.deadlock-iff, auto*)

**context**

**fixes** *P*

**assumes** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**begin**

**lemma** *reaches-all-1*:

**fixes** *b :: 'a set and y :: 'a and as :: 'a set list*

**assumes** *A*:  $\forall y. (\exists x_0 \in \bigcup(\text{closure } (\text{hd } as)). \exists xs. \text{hd } xs = x_0 \wedge \text{last } xs = y \wedge \text{steps } xs) \longrightarrow P y$

**and**  $y \in \text{last } as$  **and**  $a_0 = \text{hd } as$  **and** *Steps as*

**shows** *P y*

**proof** –

**from** *assms* **obtain** *bs* **where** [*simp*]:  $as = a_0 \# bs$  **by** (*cases as*) *auto*

**from** *Steps-Union*[*OF*  $\langle \text{Steps } \rightarrow \rangle$ ] **have** *post-defs.Steps* (*map closure as*) .

**from**  $\langle \text{Steps } as \rangle \langle a_0 = \rightarrow \rangle$  **have** *P2* (*last as*)

**by** (*rule P2-Steps-last*)

**obtain** *b2* **where**  $b2: y \in b2 \wedge b2 \in \text{last } (\text{closure } a_0 \# \text{map closure } bs)$

**apply** *atomize-elim*

```

apply simp
apply safe
using  $\langle y \in \rightarrow P2\text{-closure-subst}[OF \langle P2 (last\ as) \rangle]$ 
by  $(auto\ simp: last\ map)$ 
with  $post.Steps\ poststable[OF \langle post\ defs.Steps \rightarrow, of\ b2]$  obtain  $as'$  where
 $as'$ :
   $pre\ defs.Steps\ as'$   $list\ all2 (\in) as'$   $(closure\ a_0 \# map\ closure\ bs)$   $last\ as'$ 
 $= b2$ 
  by  $auto$ 
then obtain  $x_0$  where  $x_0 \in hd\ as'$ 
  by  $(cases\ as')$   $(auto\ split: if\ split\ asm\ simp: closure\ def)$ 
from  $pre.Steps\ prestable[OF \langle pre\ defs.Steps \rightarrow \langle x_0 \in \rightarrow \rangle]$  obtain  $xs$  where
   $steps\ (x_0 \# xs)$   $list\ all2 (\in) (x_0 \# xs)$   $as'$ 
  by  $auto$ 
from  $\langle x_0 \in \rightarrow \langle list\ all2 (\in) as' \rightarrow \rangle$  have  $x_0 \in \bigcup (closure\ a_0)$ 
  by  $(cases\ as')$   $auto$ 
with  $A \langle steps \rightarrow \rangle$  have  $P (last\ (x_0 \# xs))$ 
  by  $fastforce$ 
from  $as'$  have  $P1\ b2$ 
  using  $b2$  by  $(auto\ simp: closure\ def\ last\ map\ split: if\ split\ asm)$ 
from  $\langle list\ all2 (\in) as' \rightarrow \langle list\ all2 (\in) - as' \rangle \langle - = b2 \rangle$  have  $last\ (x_0 \# xs)$ 
 $\in b2$ 
  by  $(fastforce\ dest!: list\ all2\ last)$ 
from  $P1\ P[OF\ this\ \langle y \in b2 \rangle \langle P1\ b2 \rangle]$   $\langle P \rightarrow \rangle$  show  $P\ y ..$ 
qed

```

**lemma reaches-all-2:**

```

fixes  $x_0\ a\ xs$ 
assumes  $A: \forall b\ y. (\exists xs. hd\ xs = a_0 \wedge last\ xs = b \wedge Steps\ xs) \wedge y \in b$ 
 $\rightarrow P\ y$ 
  and  $hd\ xs \in a$  and  $a \in closure\ a_0$  and  $steps\ xs$ 
shows  $P (last\ xs)$ 
proof -
  {
    fix  $y\ x_0\ xs$ 
    assume  $hd\ xs \in a_0$  and  $steps\ xs$ 
    then obtain  $x\ ys$  where  $[simp]: xs = x \# ys\ x \in a_0$  by  $(cases\ xs)$   $auto$ 
    from  $steps\ complete[of\ x\ ys\ a_0]$   $\langle steps\ xs \rangle$   $P2\ a_0$  obtain  $as$  where
       $Steps\ (a_0 \# as)$   $list\ all2 (\in) ys\ as$ 
      by  $auto$ 
    then have  $last\ xs \in last\ (a_0 \# as)$ 
      by  $(fastforce\ dest: list\ all2\ last)$ 
    with  $A \langle Steps \rightarrow \rangle \langle x \in \rightarrow \rangle$  have  $P (last\ xs)$ 
      by  $(force\ split: if\ split\ asm)$ 
  }

```

```

} note * = this
from ⟨a ∈ closure a₀⟩ obtain x where x: x ∈ a x ∈ a₀ P1 a
  by (auto simp: closure-def)
with ⟨hd xs ∈ a⟩ ⟨steps xs⟩ bisim.steps-bisim[of hd xs tl xs a x] obtain
xs' where
  hd xs' = x steps xs' list-all2 (λ x y. ∃ a. x ∈ a ∧ y ∈ a ∧ P1 a) xs xs'
  apply atomize-elim
  apply clarsimp
  subgoal for ys
    by (inst-existentials x # ys; force simp: list-all2-Cons2)
  done
with *[of xs'] x have P (last xs')
  by auto
from ⟨steps xs⟩ ⟨list-all2 - xs xs'⟩ obtain b where last xs ∈ b last xs' ∈
b P1 b
  by atomize-elim (fastforce dest!: list-all2-last)
from P1-P[OF this] ⟨P (last xs')⟩ show P (last xs) ..
qed

```

**lemma reaches-all:**

$(\forall y. (\exists x_0 \in \bigcup (\text{closure } a_0). \text{reaches } x_0 y) \longrightarrow P y) \longleftrightarrow (\forall b y. \text{Steps.reaches } a_0 b \wedge y \in b \longrightarrow P y)$

**unfolding reaches-steps-iff Steps.reaches-steps-iff using reaches-all-1 reaches-all-2**  
**by auto**

**lemma reaches-all':**

$(\forall x_0 \in \bigcup (\text{closure } a_0). \forall y. \text{reaches } x_0 y \longrightarrow P y) = (\forall y. \text{Steps.reaches } a_0 y \longrightarrow (\forall x \in y. P x))$

**using reaches-all by auto**

**lemma reaches-all'':**

$(\forall y. \forall x_0 \in a_0. \text{reaches } x_0 y \longrightarrow P y) \longleftrightarrow (\forall b y. \text{Steps.reaches } a_0 b \wedge y \in b \longrightarrow P y)$

**proof –**

**have**  $(\forall x_0 \in a_0. \forall y. \text{reaches } x_0 y \longrightarrow P y) \longleftrightarrow (\forall x_0 \in \bigcup (\text{closure } a_0). \forall y. \text{reaches } x_0 y \longrightarrow P y)$

**apply** (rule compatible-closure-all-iff[OF - P2-a₀])

**apply safe**

**subgoal for**  $a x y y'$

**by** (blast dest: P1-P bisim.steps-bisim.A-B.simulation-reaches[of - - x])

**subgoal for**  $a x y y'$

**by** (blast dest: P1-P bisim.steps-bisim.A-B.simulation-reaches[of - - y])

**done**

**from** this[unfolded reaches-all'] **show** ?thesis

by *auto*  
qed

**lemma** *reaches-ex*:

$(\exists y. \exists x_0 \in \bigcup (\text{closure } a_0). \text{reaches } x_0 \ y \wedge P \ y) = (\exists b \ y. \text{Steps.reaches } a_0 \ b \wedge y \in b \wedge P \ y)$

**proof** (*safe, goal-cases*)

case (1 *y x<sub>0</sub> X*)

then obtain *x* where  $x \in X \ x \in a_0 \ P1 \ X$

unfolding *closure-def* by *auto*

with  $\langle x_0 \in - \rangle \langle \text{reaches } - \ - \rangle$  obtain *y'* *Y* where  $\text{reaches } x \ y' \ P1 \ Y \ y' \in Y$   
*y*  $\in Y$

by (*auto dest: bisim.steps-bisim.A-B.simulation-reaches[of - - x]*)

with *simulation.simulation-reaches[OF  $\langle \text{reaches } x \ y' \rangle \langle x \in a_0 \rangle - P2-a_0$ ]*  
 $\langle P \ - \rangle$  show ?*case*

by (*auto dest: P1-P*)

next

case (2 *b y*)

with  $\langle y \in b \rangle$  obtain *Y* where  $y \in Y \ Y \in \text{closure } b \ P1 \ Y$

unfolding *closure-def*

by (*metis (mono-tags, lifting) P2-P1-cover P2-invariant.invariant-reaches mem-Collect-eq*)

from *closure-reaches[OF  $\langle \text{Steps.reaches } - \ - \rangle$ ]* have

*post-defs.Steps.reaches* (*closure a<sub>0</sub>*) (*closure b*)

by *auto*

from *post.reaches-poststable[OF this  $\langle Y \in - \rangle$ ]* obtain *X* where

$X \in \text{closure } a_0 \ \text{pre-defs.Steps.reaches } X \ Y$

by *auto*

then obtain *x* where  $x \in X \ x \in a_0$

unfolding *closure-def* by *auto*

from *pre.reaches-prestable[OF  $\langle \text{pre-defs.Steps.reaches } X \ Y \rangle \langle x \in X \rangle$ ]* obtain *y'* where

$\text{reaches } x \ y' \ y' \in Y$

by *auto*

with  $\langle x \in X \rangle \langle X \in - \rangle \langle P \ y \rangle \langle P1 \ Y \rangle \langle y \in Y \rangle$  show ?*case*

by (*auto dest: P1-P*)

qed

**lemma** *reaches-ex'*:

$(\exists y. \exists x_0 \in a_0. \text{reaches } x_0 \ y \wedge P \ y) \longleftrightarrow (\exists b \ y. \text{Steps.reaches } a_0 \ b \wedge y \in b \wedge P \ y)$

**proof** –

have  $(\exists x_0 \in a_0. \exists y. \text{reaches } x_0 \ y \wedge P \ y) \longleftrightarrow (\exists x_0 \in \bigcup (\text{closure } a_0). \exists y.$

```

reaches  $x_0 y \wedge P y$ )
  apply (rule compatible-closure-ex-iff[OF - P2-a0])
  apply safe
  subgoal for  $a x y y'$ 
    by (blast dest: P1-P bisim.steps-bisim.A-B.simulation-reaches[of - - y])
  subgoal for  $a x y y'$ 
    by (blast dest: P1-P bisim.steps-bisim.A-B.simulation-reaches[of - - x])
  done
from this reaches-ex show ?thesis
  by auto
qed

```

end

```

lemma (in Double-Simulation-Complete-Bisim) P1-deadlocked-compatible:
  deadlocked  $x = deadlocked y$  if  $x \in a y \in a P1 a$  for  $x y a$ 
  unfolding deadlocked-def using that apply auto
  subgoal
    using A1-complete prestable by blast
  subgoal using A1-complete prestable by blast
  done

```

lemma steps-Steps-no-deadlock:

```

 $\neg Steps.deadlock a_0$ 
if no-deadlock:  $\forall x_0 \in \bigcup (closure a_0). \neg deadlock x_0$ 
proof -
  from P1-deadlocked-compatible have
    ( $\forall y. (\exists x_0 \in \bigcup (closure a_0). reaches x_0 y) \longrightarrow (Not \circ deadlocked) y$ ) =
    ( $\forall b y. Steps.reaches a_0 b \wedge y \in b \longrightarrow (Not \circ deadlocked) y$ )
  using reaches-all[of Not o deadlocked] unfolding comp-def by blast
  then show  $\neg Steps.deadlock a_0$ 
  using no-deadlock
  unfolding Steps.deadlock-def deadlock-def
  apply safe
  subgoal
    by (simp add: Graph-Defs.deadlocked-def)
    (metis P2-cover P2-invariant.invariant-reaches disjoint-iff-not-equal
simulation.A-B-step)
  subgoal
    by auto
  done
qed

```

lemma steps-Steps-no-deadlock1:

$\neg \text{Steps.deadlock } a_0$   
**if** *no-deadlock*:  $\forall x_0 \in a_0. \neg \text{deadlock } x_0$  **and** *closure-simp*:  $\bigcup(\text{closure } a_0)$   
 $= a_0$   
**using** *steps-Steps-no-deadlock*[*unfolded closure-simp, OF no-deadlock*] .

**lemma** *Alw-alw-iff*:

$(\forall x_0 \in \bigcup(\text{closure } a_0). \text{Alw-alw } P x_0) \longleftrightarrow \text{Steps.Alw-alw } (\lambda a. \forall c \in a. P c) a_0$   
**if** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$   
**and** *no-deadlock*:  $\forall x_0 \in \bigcup(\text{closure } a_0). \neg \text{deadlock } x_0$

**proof** –

**from** *steps-Steps-no-deadlock*[*OF no-deadlock*] **show** *?thesis*  
**by** (*simp add: Alw-alw-iff Steps.Alw-alw-iff no-deadlock Steps.Ex-ev Ex-ev*)  
*(rule reaches-all'[simplified]; erule P1-P; assumption)*

**qed**

**lemma** *Alw-alw-iff1*:

$(\forall x_0 \in a_0. \text{Alw-alw } P x_0) \longleftrightarrow \text{Steps.Alw-alw } (\lambda a. \forall c \in a. P c) a_0$   
**if** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$   
**and** *no-deadlock*:  $\forall x_0 \in a_0. \neg \text{deadlock } x_0$  **and** *closure-simp*:  $\bigcup(\text{closure } a_0) = a_0$   
**using** *Alw-alw-iff*[*OF P1-P*] *no-deadlock unfolding closure-simp by auto*

**lemma** *Alw-alw-iff2*:

$(\forall x_0 \in a_0. \text{Alw-alw } P x_0) \longleftrightarrow \text{Steps.Alw-alw } (\lambda a. \forall c \in a. P c) a_0$   
**if** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$   
**and** *no-deadlock*:  $\forall x_0 \in a_0. \neg \text{deadlock } x_0$

**proof** –

**have**  $(\forall x_0 \in a_0. \text{Alw-alw } P x_0) \longleftrightarrow (\forall x_0 \in \bigcup(\text{closure } a_0). \text{Alw-alw } P x_0)$

**apply** –

**apply** (*rule compatible-closure-all-iff, rule bisim.steps-bisim.Alw-alw-iff-strong*)

**unfolding** *bisim.steps-bisim.A-B.equiv'-def*

**by** (*blast intro: P2-a<sub>0</sub> dest: P1-P*)**+**

**also have**  $\dots \longleftrightarrow \text{Steps.Alw-alw } (\lambda a. \forall c \in a. P c) a_0$

**by** (*rule Alw-alw-iff*[*OF P1-P no-deadlock-closureI*][*OF no-deadlock*])

**finally show** *?thesis* .

**qed**

**lemma** *Steps-all-Alw-ev*:

$\forall x_0 \in a_0. \text{Alw-ev } P x_0$  **if** *Steps.Alw-ev*  $(\lambda a. \forall c \in a. P c) a_0$

**using** *that unfolding Alw-ev-def Steps.Alw-ev-def*

**apply** *safe*

**apply** (*drule run-complete*[*OF - - P2-a<sub>0</sub>*], *assumption*)

**apply** *safe*  
**apply** (*elim allE impE, assumption*)  
**subgoal premises** *prems* **for** *x xs* **as**  
**using** *prems(4,3,1)*  
**by** (*induction a<sub>0</sub> ## as arbitrary: a<sub>0</sub> as x xs rule: ev.induct*)  
*(auto 4 3 elim: stream.rel-cases intro: ev-Stream)*  
**done**

**lemma** *closure-compatible-Steps-all-ex-iff:*

*Steps.Alw-ev* ( $\lambda a. \forall c \in a. P c$ )  $a_0 \longleftrightarrow$  *Steps.Alw-ev* ( $\lambda a. \exists c \in a. P c$ )  $a_0$

**if** *closure-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P2 a \implies P x \longleftrightarrow P y$

**proof** –

**interpret** *Bisimulation-Invariant A2 A2* (=) *P2 P2*

**by** *standard auto*

**show** *?thesis*

**using** *P2-a<sub>0</sub>*

**by** – (*rule Alw-ev-iff, unfold A-B.equiv'-def; meson P2-cover closure-P disjoint-iff-not-equal*)

**qed**

**lemma** (*in* –) *compatible-imp:*

**assumes**  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**and**  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies Q x \longleftrightarrow Q y$

**shows**  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies (Q x \longrightarrow P x) \longleftrightarrow (Q y \longrightarrow P y)$

**using** *assms by metis*

**lemma** *Leadsto-iff:*

$(\forall x_0 \in \bigcup(\text{closure } a_0). \text{leadsto } P Q x_0) \longleftrightarrow$  *Steps.Alw-alw* ( $\lambda a. \forall c \in a. P c \longrightarrow \text{Alw-ev } Q c$ )  $a_0$

**if** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**and** *P1-Q*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies Q x \longleftrightarrow Q y$

**and** *no-deadlock*:  $\forall x_0 \in \bigcup(\text{closure } a_0). \neg \text{deadlock } x_0$

**unfolding** *leadsto-def*

**by** (*subst Alw-alw-iff[OF - no-deadlock],*

*intro compatible-imp bisim.Alw-ev-compatible,*

*(subst (asm) P1-Q; force), (assumption | intro HOL.refl P1-P)+*

)

**lemma** *Leadsto-iff1:*

$(\forall x_0 \in a_0. \text{leadsto } P Q x_0) \longleftrightarrow$  *Steps.Alw-alw* ( $\lambda a. \forall c \in a. P c \longrightarrow \text{Alw-ev } Q c$ )  $a_0$

**if** *P1-P*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**and**  $P1-Q$ :  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies Q x \longleftrightarrow Q y$   
**and**  $no\_deadlock$ :  $\forall x_0 \in a_0. \neg deadlock x_0$  **and**  $closure\_simp$ :  $\bigcup(closure a_0) = a_0$   
**by** ( $subst\ closure\_simp[symmetric]$ ,  $rule\ Leadsto\_iff$ )  
 ( $auto\ simp$ :  $closure\_simp\ no\_deadlock\ dest$ :  $P1-Q\ P1-P$ )

**lemma**  $Leadsto\_iff2$ :

$(\forall x_0 \in a_0. leadsto\ P\ Q\ x_0) \longleftrightarrow Steps.Alw\_alw\ (\lambda a. \forall c \in a. P\ c \longrightarrow Alw\_ev\ Q\ c)\ a_0$   
**if**  $P1-P$ :  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$   
**and**  $P1-Q$ :  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies Q x \longleftrightarrow Q y$   
**and**  $no\_deadlock$ :  $\forall x_0 \in a_0. \neg deadlock x_0$

**proof** –

**have**  $(\forall x_0 \in a_0. leadsto\ P\ Q\ x_0) \longleftrightarrow (\forall x_0 \in \bigcup(closure\ a_0). leadsto\ P\ Q\ x_0)$

**apply** –

**apply** ( $rule\ compatible\_closure\_all\_iff$ ,  $rule\ bisim.steps\_bisim.Leadsto\_iff$ )

**unfolding**  $bisim.steps\_bisim.A-B.equiv'\_def$  **by** ( $blast\ intro$ :  $P2-a_0\ dest$ :  $P1-P\ P1-Q$ ) $+$

**also have**  $\dots \longleftrightarrow Steps.Alw\_alw\ (\lambda a. \forall c \in a. P\ c \longrightarrow Alw\_ev\ Q\ c)\ a_0$

**by** ( $rule\ Leadsto\_iff[OF\ -\ -\ no\_deadlock\_closureI[OF\ no\_deadlock]]$ ;  $rule\ P1-P\ P1-Q$ )

**finally show**  $?thesis$  .

**qed**

**lemma** (**in**  $-$ )  $compatible\_convert1$ :

**assumes**  $\bigwedge x y a. P x \implies x \in a \implies y \in a \implies P1 a \implies P y$

**shows**  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**by** ( $auto\ intro$ :  $assms$ )

**lemma** (**in**  $-$ )  $compatible\_convert2$ :

**assumes**  $\bigwedge a x y. x \in a \implies y \in a \implies P1 a \implies P x \longleftrightarrow P y$

**shows**  $\bigwedge x y a. P x \implies x \in a \implies y \in a \implies P1 a \implies P y$

**using**  $assms$  **by**  $meson$

**lemma** (**in**  $Double-Simulation-Defs$ )

**assumes**  $compatible$ :  $\bigwedge x y a. P x \implies x \in a \implies y \in a \implies P1 a \implies P y$

**and that**:  $\forall x \in a. P x$

**shows**  $\forall x \in \bigcup(closure\ a). P x$

**using that** **unfolding**  $closure\_def$  **by** ( $auto\ dest$ :  $compatible$ )

**end**

**context** *Double-Simulation-Finite-Complete-Bisim-Cover*  
**begin**

**lemma** *Alw-ev-Steps-ex:*

$(\forall x_0 \in \bigcup(\text{closure } a_0). \text{Alw-ev } P x_0) \longrightarrow \text{Steps.Alw-ev } (\lambda a. \exists c \in a. P c) a_0$

**if** *closure-P*:  $\bigwedge a x y. x \in \bigcup(\text{closure } a) \implies y \in \bigcup(\text{closure } a) \implies P2 a \implies P x \longleftrightarrow P y$

**unfolding** *Alw-ev Steps.Alw-ev*

**apply** *safe*

**apply** (*frule Steps-finite.run-finite-state-set-cycle-steps*)

**apply** *clarify*

**apply** (*frule Steps-run-cycle'*)

**apply** (*auto dest!: P2-invariant.invariant-run simp: stream.pred-set; fail*)

**unfolding** *that*

**apply** *clarify*

**subgoal** **premises** *prems* **for** *xs x ys zs x' xs' R*

**proof** –

**from**  $\langle x' \in R \rangle \langle R \in - \rangle$  **that** **have**  $\langle x' \in \bigcup(\text{closure } a_0) \rangle$

**by** *auto*

**with** *prems(5,9)* **have**

$\forall c \in \{x'\} \cup \text{sset } xs'. \exists y \in \{a_0\} \cup \text{sset } xs. c \in \bigcup(\text{closure } y)$

**by** *fast*

**with** *prems(3)* **have** \*:

$\forall c \in \{x'\} \cup \text{sset } xs'. \exists y \in \{a_0\} \cup \text{sset } xs. c \in \bigcup(\text{closure } y) \wedge (\forall c \in y. \neg P c)$

**unfolding** *alw-holds-sset* **by** *simp*

**from**  $\langle \text{Run } - \rangle$  **have** \*\*:  $P2 y$  **if**  $y \in \{a_0\} \cup \text{sset } xs$  **for**  $y$

**using** *that* **by** (*auto dest!: P2-invariant.invariant-run simp: stream.pred-set*)

**have** \*\*:  $\neg P c$  **if**  $c \in \bigcup(\text{closure } y) \forall d \in y. \neg P d$   $P2 y$  **for**  $c y$

**proof** –

**from** *that*  $P2\text{-cover}[OF \langle P2 y \rangle]$  **obtain**  $d$  **where**  $d \in y d \in \bigcup(\text{closure } y)$

**by** (*fastforce dest!: P2-closure-subs*)

**with** *that closure-P* **show** *?thesis*

**by** *blast*

**qed**

**from** \* **have**  $\forall c \in \{x'\} \cup \text{sset } xs'. \neg P c$

**by** (*fastforce intro: \*\* dest!: \*\*[rotated]*)

**with** *prems(1)*  $\langle \text{run } - \rangle \langle x' \in \bigcup(\text{closure } -) \rangle$  **show** *?thesis*

**unfolding** *alw-holds-sset* **by** *auto*

**qed**

**done**

**lemma** *Alw-ev-Steps-ex2*:

$(\forall x_0 \in a_0. \text{Alw-ev } P x_0) \longrightarrow \text{Steps.Alw-ev } (\lambda a. \exists c \in a. P c) a_0$   
**if** *closure-P*:  $\bigwedge a x y. x \in \bigcup(\text{closure } a) \Longrightarrow y \in \bigcup(\text{closure } a) \Longrightarrow P2 a$   
 $\Longrightarrow P x \longleftrightarrow P y$   
**and** *P1-P*:  $\bigwedge a x y. x \in a \Longrightarrow y \in a \Longrightarrow P1 a \Longrightarrow P x \longleftrightarrow P y$

**proof** –

**have**  $(\forall x_0 \in a_0. \text{Alw-ev } P x_0) \longleftrightarrow (\forall x_0 \in \bigcup(\text{closure } a_0). \text{Alw-ev } P x_0)$   
**by** (*intro compatible-closure-all-iff bisim.Alw-ev-compatible; auto dest: P1-P simp: P2-a0*)  
**also have**  $\dots \longrightarrow \text{Steps.Alw-ev } (\lambda a. \exists c \in a. P c) a_0$   
**by** (*intro Alw-ev-Steps-ex that*)  
**finally show** *?thesis* .

**qed**

**lemma** *Alw-ev-Steps-ex1*:

$(\forall x_0 \in a_0. \text{Alw-ev } P x_0) \longrightarrow \text{Steps.Alw-ev } (\lambda a. \exists c \in a. P c) a_0$  **if**  
 $\bigcup(\text{closure } a_0) = a_0$   
**and** *closure-P*:  $\bigwedge a x y. x \in \bigcup(\text{closure } a) \Longrightarrow y \in \bigcup(\text{closure } a) \Longrightarrow P2 a$   
 $a \Longrightarrow P x \longleftrightarrow P y$   
**by** (*subst that(1)[symmetric]*) (*intro Alw-ev-Steps-ex closure-P; assumption*)

**lemma** *closure-compatible-Alw-ev-Steps-iff*:

$(\forall x_0 \in a_0. \text{Alw-ev } P x_0) \longleftrightarrow \text{Steps.Alw-ev } (\lambda a. \forall c \in a. P c) a_0$   
**if** *closure-P*:  $\bigwedge a x y. x \in \bigcup(\text{closure } a) \Longrightarrow y \in \bigcup(\text{closure } a) \Longrightarrow P2 a$   
 $\Longrightarrow P x \longleftrightarrow P y$   
**and** *P1-P*:  $\bigwedge a x y. x \in a \Longrightarrow y \in a \Longrightarrow P1 a \Longrightarrow P x \longleftrightarrow P y$   
**apply** *standard*  
**subgoal**  
**apply** (*subst closure-compatible-Steps-all-ex-iff[OF closure-P]*)  
**prefer** 4  
**apply** (*rule Alw-ev-Steps-ex2[OF that, rule-format]*)  
**by** (*auto dest!: P2-closure-subs*)  
**by** (*rule Steps-all-Alw-ev*) (*auto dest: P2-closure-subs*)

**lemma** *Leadsto-iff'*:

$(\forall x_0 \in a_0. \text{leadsto } P Q x_0)$   
 $\longleftrightarrow \text{Steps.Alw-alw } (\lambda a. (\forall c \in a. P c) \longrightarrow \text{Steps.Alw-ev } (\lambda a. \forall c \in a. Q c) a) a_0$   
**if** *P1-P*:  $\bigwedge a x y. x \in a \Longrightarrow y \in a \Longrightarrow P1 a \Longrightarrow P x \longleftrightarrow P y$   
**and** *P1-Q*:  $\bigwedge a x y. x \in a \Longrightarrow y \in a \Longrightarrow P1 a \Longrightarrow Q x \longleftrightarrow Q y$   
**and** *closure-Q*:  $\bigwedge a x y. x \in \bigcup(\text{closure } a) \Longrightarrow y \in \bigcup(\text{closure } a) \Longrightarrow P2 a$   
 $\Longrightarrow Q x \longleftrightarrow Q y$   
**and** *closure-P*:  $\bigwedge a x y. x \in a \Longrightarrow y \in a \Longrightarrow P2 a \Longrightarrow P x \longleftrightarrow P y$

```

and no-deadlock:  $\forall x_0 \in a_0. \neg \text{deadlock } x_0$  and closure-simp:  $\bigcup(\text{closure } a_0) = a_0$ 
apply (subst Leadsto-iff1, (rule that; assumption)+)
subgoal
  apply (rule P2-invariant.Alw-alw-iff-default)
  subgoal premises prems for a
  proof –
    have P2 a
    by (rule P2-invariant.invariant-reaches[OF prems[unfolded Graph-Start-Defs.reachable-def]])
    interpret a: Double-Simulation-Finite-Complete-Bisim-Cover C A1
P1 A2 P2 a
    apply standard
      apply (rule complete; assumption; fail)
      apply (rule P2-invariant; assumption)
    subgoal
      by (fact  $\langle P2 a \rangle$ )
    subgoal
      proof –
        have  $\{b. \text{Steps.reaches } a \ b\} \subseteq \{b. \text{Steps.reaches } a_0 \ b\}$ 
        by (blast intro: rtranclp-trans prems[unfolded Graph-Start-Defs.reachable-def])
        with finite-abstract-reachable show ?thesis
          by – (rule finite-subset)
      qed

      apply (rule A1-complete; assumption)
      apply (rule P1-invariant; assumption)
      apply (rule P2-P1-cover; assumption)
      done
    from  $\langle P2 a \rangle$  show ?thesis
      by – (subst a.closure-compatible-Alw-ev-Steps-iff[symmetric], (rule that; assumption)+,
        auto dest: closure-P intro: that
      )
    qed
  ..
done

context
  fixes P :: 'a  $\Rightarrow$  bool — The property we want to check
  assumes closure-P:  $\bigwedge a \ x \ y. x \in \bigcup(\text{closure } a) \Longrightarrow y \in \bigcup(\text{closure } a) \Longrightarrow P2 \ a \Longrightarrow P \ x \longleftrightarrow P \ y$ 
  and P1-P:  $\bigwedge a \ x \ y. P \ x \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P1 \ a \Longrightarrow P \ y$ 
begin

```

**lemma** *run-alw-ev-bisim*:

$run (x \#\# xs) \implies x \in \bigcup (closure\ a_0) \implies alw (ev (holds\ P))\ xs$   
 $\implies \exists y\ ys. y \in a_0 \wedge run (y \#\# ys) \wedge alw (ev (holds\ P))\ ys$   
**unfolding** *closure-def*  
**apply** *safe*  
**apply** (*rotate-tac 3*)  
**apply** (*drule bisim.runs-bisim, assumption+*)  
**apply** (*auto elim: P1-P dest: alw-ev-lockstep[of P - - - P]*)  
**done**

**lemma**  *$\varphi$ -closure-compatible*:

$P2\ a \implies x \in \bigcup (closure\ a) \implies P\ x \longleftrightarrow (\forall x \in \bigcup (closure\ a). P\ x)$   
**using** *closure-P* **by** *blast*

**theorem** *infinite-buechi-run-cycle-iff*:

$(\exists x_0\ xs. x_0 \in \bigcup (closure\ a_0) \wedge run (x_0 \#\# xs) \wedge alw (ev (holds\ P)) (x_0 \#\# xs))$   
 $\longleftrightarrow (\exists as\ a\ bs. a \neq \{\}\ \wedge post-defs.Steps (closure\ a_0 \# as @ a \# bs @ [a]) \wedge (\forall x \in \bigcup a. P\ x))$   
**by** (*rule*  
*infinite-buechi-run-cycle-iff-closure[OF*  
 *$\varphi$ -closure-compatible run-alw-ev-bisim P2-closure-subs*  
 $\ ]$   
 $\ )$   
**end**

**end**

Possible Solution

**context** *Graph-Invariant*

**begin**

**definition** *E-inv*  $x\ y \equiv E\ x\ y \wedge P\ x \wedge P\ y$

**lemma** *bisim-E-inv*:

*Bisimulation-Invariant E E-inv (=) P P*  
**by** *standard (auto intro: invariant simp: E-inv-def)*

**interpretation** *G-inv*: *Graph-Defs E-inv* .

**lemma** *steps-G-inv-steps*:

$steps (x \# xs) \longleftrightarrow G-inv.steps (x \# xs)$  **if**  $P\ x$

**proof** –

**interpret** *Bisimulation-Invariant E E-inv (=) P P*

**by** (*rule bisim-E-inv*)  
**from**  $\langle P \ x \rangle$  **show** *?thesis*  
**by** (*auto 4 3 simp: equiv'-def list.rel-eq*  
*dest: bisim.A-B.simulation-steps bisim.B-A.simulation-steps*  
*list-all2-mono[of - - - (=)]*  
)

**qed**

**end**

*R-of/from-R* **definition** *R-of lR = snd ' lR*

**definition** *from-R l R = {(l, u) | u. u ∈ R}*

**lemma** *from-R-fst:*  
 $\forall x \in \text{from-R } l \ R. \text{fst } x = l$   
**unfolding** *from-R-def* **by** *auto*

**lemma** *R-of-from-R [simp]:*  
 $R\text{-of } (\text{from-R } l \ R) = R$   
**unfolding** *R-of-def from-R-def image-def* **by** *auto*

**lemma** *from-R-loc:*  
 $l' = l$  **if**  $(l', u) \in \text{from-R } l \ Z$   
**using** *that* **unfolding** *from-R-def* **by** *auto*

**lemma** *from-R-val:*  
 $u \in Z$  **if**  $(l', u) \in \text{from-R } l \ Z$   
**using** *that* **unfolding** *from-R-def* **by** *auto*

**lemma** *from-R-R-of:*  
 $\text{from-R } l \ (R\text{-of } S) = S$  **if**  $\forall x \in S. \text{fst } x = l$   
**using** *that* **unfolding** *from-R-def R-of-def* **by** *force*

**lemma** *R-ofI[intro]:*  
 $Z \in R\text{-of } S$  **if**  $(l, Z) \in S$   
**using** *that* **unfolding** *R-of-def* **by** *force*

**lemma** *from-R-I[intro]:*  
 $(l', u') \in \text{from-R } l' \ Z'$  **if**  $u' \in Z'$   
**using** *that* **unfolding** *from-R-def* **by** *auto*

**lemma** *R-of-non-emptyD:*

$a \neq \{\}$  **if**  $R\text{-of } a \neq \{\}$   
**using that unfolding**  $R\text{-of-def}$  **by**  $\text{simp}$

**lemma**  $R\text{-of-empty}[\text{simp}]$ :  
 $R\text{-of } \{\} = \{\}$   
**using**  $R\text{-of-non-emptyD}$  **by**  $\text{metis}$

**lemma**  $\text{fst-simp}$ :  
 $x = l$  **if**  $\forall x \in a. \text{fst } x = l \ (x, y) \in a$   
**using that by**  $\text{auto}$

**lemma**  $\text{from-R-D}$ :  
 $u \in Z$  **if**  $(l', u) \in \text{from-R } l \ Z$   
**using that unfolding**  $\text{from-R-def}$  **by**  $\text{auto}$

**locale**  $\text{Double-Simulation-paired-Defs} =$   
**fixes**  $C :: ('a \times 'b) \Rightarrow ('a \times 'b) \Rightarrow \text{bool}$  — Concrete step relation  
**and**  $A1 :: ('a \times 'b \text{ set}) \Rightarrow ('a \times 'b \text{ set}) \Rightarrow \text{bool}$   
— Step relation for the first abstraction layer  
**and**  $P1 :: ('a \times 'b \text{ set}) \Rightarrow \text{bool}$  — Valid states of the first abstraction  
layer  
**and**  $A2 :: ('a \times 'b \text{ set}) \Rightarrow ('a \times 'b \text{ set}) \Rightarrow \text{bool}$   
— Step relation for the second abstraction layer  
**and**  $P2 :: ('a \times 'b \text{ set}) \Rightarrow \text{bool}$  — Valid states of the second abstraction  
layer  
**begin**

**definition**  
 $A1' = (\lambda \text{LR } \text{LR}'. \exists l \ l'. (\forall x \in \text{LR}. \text{fst } x = l) \wedge (\forall x \in \text{LR}'. \text{fst } x = l') \wedge P1 \ (l, R\text{-of } \text{LR}) \wedge A1 \ (l, R\text{-of } \text{LR}) \ (l', R\text{-of } \text{LR}')$   
 $)$

**definition**  
 $A2' = (\lambda \text{LR } \text{LR}'. \exists l \ l'. (\forall x \in \text{LR}. \text{fst } x = l) \wedge (\forall x \in \text{LR}'. \text{fst } x = l') \wedge P2 \ (l, R\text{-of } \text{LR}) \wedge A2 \ (l, R\text{-of } \text{LR}) \ (l', R\text{-of } \text{LR}')$   
 $)$

**definition**  
 $P1' = (\lambda \text{LR}. \exists l. (\forall x \in \text{LR}. \text{fst } x = l) \wedge P1 \ (l, R\text{-of } \text{LR}))$

**definition**  
 $P2' = (\lambda \text{LR}. \exists l. (\forall x \in \text{LR}. \text{fst } x = l) \wedge P2 \ (l, R\text{-of } \text{LR}))$

**definition**  $\text{closure}' \ l \ a = \{x. P1 \ (l, x) \wedge a \cap x \neq \{\}\}$

**sublocale** *sim*: *Double-Simulation-Defs*  $C$   $A1'$   $P1'$   $A2'$   $P2'$  .

**end**

**locale** *Double-Simulation-paired* = *Double-Simulation-paired-Defs* +  
**assumes** *prestable*:  $P1 (l, S) \implies A1 (l, S) (l', T) \implies \forall s \in S. \exists s' \in T. C (l, s) (l', s')$

**and** *closure-poststable*:

$s' \in \text{closure}' l' y \implies P2 (l, x) \implies A2 (l, x) (l', y)$

$\implies \exists s \in \text{closure}' l x. A1 (l, s) (l', s')$

**and** *P1-distinct*:  $P1 (l, x) \implies P1 (l, y) \implies x \neq y \implies x \cap y = \{\}$

**and** *P1-finite*: *finite*  $\{(l, x). P1 (l, x)\}$

**and** *P2-cover*:  $P2 (l, a) \implies \exists x. P1 (l, x) \wedge x \cap a \neq \{\}$

**begin**

**sublocale** *sim*: *Double-Simulation*  $C$   $A1'$   $P1'$   $A2'$   $P2'$

**proof** (*standard, goal-cases*)

**case** ( $1$   $S$   $T$ )

**then show** *?case*

**unfolding** *A1'-def* **by** (*metis from-R-I from-R-R-of from-R-val prestable prod.collapse*)

**next**

**case** ( $2$   $s'$   $y$   $x$ )

**then show** *?case*

**unfolding** *A2'-def A1'-def sim.closure-def*

**unfolding** *P1'-def*

**apply** *clarify*

**subgoal premises** *prems* **for**  $l$   $l1$   $l2$

**proof** –

**from** *prems* **have**  $l2 = l1$

**by** *force*

**from** *prems* **have** *R-of*  $s' \in \text{closure}' l1$  (*R-of*  $y$ )

**unfolding** *closure'-def* **by** *auto*

**with**  $\langle A2 - \rightarrow \rangle \langle P2 - \rightarrow \rangle$  *closure-poststable*[*of R-of s' l1 R-of y l R-of x*]

**obtain**  $s$  **where**

$s \in \text{closure}' l$  (*R-of*  $x$ )  $A1 (l, s) (l1, \text{R-of } s')$

**by** *auto*

**with** *prems from-R-fst R-of-from-R* **show** *?thesis*

**apply** –

**unfolding**  $\langle l2 = l1 \rangle$

**apply** (*rule bexI*[**where**  $x = \text{from-R } l s$ ])

**apply** (*inst-existentials l l1*)

**apply** (*simp add: from-R-fst; fail*)+

```

    subgoal
      unfolding closure'-def by auto
      apply (simp; fail)
      unfolding closure'-def
      apply (intro CollectI conjI exI)
      apply fastforce
      apply fastforce
      apply (fastforce simp: R-of-def from-R-def)
    done
  qed
done
next
  case (3 x y)
  then show ?case
    unfolding P1'-def using P1-distinct
    by (smt disjoint-iff-not-equal eq-fst-iff from-R-R-of from-R-val)
next
  case 4
  have {x. ∃ l. (∀ x∈x. fst x = l) ∧ P1 (l, R-of x)} ⊆ (λ (l, x). from-R l x)
  ' {(l, x). P1 (l, x)}
    using from-R-R-of image-iff by fastforce
  with P1-finite show ?case
    unfolding P1'-def by (auto elim: finite-subset)
next
  case (5 a)
  then show ?case
    unfolding P1'-def P2'-def
    apply clarify
    apply (frule P2-cover)
    apply clarify
    subgoal for l x
      apply (inst-existentials from-R l x l, (simp add: from-R-fst)+)
      using R-of-def by (fastforce simp: from-R-fst)
    done
  qed
context
  assumes P2-invariant: P2 a ⇒ A2 a a' ⇒ P2 a'
begin

lemma A2-A2'-bisim: Bisimulation-Invariant A2 A2' (λ (l, Z) b. b =
from-R l Z) P2 P2'
  apply standard

```

```

subgoal  $A2$ - $A2'$  for  $a$   $b$   $a'$ 
  unfolding  $P2'$ -def
  apply clarify
  apply (inst-existentials from-R (fst b) (snd b))
  subgoal for  $x$   $y$   $l$ 
    unfolding  $A2'$ -def
    apply simp
    apply (inst-existentials l)
    by (auto dest!: P2-cover simp: from-R-def)
    by clarsimp
  subgoal  $A2'$ - $A2$  for  $a$   $a'$   $b'$ 
    using from-R-fst by (fastforce dest: sim.P2-cover simp: from-R-R-of
 $A2'$ -def)
    subgoal  $P2$ -invariant for  $a$   $b$ 
      by (fact P2-invariant)
    subgoal  $P2'$ -invariant for  $a$   $b$ 
      unfolding  $P2'$ -def  $A2'$ -def using  $P2$ -invariant by blast
    done

end

end

locale Double-Simulation-Complete-paired = Double-Simulation-paired +
  fixes  $l_0$   $a_0$ 
  assumes complete:  $C$  ( $l$ ,  $x$ ) ( $l'$ ,  $y$ )  $\implies x \in S \implies P2$  ( $l$ ,  $S$ )  $\implies \exists T$ .  $A2$ 
( $l$ ,  $S$ ) ( $l'$ ,  $T$ )  $\wedge y \in T$ 
  assumes  $P2$ -invariant:  $P2$   $a \implies A2$   $a$   $a' \implies P2$   $a'$ 
    and  $P2$ - $a_0'$ :  $P2$  ( $l_0$ ,  $a_0$ )
begin

interpretation Bisimulation-Invariant  $A2$   $A2'$   $\lambda$  ( $l$ ,  $Z$ )  $b$ .  $b =$  from-R l Z
 $P2$   $P2'$ 
  by (rule A2-A2'-bisim[OF P2-invariant])

sublocale Double-Simulation-Complete  $C$   $A1'$   $P1'$   $A2'$   $P2'$  from-R  $l_0$   $a_0$ 
proof (standard, goal-cases)
  case prems: ( $1$   $x$   $y$   $S$ ) — complete
  then show ?case
    unfolding  $A2'$ -def  $P2'$ -def using from-R-fst
    by (clarify; cases x; cases y; simp; fastforce dest!: complete[of - - - R-of
 $S$ ])
  next
  case prems: ( $2$   $a$   $a'$ ) —  $P2$  invariant

```

**then show** *?case*  
 by (*meson A2'-def P2'-def P2-invariant*)  
**next**  
 case *prems: 3 — P2 start*  
**then show** *?case*  
 using *P2'-def P2-a0' from-R-fst by fastforce*  
**qed**

**sublocale** *P2-invariant': Graph-Invariant-Start A2 (l<sub>0</sub>, a<sub>0</sub>) P2*  
 by (*standard; rule P2-a0'*)

**end**

**locale** *Double-Simulation-Finite-Complete-paired = Double-Simulation-Complete-paired*  
 +  
**assumes** *finite-abstract-reachable: finite {(l, a). A2\*\* (l<sub>0</sub>, a<sub>0</sub>) (l, a) ∧ P2 (l, a)}*  
**begin**

**interpretation** *Bisimulation-Invariant A2 A2' λ (l, Z) b. b = from-R l Z P2 P2'*  
 by (*rule A2-A2'-bisim[OF P2-invariant]*)

**sublocale** *Double-Simulation-Finite-Complete C A1' P1' A2' P2' from-R l<sub>0</sub> a<sub>0</sub>*

**proof** (*standard, goal-cases*)

**case** *prems: 1 — The set of abstract reachable states is finite.*  
**have** \*:  $\exists l. x = \text{from-R } l (R\text{-of } x) \wedge A2^{**} (l_0, a_0) (l, R\text{-of } x)$   
**if** *sim.Steps.reaches (from-R l<sub>0</sub> a<sub>0</sub>) x for x*  
**using** *bisim.B-A-reaches[OF that, of (l<sub>0</sub>, a<sub>0</sub>)] P2-a0' P2'-def equiv'-def from-R-fst by fastforce*  
**have**  $\{a. \text{sim.Steps.reaches (from-R } l_0 \ a_0) \ a\}$   
 $\subseteq (\lambda (l, R). \text{from-R } l \ R) \ \{(l, a). A2^{**} (l_0, a_0) (l, a) \wedge P2 (l, a)\}$   
**using** *P2-a0' by (fastforce dest: \* intro: P2-invariant'.invariant-reaches)*  
**then show** *?case*  
**using** *finite-abstract-reachable by (auto elim!: finite-subset)*  
**qed**

**end**

**locale** *Double-Simulation-Complete-Bisim-paired = Double-Simulation-Complete-paired*  
 +

**assumes** *A1-complete: C (l, x) (l', y)  $\implies$  P1 (l, S)  $\implies$  x ∈ S  $\implies$   $\exists$  T. A1 (l, S) (l', T) ∧ y ∈ T*

**and** *P1-invariant*:  $P1(l, S) \implies A1(l, S) (l', T) \implies P1(l', T)$   
**begin**

**sublocale** *Double-Simulation-Complete-Bisim*  $C A1' P1' A2' P2'$  *from-R*  
 $l_0 a_0$

**proof** (*standard, goal-cases*)

**case** ( $1 x y S$ )

**then show** *?case*

**unfolding** *A1'-def P1'-def*

**apply** (*cases x; cases y; simp*)

**apply** (*drule A1-complete[where S = R-of S]*)

**apply** *fastforce*

**apply** *fastforce*

**apply** *clarify*

**subgoal for**  $a b l' ba l T$

**by** (*inst-existentials from-R l' T l l'*) (*auto simp: from-R-fst*)

**done**

**next**

**case** ( $2 S T$ )

**then show** *?case*

**unfolding** *A1'-def P1'-def* **by** (*auto intro: P1-invariant*)

**qed**

**end**

**locale** *Double-Simulation-Finite-Complete-Bisim-paired* = *Double-Simulation-Finite-Complete-paired*  
 +

*Double-Simulation-Complete-Bisim-paired*

**begin**

**sublocale** *Double-Simulation-Finite-Complete-Bisim*  $C A1' P1' A2' P2'$   
*from-R*  $l_0 a_0 ..$

**end**

**locale** *Double-Simulation-Complete-Bisim-Cover-paired* =

*Double-Simulation-Complete-Bisim-paired* +

**assumes** *P2-P1-cover*:  $P2(l, a) \implies x \in a \implies \exists a'. a \cap a' \neq \{\} \wedge P1$   
 $(l, a') \wedge x \in a'$

**begin**

**sublocale** *Double-Simulation-Complete-Bisim-Cover*  $C A1' P1' A2' P2'$   
*from-R*  $l_0 a_0$

**apply** *standard*

```

unfolding P2'-def P1'-def
apply clarify
apply (drule P2-P1-cover, force)
apply clarify
subgoal for a aa b l a'
  by (inst-existentials from-R l a') (fastforce simp: from-R-fst)+
done

end

locale Double-Simulation-Finite-Complete-Bisim-Cover-paired =
  Double-Simulation-Complete-Bisim-Cover-paired +
  Double-Simulation-Finite-Complete-Bisim-paired
begin

sublocale Double-Simulation-Finite-Complete-Bisim-Cover C A1' P1' A2'
  P2' from-R l0 a0 ..

end

locale Double-Simulation-Complete-Abstraction-Prop-paired =
  Double-Simulation-Complete-paired +
  fixes P :: 'a ⇒ bool — The property we want to check
  assumes P2-non-empty: P2 (l, a) ⇒ a ≠ {}
begin

definition φ = P o fst

lemma P2-φ:
  a ∩ Collect φ = a if P2' a a ∩ Collect φ ≠ {}
  using that unfolding φ-def P2'-def by (auto simp del: fst-conv)

sublocale Double-Simulation-Complete-Abstraction-Prop C A1' P1' A2'
  P2' from-R l0 a0 φ
proof (standard, goal-cases)
  case (1 a b)
  then obtain l where ∀ x∈b. fst x = l
    unfolding A1'-def by fast
  then show ?case
    unfolding φ-def by (auto simp del: fst-conv)
next
  case (2 a)
  then show ?case
    by — (frule P2-φ, auto)

```

```

next
  case prems: (3 a)
  then have  $P2' a$ 
    by (simp add: P2-invariant.invariant-reaches)
  from  $P2\text{-}\varphi[OF\ this]\ prems$  show ?case
    by simp
next
  case (4 a)
  then show ?case
    unfolding  $P2'\text{-}def$  by (auto dest!: P2-non-empty)
qed

end

locale Double-Simulation-Finite-Complete-Abstraction-Prop-paired =
  Double-Simulation-Complete-Abstraction-Prop-paired +
  Double-Simulation-Finite-Complete-paired
begin

sublocale Double-Simulation-Finite-Complete-Abstraction-Prop C A1' P1'
   $A2' P2' from\text{-}R\ l_0\ a_0\ \varphi\ ..$ 

end

locale Double-Simulation-Complete-Abstraction-Prop-Bisim-paired =
  Double-Simulation-Complete-Abstraction-Prop-paired +
  Double-Simulation-Complete-Bisim-paired
begin

interpretation bisim: Bisimulation-Invariant A2 A2'  $\lambda\ (l, Z)\ b.\ b = from\text{-}R\ l\ Z\ P2\ P2'$ 
  by (rule A2-A2'-bisim[OF P2-invariant])

sublocale Double-Simulation-Complete-Abstraction-Prop-Bisim
   $C\ A1'\ P1'\ A2'\ P2' from\text{-}R\ l_0\ a_0\ \varphi\ ..$ 

lemma P2'-non-empty:
   $P2' a \implies a \neq \{\}$ 
  using P2-non-empty unfolding P2'-def by force

lemma from-R-int- $\varphi[simp]$ :
   $from\text{-}R\ l\ R \cap\ Collect\ \varphi = from\text{-}R\ l\ R$  if  $P\ l$ 
  using from-R-fst that unfolding  $\varphi\text{-}def$  by fastforce

```

**interpretation**  $G_\varphi$ : *Graph-Start-Defs*  
 $\lambda (l, Z) (l', Z'). A2 (l, Z) (l', Z') \wedge P l' (l_0, a_0) .$

**interpretation** *Bisimulation-Invariant*  $\lambda (l, Z) (l', Z'). A2 (l, Z) (l', Z')$   
 $\wedge P l'$   
 $A2\text{-}\varphi \lambda (l, Z) b. b = \text{from-R } l \ Z \ P2 \ P2'$   
**apply** *standard*  
**unfolding**  $A2\text{-}\varphi\text{-def}$   
**apply** *clarify*  
**subgoal** **for**  $l \ a \ l' \ a'$   
**apply** (*drule bisim.A-B-step*)  
**prefer** 3  
**apply** *assumption*  
**apply** *safe*  
**apply** (*frule P-invariant, assumption+*)  
**using** *from-R-fst* **by** (*fastforce simp:  $\varphi\text{-def}$   $P2'\text{-def}$   $\text{dest!}: P2'\text{-non-empty}$* )  
**subgoal** **for**  $a \ a' \ b'$   
**apply** *clarify*  
**apply** (*drule bisim.B-A-step*)  
**prefer** 2  
**apply** *assumption*  
**apply** *safe*  
**apply** (*frule P2-invariant, assumption+*)  
**apply** (*subst (asm) (3)  $\varphi\text{-def}$* )  
**apply** *simp*  
**apply** (*elim allE impE, assumption*)  
**using** *from-R-fst* **apply** *force*  
**apply** (*subst (asm) (2) from-R-int- $\varphi$* )  
**using** *from-R-fst* **by** *fastforce+*  
**subgoal**  
**by** *blast*  
**subgoal**  
**using**  *$\varphi\text{-P2-compatible}$*  **by** *blast*  
**done**

**lemma** *from-R-subs- $\varphi$* :  
 $\text{from-R } l \ a \subseteq \text{Collect } \varphi \text{ if } P \ l$   
**using** *that* **unfolding**  *$\varphi\text{-def}$  from-R-def* **by** *auto*

**lemma** *P2'-from-R*:  
 $\exists l' \ Z'. x = \text{from-R } l' \ Z' \text{ if } P2' \ x$   
**using** *that* **unfolding** *P2'-def* **by** (*fastforce dest: from-R-R-of*)

**lemma** *P2-from-R-list'*:

$\exists as'. \text{map } (\lambda(x, y). \text{from-R } x \ y) \ as' = as$  **if** *list-all*  $P2'$  *as*  
**by** (*rule list-all-map*[*OF - that*]) (*auto dest!*:  $P2'$ -*from-R*)

**end**

**locale** *Double-Simulation-Finite-Complete-Abstraction-Prop-Bisim-paired* =  
*Double-Simulation-Complete-Abstraction-Prop-Bisim-paired* +  
*Double-Simulation-Finite-Complete-Bisim-paired*

**begin**

**interpretation** *bisim*: *Bisimulation-Invariant*  $A2 \ A2' \ \lambda \ (l, Z) \ b. \ b = \text{from-R}$   
 $l \ Z \ P2 \ P2'$

**by** (*rule A2-A2'-bisim*[*OF P2-invariant*])

**sublocale** *Double-Simulation-Finite-Complete-Abstraction-Prop-Bisim*

$C \ A1' \ P1' \ A2' \ P2' \ \text{from-R} \ l_0 \ a_0 \ \varphi \ ..$

**interpretation**  $G_\varphi$ : *Graph-Start-Defs*

$\lambda \ (l, Z) \ (l', Z'). \ A2 \ (l, Z) \ (l', Z') \ \wedge \ P \ l' \ (l_0, a_0) \ .$

**interpretation** *Bisimulation-Invariant*  $\lambda \ (l, Z) \ (l', Z'). \ A2 \ (l, Z) \ (l', Z')$   
 $\wedge \ P \ l'$

$A2\text{-}\varphi \ \lambda \ (l, Z) \ b. \ b = \text{from-R} \ l \ Z \ P2 \ P2'$

**apply** *standard*

**unfolding**  $A2\text{-}\varphi\text{-def}$

**apply** *clarify*

**subgoal for**  $l \ a \ l' \ a'$

**apply** (*drule bisim.A-B-step*)

**prefer** 3

**apply** *assumption*

**apply** *safe*

**apply** (*frule P-invariant, assumption+*)

**using** *from-R-fst* **by** (*fastforce simp: \varphi-def P2'-def dest!: P2'-non-empty*)+

**subgoal for**  $a \ a' \ b'$

**apply** *clarify*

**apply** (*drule bisim.B-A-step*)

**prefer** 2

**apply** *assumption*

**apply** *safe*

**apply** (*frule P2-invariant, assumption+*)

**apply** (*subst (asm) (3) \varphi-def*)

**apply** *simp*

**apply** (*elim allE impE, assumption*)

**using** *from-R-fst* **apply** *force*

```

apply (subst (asm) (2) from-R-int-φ)
using from-R-fst by fastforce+
subgoal
  by blast
subgoal
  using φ-P2-compatible by blast
done

```

**theorem** *Alw-ev-mc*:

```

(∀ x₀ ∈ a₀. sim.Alw-ev (Not ∘ φ) (l₀, x₀)) ↔
  ¬ P l₀ ∨ (∄ as a bs. G_φ.steps ((l₀, a₀) # as @ a # bs @ [a]))
apply (subst steps-map-equiv[of λ (l, Z). from-R l Z - from-R l₀ a₀])
  apply force
  apply (clarsimp simp: from-R-def)
subgoal
  by (fastforce dest!: P2'-non-empty)
  apply (simp; fail)
  apply (rule P2-a₀'; fail)
  apply (rule phi.P2-a₀; fail)
proof (cases P l₀, goal-cases)
  case 1
  have *: (∀ x₀ ∈ a₀. sim.Alw-ev (Not ∘ φ) (l₀, x₀)) ↔ (∀ x₀ ∈ from-R l₀ a₀.
sim.Alw-ev (Not ∘ φ) x₀)
  unfolding from-R-def by auto
  from ⟨P →⟩ show ?case
  unfolding *
  apply (subst Alw-ev-mc[OF from-R-subst-φ], assumption)
  apply (auto simp del: map-map)
  apply (rule phi.P2-invariant.invariant-steps)
  apply (auto dest!: P2'-from-R P2-from-R-list')
  done
next
  case 2
  then have ∀ x₀ ∈ a₀. sim.Alw-ev (Not ∘ φ) (l₀, x₀)
  unfolding sim.Alw-ev-def by (force simp: φ-def)
  with ⟨¬ P l₀⟩ show ?case
  by auto
qed

```

**theorem** *Alw-ev-mc1*:

```

(∀ x₀ ∈ a₀. sim.Alw-ev (Not ∘ φ) (l₀, x₀)) ↔ ¬ (P l₀ ∧ (∃ a. G_φ.reachable
a ∧ G_φ.reaches1 a a))
unfolding Alw-ev-mc using G_φ.reachable-cycle-iff by auto

```

**end**

**context** *Double-Simulation-Complete-Bisim-Cover-paired*  
**begin**

**interpretation** *bisim: Bisimulation-Invariant A2 A2' λ (l, Z) b. b = from-R l Z P2 P2'*  
**by** (rule *A2-A2'-bisim[OF P2-invariant]*)

**interpretation** *Start: Double-Simulation-Complete-Abstraction-Prop-Bisim-paired C A1 P1 A2 P2 l0 a0 λ -. True*  
**using** *P2-cover* **by** – (standard, blast)

**lemma** *sim-reaches-equiv:*

*P2-invariant'.reaches (l, Z) (l', Z') ⟷ sim.Steps.reaches (from-R l Z) (from-R l' Z')*  
**if** *P2 (l, Z)*  
**apply** (subst *bisim.reaches-equiv[of λ (l, Z). from-R l Z]*)  
  **apply** *force*  
  **apply** *clarsimp*  
**subgoal**  
  **by** (metis *Int-emptyI R-of-from-R from-R-fst sim.P2-cover*)  
  **apply** (rule *that*)  
**subgoal**  
  **apply** *clarsimp*  
  **using** *P2'-def from-R-fst that* **by** *force*  
**by** *auto*

**lemma** *reaches-all:*

**assumes**  
   $\bigwedge u u' R l. u \in R \implies u' \in R \implies P1 (l, R) \implies P l u \longleftrightarrow P l u'$   
**shows**  
   $(\forall u. (\exists x_0 \in \bigcup (sim.closure (from-R l_0 a_0)). sim.reaches x_0 (l, u)) \longrightarrow P l u) \longleftrightarrow$   
   $(\forall Z u. P2\text{-invariant}'.reaches (l_0, a_0) (l, Z) \wedge u \in Z \longrightarrow P l u)$   
**proof** –  
  **let**  $?P = \lambda (l, u). P l u$   
  **have** \*:  $\bigwedge a x y. x \in a \implies y \in a \implies P1' a \implies ?P x = ?P y$   
    **unfolding** *P1'-def* **by** *clarsimp (subst assms[rotated 2], force+, metis fst-conv)+*  
  **let**  $?P = \lambda (l', u). l' = l \longrightarrow P l u$   
  **have** \*:  $x \in a \implies y \in a \implies P1' a \implies ?P x = ?P y$  **for**  $a x y$   
    **by** (frule \*[of x a y], *assumption+*; *auto simp: P1'-def; metis fst-conv*)  
  **have**

```

(∀ b. (∃ y ∈ sim.closure (from-R l0 a0). ∃ x0 ∈ y. sim.reaches x0 (l, b)) →
P l b) ↔
(∀ b ba. sim.Steps.reaches (from-R l0 a0) b ∧ (l, ba) ∈ b → P l ba)
unfolding sim.reaches-steps-iff sim.Steps.reaches-steps-iff
apply safe
subgoal for b b' xs
apply (rule reaches-all-1[of ?P xs (l, b'), simplified])
apply (erule *; assumption; fail)
apply (simp; fail)+
done

subgoal premises prems for b y a b' xs
apply (rule
reaches-all-2[of ?P xs y, unfolded ⟨last xs = (l, b)⟩, simplified]
)
apply (erule *; assumption; fail)
using prems by auto
done
then show ?thesis
unfolding sim-reaches-equiv[OF P2-a0]
apply simp
subgoal premises prems
apply safe
subgoal for Z u
unfolding from-R-def by auto
subgoal for a u
apply (frule P2-invariant.invariant-reaches)
apply (auto dest!: Start.P2'-from-R simp: from-R-def)
done
done
done
qed

```

**context**

**fixes**  $P Q :: 'a \Rightarrow \text{bool}$  — The state properties we want to check  
**begin**

**definition**  $\varphi' = P \circ \text{fst}$

**definition**  $\psi = Q \circ \text{fst}$

**lemma**  $\psi$ -closure-compatible:

$\psi(l, x) \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P1(l, a) \Longrightarrow \psi(l, y)$   
**unfolding**  $\varphi'$ -def  $\psi$ -def **by** auto

**lemma**  *$\psi$ -closure-compatible'*:

$(\text{Not } o \psi) (l, x) \implies x \in a \implies y \in a \implies P1 (l, a) \implies (\text{Not } o \psi) (l, y)$   
**by** (*auto dest:  $\psi$ -closure-compatible*)

**lemma** *P1-P1'*:

$R \neq \{\}$   $\implies P1 (l, R) \implies P1' (\text{from-R } l R)$   
**using** *P1'-def from-R-fst by fastforce*

**lemma**  *$\psi$ -Alw-ev-compatible*:

**assumes**  $u \in R \ u' \in R \ P1 (l, R)$   
**shows**  $\text{sim.Alw-ev } (\text{Not } o \psi) (l, u) = \text{sim.Alw-ev } (\text{Not } o \psi) (l, u')$   
**apply** (*rule bisim.Alw-ev-compatible[of - - from-R l R]*)  
**subgoal for**  $x \ a \ y$   
**using**  *$\psi$ -closure-compatible unfolding P1'-def by (metis  $\psi$ -def comp-def)*  
**using** *assms by (auto intro: P1-P1')*

**interpretation** *Graph-Start-Defs A2*  $(l_0, a_0)$  .

**interpretation**  $G_\psi$ : *Graph-Start-Defs*

$\lambda (l, Z) (l', Z'). A2 (l, Z) (l', Z') \wedge Q \ l' (l_0, a_0)$  .

**end**

**end**

**context** *Double-Simulation-Finite-Complete-Bisim-Cover-paired*

**begin**

**interpretation** *bisim: Bisimulation-Invariant A2 A2'*  $\lambda (l, Z) b. b = \text{from-R } l \ Z \ P2 \ P2'$

**by** (*rule A2-A2'-bisim[OF P2-invariant]*)

**context**

**fixes**  $P \ Q :: 'a \Rightarrow \text{bool}$  — The state properties we want to check

**begin**

**interpretation** *Graph-Start-Defs A2*  $(l_0, a_0)$  .

**interpretation**  $G_\psi$ : *Graph-Start-Defs*

$\lambda (l, Z) (l', Z'). A2 (l, Z) (l', Z') \wedge Q \ l' (l_0, a_0)$  .

**lemma** *Alw-ev-mc1*:

$(\forall x_0 \in \text{from-R } l \ Z. \text{sim.Alw-ev } (\text{Not } o \psi \ Q) \ x_0) \longleftrightarrow$

$\neg (Q \ l \wedge (\exists a. G_\psi.\text{reaches} (l, Z) \ a \wedge G_\psi.\text{reaches1} \ a \ a))$   
**if**  $P2\text{-invariant}'.\text{reachable} (l, Z)$  **for**  $l \ Z$   
**proof** –  
**from** *that* **have**  $P2 (l, Z)$   
**using**  $P2\text{-invariant}'.\text{invariant-reaches}$  **unfolding**  $P2\text{-invariant}'.\text{reachable-def}$   
**by** *auto*  
**interpret**  $Start'$ : *Double-Simulation-Finite-Complete-Abstraction-Prop-Bisim-paired*  
 $C \ A1 \ P1 \ A2 \ P2 \ l \ Z \ Q$   
**apply** *standard*  
**subgoal**  
**by** (*fact complete*)  
**subgoal**  
**by** (*fact P2-invariant*)  
**subgoal**  
**by** (*fact <P2 (l, Z)>*)  
**subgoal**  
**using**  $P2\text{-cover}$  **by** *blast*  
**subgoal**  
**by** (*fact A1-complete*)  
**subgoal**  
**by** (*fact P1-invariant*)  
**subgoal**  
**proof** –  
**have**  $\{(l', a). A2^{**} (l, Z) (l', a) \wedge P2 (l', a)\} \subseteq \{(l, a). A2^{**} (l_0, a_0)$   
 $(l, a) \wedge P2 (l, a)\}$   
**using** *that* **unfolding**  $P2\text{-invariant}'.\text{reachable-def}$  **by** *auto*  
**with** *finite-abstract-reachable* **show** *?thesis*  
**by** – (*erule finite-subset*)  
**qed**  
**done**  
**show** *?thesis*  
**using**  $Start'.Alw\text{-ev}\text{-mc1}[unfolding \ Start'.\varphi\text{-def}]$   
**unfolding**  $\psi\text{-def}$   $Graph\text{-Start}\text{-Defs}.\text{reachable-def}$  **from**  $R\text{-def}$  **by** *auto*  
**qed**

**theorem** *leadsto-mc1*:

$(\forall x_0 \in a_0. \text{sim}.\text{leadsto} (\varphi' \ P) (Not \circ \psi \ Q) (l_0, x_0)) \longleftrightarrow$   
 $(\nexists x. P2\text{-invariant}'.\text{reaches} (l_0, a_0) \ x \wedge P (fst \ x) \wedge Q (fst \ x)$   
 $\wedge (\exists a. G_\psi.\text{reaches} \ x \ a \wedge G_\psi.\text{reaches1} \ a \ a)$   
 $)$

**if** *no-deadlock*:  $\forall x_0 \in a_0. \neg \text{sim}.\text{deadlock} (l_0, x_0)$

**proof** –

**from**  $\text{steps}\text{-Steps}\text{-no-deadlock}[OF \ \text{no-deadlock-closureI}]$  *no-deadlock* **have**  
 $\neg \text{sim}.\text{Steps}.\text{deadlock} (\text{from-R} \ l_0 \ a_0)$

**unfolding from-R-def by auto**  
**then have**  $no\_deadlock'$ :  $\neg P2\_invariant'.deadlock (l_0, a_0)$   
**by** (*subst bisim.deadlock-iff*) (*auto simp: P2-a0' from-R-fst P2'-def*)  
**have**  $(\forall x_0 \in a_0. sim.leadsto (\varphi' P) (Not \circ \psi Q) (l_0, x_0)) \longleftrightarrow$   
 $(\forall x_0 \in from-R l_0 a_0. sim.leadsto (\varphi' P) (Not \circ \psi Q) x_0)$

**unfolding from-R-def by auto**  
**also have**  $\dots \longleftrightarrow sim.Steps.Alw-alw (\lambda a. \forall c \in a. \varphi' P c \longrightarrow sim.Alw-ev$   
 $(Not \circ \psi Q) c) (from-R l_0 a_0)$   
**apply** (*rule Leadsto-iff2[OF - - -]*)  
**subgoal for**  $a x y$   
**unfolding**  $P1'-def \varphi'-def$  **by** (*auto dest: fst-simp*)  
**subgoal for**  $a x y$   
**unfolding**  $P1'-def \psi-def$  **by** (*auto dest: fst-simp*)  
**subgoal**  
**using**  $no\_deadlock$  **unfolding from-R-def by auto**  
**done**  
**also have**  
 $\dots \longleftrightarrow P2\_invariant'.Alw-alw (\lambda(l,Z). \forall c \in from-R l Z. \varphi' P c \longrightarrow sim.Alw-ev$   
 $(Not \circ \psi Q) c) (l_0, a_0)$   
**by** (*auto simp: bisim.A-B.equiv'-def P2-a0 P2-a0' intro!: bisim.Alw-alw-iff-strong[symmetric]*)  
**also have**  
 $\dots \longleftrightarrow P2\_invariant'.Alw-alw$   
 $(\lambda(l, Z). P l \longrightarrow \neg (Q l \wedge (\exists a. G_\psi.reaches (l, Z) a \wedge G_\psi.reaches1 a$   
 $a))) (l_0, a_0)$   
**by** (*rule P2-invariant'.Alw-alw-iff-default*)  
 $(auto simp: \varphi'-def from-R-def dest: Alw-ev-mc1[symmetric])$   
**also have**  
 $\dots \longleftrightarrow (\nexists x. P2\_invariant'.reaches (l_0, a_0) x \wedge P (fst x) \wedge Q (fst x)$   
 $\wedge (\exists a. G_\psi.reaches x a \wedge G_\psi.reaches1 a a))$   
**unfolding**  $P2\_invariant'.Alw-alw-iff$  **by** (*auto simp: P2-invariant'.Ex-ev*  
 $no\_deadlock'$ )  
**finally show**  $?thesis$  .  
**qed**

**end**

**end**

**The second bisimulation property in prestable and complete simulation graphs.** **context** *Simulation-Graph-Complete-Prestable*  
**begin**

**lemma** *C-A-bisim*:

*Bisimulation-Invariant*  $C\ A\ (\lambda\ x\ a.\ x \in a)\ (\lambda\ -. \text{True})\ P$   
**by** (*standard*; *blast intro*: *complete dest*: *prestable*)

**interpretation** *Bisimulation-Invariant*  $C\ A\ \lambda\ x\ a.\ x \in a\ \lambda\ -. \text{True}\ P$   
**by** (*rule C-A-bisim*)

**lemma** *C-A-Leadsto-iff*:

**fixes**  $\varphi\ \psi :: 'a \Rightarrow \text{bool}$

**assumes**  $\varphi$ -*compatible*:  $\bigwedge\ x\ y\ a.\ \varphi\ x \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P\ a \Longrightarrow \varphi\ y$

**and**  $\psi$ -*compatible*:  $\bigwedge\ x\ y\ a.\ \psi\ x \Longrightarrow x \in a \Longrightarrow y \in a \Longrightarrow P\ a \Longrightarrow \psi\ y$

**and**  $x \in a\ P\ a$

**shows**  $\text{leadsto}\ \varphi\ \psi\ x = \text{Steps.leadsto}\ (\lambda\ a.\ \forall\ x \in a.\ \varphi\ x)\ (\lambda\ a.\ \forall\ x \in a.\ \psi\ x)\ a$

**by** (*rule Leadsto-iff*)

(*auto intro*:  $\varphi$ -*compatible*  $\psi$ -*compatible simp*:  $\langle x \in a \rangle\ \langle P\ a \rangle$  *simulation.equiv'-def*)

**end**

## Comments

- Pre-stability can easily be extended to infinite runs (see construction with *sscan* above)
- Post-stability can not
- Pre-stability + Completeness means that for every two concrete states in the same abstract class, there are equivalent runs
- For Büchi properties, the predicate has to be compatible with whole closures instead of single *P1*-states. This is because for a finite graph where every node has at least indegree one, we cannot necessarily conclude that there is a cycle through *every* node.

**locale** *Graph-Abstraction* =

*Graph-Defs*  $A$  **for**  $A :: 'a\ \text{set} \Rightarrow 'a\ \text{set} \Rightarrow \text{bool} +$

**fixes**  $\alpha :: 'a\ \text{set} \Rightarrow 'a\ \text{set}$

**assumes** *idempotent*:  $\alpha(\alpha(x)) = \alpha(x)$

**assumes** *enlarging*:  $x \subseteq \alpha(x)$

**assumes**  $\alpha$ -*mono*:  $x \subseteq y \Longrightarrow \alpha(x) \subseteq \alpha(y)$

**assumes** *mono*:  $a \subseteq a' \Longrightarrow A\ a\ b \Longrightarrow \exists\ b'.\ b \subseteq b' \wedge A\ a'\ b'$

**assumes** *finite-abstraction*: *finite* ( $\alpha$  ' *UNIV*)

**begin**

**definition** *E* where  $E a b \equiv \exists b'. A a b' \wedge b = \alpha(b')$

**interpretation** *sim1*: *Simulation-Invariant*  $A E \lambda a b. \alpha(a) \subseteq b \lambda-. True$   
 $\lambda-. True$

```
apply standard
unfolding E-def
  apply auto
  apply (frule mono[rotated])
  apply (erule order.trans[rotated], rule enlarging)
  apply (auto intro!:  $\alpha$ -mono)
done
```

**interpretation** *sim2*: *Simulation-Invariant*  $A E \lambda a b. a \subseteq b \lambda-. True \lambda x.$   
 $\alpha(x) = x$

```
apply standard
subgoal
  unfolding E-def
  apply auto
  apply (drule (1) mono)
  apply safe
  apply (intro conjI exI)
  apply assumption
  apply (rule HOL.refl)
  apply (erule order.trans, rule enlarging)
done
apply assumption
unfolding E-def
apply (elim exE conjE)
apply (simp add: idempotent)
done
```

This variant needs the least assumptions.

**interpretation** *sim3*: *Simulation-Invariant*  $A E \lambda a b. a \subseteq b \lambda-. True \lambda-. True$   
 $True$

```
apply standard
unfolding E-def
  apply auto
  apply (drule (1) mono)
  apply safe
  apply (intro conjI exI)
  apply assumption
  apply (rule HOL.refl)
  apply (erule order.trans, rule enlarging)
```

done

**interpretation** *sim4*: *Simulation-Invariant A E λa b. a ⊆ b λ-. True λa.*

$\exists a'. \alpha a' = a$

apply *standard*

unfolding *E-def*

apply *auto*

apply (*drule* (1) *mono*)

apply *safe*

apply (*intro conjI exI*)

apply *assumption*

apply (*rule HOL.refl*)

apply (*erule order.trans, rule enlarging*)

done

end

lemmas [*simp del*] = *holds.simps*

end

**theory** *Simulation-Graphs-TA*

imports *Simulation-Graphs DBM-Zone-Semantics Approx-Beta*

**begin**

## 7.9 Instantiation of Simulation Locales

**inductive** *step-trans* ::

$(a, c, t, s) ta \Rightarrow s \Rightarrow (c, (t::time)) cval \Rightarrow ((c, t) cconstraint \times a \times c list)$

$\Rightarrow s \Rightarrow (c, t) cval \Rightarrow bool$

$(\langle \cdot \vdash_t \langle \cdot, \cdot \rangle \rightarrow \langle \cdot, \cdot \rangle [61,61,61] 61)$

**where**

$\llbracket A \vdash l \xrightarrow{g,a,r} l'; u \vdash g; u' \vdash inv\text{-of } A l'; u' = [r \rightarrow 0]u \rrbracket$

$\Longrightarrow (A \vdash_t \langle l, u \rangle \rightarrow_{(g,a,r)} \langle l', u' \rangle)$

**inductive** *step-trans'* ::

$(a, c, t, s) ta \Rightarrow s \Rightarrow (c, (t::time)) cval \Rightarrow (c, t) cconstraint \times a \times c list$

$\Rightarrow s \Rightarrow (c, t) cval \Rightarrow bool$

$(\langle \cdot \vdash'' \langle \cdot, \cdot \rangle \rightarrow \langle \cdot, \cdot \rangle [61,61,61,61] 61)$

**where**

*step'*:  $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \Longrightarrow A \vdash_t \langle l', u' \rangle \rightarrow_t \langle l'', u'' \rangle \Longrightarrow A \vdash' \langle l, u \rangle \rightarrow^t \langle l'', u'' \rangle$

**inductive** *step-trans-z* ::

$(\ 'a, 'c, 't, 's) ta \Rightarrow 's \Rightarrow ('c, ('t::time)) zone$   
 $\Rightarrow (('c, 't) cconstraint \times 'a \times 'c list) action \Rightarrow 's \Rightarrow ('c, 't) zone \Rightarrow bool$   
 $(\langle - \vdash \langle -, - \rangle \rightsquigarrow^{\tau} \langle -, - \rangle \rangle [61,61,61,61] 61)$

**where**

*step-trans-t-z*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^{\tau} \langle l, Z^{\uparrow} \cap \{u. u \vdash inv\text{-of } A \ l\} \rangle \mid$

*step-trans-a-z*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^{1(g,a,r)} \langle l', zone\text{-set } (Z \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash inv\text{-of } A \ l'\} \rangle$

**if**  $A \vdash l \longrightarrow^{g,a,r} l'$

**inductive** *step-trans-z'* ::

$(\ 'a, 'c, 't, 's) ta \Rightarrow 's \Rightarrow ('c, ('t::time)) zone \Rightarrow (('c, 't) cconstraint \times 'a \times 'c list)$   
 $\Rightarrow 's \Rightarrow ('c, 't) zone \Rightarrow bool$   
 $(\langle - \vdash'' \langle -, - \rangle \rightsquigarrow^{\tau} \langle -, - \rangle \rangle [61,61,61,61] 61)$

**where**

*step-trans-z'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^{\tau} \langle l, Z^{\wedge} \rangle \Longrightarrow A \vdash \langle l, Z^{\wedge} \rangle \rightsquigarrow^{1t} \langle l', Z'^{\wedge} \rangle \Longrightarrow A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z'^{\wedge} \rangle$

**lemmas** [*intro*] =

*step-trans.intros*

*step-trans'.intros*

*step-trans-z.intros*

*step-trans-z'.intros*

**context**

**notes** [*elim!*] =

*step.cases step-t.cases*

*step-trans.cases step-trans'.cases step-trans-z.cases step-trans-z'.cases*

**begin**

**lemma** *step-trans-t-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^{\tau} \langle l', Z^{\wedge} \rangle \Longrightarrow \forall u' \in Z'. \exists u \in Z. \exists d. A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$   
**by** (*auto 4 5 simp: zone-delay-def zone-set-def*)

**lemma** *step-trans-a-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^{1t} \langle l', Z^{\wedge} \rangle \Longrightarrow \forall u' \in Z'. \exists u \in Z. \exists d. A \vdash_t \langle l, u \rangle \rightarrow_t \langle l', u' \rangle$   
**by** (*auto 4 4 simp: zone-delay-def zone-set-def*)

**lemma** *step-trans-a-z-complete*:

$A \vdash_t \langle l, u \rangle \rightarrow_t \langle l', u' \rangle \Longrightarrow u \in Z \Longrightarrow \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow^{1t} \langle l', Z^{\wedge} \rangle \wedge u'$

$\in Z'$

by (auto 4 4 simp: zone-delay-def zone-set-def elim!: step-a.cases)

**lemma** *step-trans-t-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow^\tau \langle l', Z' \rangle \wedge u' \in Z'$

by (auto 4 4 simp: zone-delay-def zone-set-def elim!: step-a.cases)

**lemma** *step-trans-t-z-iff*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^\tau \langle l', Z' \rangle = A \vdash \langle l, Z \rangle \rightsquigarrow_\tau \langle l', Z' \rangle$

by auto

**lemma** *step-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies u \in Z \implies \exists Z' t. A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle \wedge u' \in Z'$

by (auto 4 4 simp: zone-delay-def zone-set-def elim!: step-a.cases)

**lemma** *step-trans-a-z-exact*:

$u' \in Z'$  if  $A \vdash_t \langle l, u \rangle \rightarrow_t \langle l', u' \rangle$   $A \vdash \langle l, Z \rangle \rightsquigarrow^{!t} \langle l', Z' \rangle$   $u \in Z$   
using that by (auto 4 4 simp: zone-delay-def zone-set-def)

**lemma** *step-trans-t-z-exact*:

$u' \in Z'$  if  $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$   $A \vdash \langle l, Z \rangle \rightsquigarrow^\tau \langle l', Z' \rangle$   $u \in Z$   
using that by (auto simp: zone-delay-def)

**lemma** *step-trans-z'-exact*:

$u' \in Z'$  if  $A \vdash' \langle l, u \rangle \rightarrow^t \langle l', u' \rangle$   $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle$   $u \in Z$   
using that by (auto 4 4 simp: zone-delay-def zone-set-def)

**lemma** *step-trans-z-step-z-action*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{1a} \langle l', Z' \rangle$  if  $A \vdash \langle l, Z \rangle \rightsquigarrow^{!(g,a,r)} \langle l', Z' \rangle$   
using that by auto

**lemma** *step-trans-z-step-z*:

$\exists a. A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle$  if  $A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle$   
using that by auto

**lemma** *step-z-step-trans-z-action*:

$\exists g r. A \vdash \langle l, Z \rangle \rightsquigarrow^{!(g,a,r)} \langle l', Z' \rangle$  if  $A \vdash \langle l, Z \rangle \rightsquigarrow_{1a} \langle l', Z' \rangle$   
using that by (auto 4 4)

**lemma** *step-z-step-trans-z*:

$\exists t. A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle$  if  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle$   
using that by cases auto

end

**lemma** *step-z'-step-trans-z'*:

$\exists t. A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle$

**using** that **unfolding** *step-z'-def*

**by** (*auto dest!*: *step-z-step-trans-z-action simp: step-trans-t-z-iff[symmetric]*)

**lemma** *step-trans-z'-step-z'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle$  **if**  $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle$

**using** that **unfolding** *step-z'-def*

**by** (*auto elim!*: *step-trans-z'.cases dest!*: *step-trans-z-step-z-action simp: step-trans-t-z-iff*)

**lemma** *step-trans-z-determ:*

$Z1 = Z2$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z1 \rangle$   $A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z2 \rangle$

**using** that **by** (*auto elim!*: *step-trans-z.cases*)

**lemma** *step-trans-z'-determ:*

$Z1 = Z2$  **if**  $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z1 \rangle$   $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z2 \rangle$

**using** that **by** (*auto elim!*: *step-trans-z'.cases step-trans-z.cases*)

**lemma** (**in** *Alpha-defs*) *step-trans-z-V*:  $A \vdash \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$

**by** (*induction rule: step-trans-z.induct; blast intro!*: *reset-V le-infI1 up-V*)

### 7.9.1 Additional Lemmas on Regions

**context** *AlphaClosure*

**begin**

**inductive** *step-trans-r* ::

$(\ 'a, 'c, t, 's) \text{ ta} \Rightarrow - \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow ((\ 'c, t) \text{ cconstraint} \times 'a \times 'c$   
*list*) *action*

$\Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow \text{bool}$

$(\langle -, - \vdash \langle -, - \rangle \rightsquigarrow^r \langle -, - \rangle \rangle [61,61,61,61,61] 61)$

**where**

*step-trans-t-r*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^\tau \langle l, R' \rangle$  **if**

*valid-abstraction*  $A X (\lambda x. \text{real } o \ k \ x) R \in \mathcal{R} \mid R' \in \text{Succ } (\mathcal{R} \ l) R R' \subseteq$   
 $\{\text{inv-of } A \ l\} \mid$

*step-trans-a-r*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^{1(g,a,r)} \langle l', R' \rangle$  **if**

*valid-abstraction*  $A X (\lambda x. \text{real } o \ k \ x) A \vdash l \longrightarrow^{g,a,r} l' R \in \mathcal{R} \mid$

$R \subseteq \{g\}$  region-set'  $R \ r \ 0 \subseteq R' \ R' \subseteq \{inv\text{-of } A \ l'\} \ R' \in \mathcal{R} \ l'$

**lemmas** [intro] = step-trans-r.intros

**lemma** step-trans-t-r-iff[simp]:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^\tau \langle l', R' \rangle = A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_\tau \langle l', R' \rangle$   
**by** (auto elim!: step-trans-r.cases)

**lemma** step-trans-r-step-r-action:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{1a} \langle l', R' \rangle$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^{1(g,a,r)} \langle l', R' \rangle$   
**using that by** (auto elim: step-trans-r.cases)

**lemma** step-r-step-trans-r-action:

$\exists g \ r. A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^{1(g,a,r)} \langle l', R' \rangle$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{1a} \langle l', R' \rangle$   
**using that by** (auto elim: step-trans-r.cases)

**inductive** step-trans-r' ::

$(\ 'a, \ 'c, \ t, \ 's) \ ta \Rightarrow \ - \Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ zone \Rightarrow (\ 'c, \ t) \ cconstraint \times \ 'a \times \ 'c$   
*list*

$\Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ zone \Rightarrow \ bool$

$(\langle -, - \rangle \vdash'' \langle -, - \rangle \rightsquigarrow^\tau \langle -, - \rangle) \ [61, 61, 61, 61, 61] \ 61)$

**where**

$A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^t \langle l', R'' \rangle$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^\tau \langle l, R' \rangle \ A, \mathcal{R} \vdash \langle l, R' \rangle \rightsquigarrow^{1t} \langle l', R'' \rangle$

**lemma** step-trans-r'-step-r':

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$  **if**  $A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^{(g,a,r)} \langle l', R' \rangle$   
**using that by** cases (auto dest: step-trans-r-step-r-action intro!: step-r'.intros)

**lemma** step-r'-step-trans-r':

$\exists g \ r. A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^{(g,a,r)} \langle l', R' \rangle$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$   
**using that by** cases (auto dest: step-r-step-trans-r-action intro!: step-trans-r'.intros)

**lemma** step-trans-a-r-sound:

**assumes**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^{1a} \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. A \vdash_t \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$

**using** assms **proof** cases

**case** A: (step-trans-a-r g a r)

**show** ?thesis

**unfolding** A(1) **proof**

**fix** u **assume** u ∈ R

**from**  $\langle u \in R \rangle \ A$  **have**  $u \vdash g \ [r \rightarrow 0]u \vdash \text{inv-of } A \ l' \ [r \rightarrow 0]u \in R'$

**unfolding** region-set'-def cval-def **by** auto

**with** A **show**  $\exists u' \in R'. A \vdash_t \langle l, u \rangle \rightarrow_{(g,a,r)} \langle l', u' \rangle$

by *auto*  
qed  
qed

**lemma** *step-trans-r'-sound*:

**assumes**  $A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^t \langle l', R' \rangle$

**shows**  $\forall u \in R. \exists u' \in R'. A \vdash' \langle l, u \rangle \rightarrow^t \langle l', u' \rangle$

**using** *assms* **by** *cases (auto 6 0 dest! step-trans-a-r-sound step-t-r-sound)*

**end**

**context** *AlphaClosure*

**begin**

**context**

**fixes**  $l\ l' :: 's$  **and**  $A :: ('a, 'c, t, 's) ta$

**assumes** *valid-abstraction: valid-abstraction A X k*

**begin**

**interpretation** *alpha: AlphaClosure-global - k l R l by standard (rule finite)*

**lemma** [*simp*]:  $alpha.cla = cla\ l$  **unfolding**  $alpha.cla-def\ cla-def ..$

**interpretation** *alpha': AlphaClosure-global - k l' R l' by standard (rule finite)*

**lemma** [*simp*]:  $alpha'.cla = cla\ l'$  **unfolding**  $alpha'.cla-def\ cla-def ..$

**lemma** *regions-poststable1*:

**assumes**

$A \vdash \langle l, Z \rangle \rightsquigarrow^a \langle l', Z' \rangle\ Z \subseteq V\ R' \in \mathcal{R}\ l' R' \cap Z' \neq \{\}$

**shows**  $\exists R \in \mathcal{R}\ l. A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^a \langle l', R' \rangle \wedge R \cap Z \neq \{\}$

**using** *assms* **proof** (*induction A  $\equiv A\ l \equiv l - - l' \equiv l'$  -rule: step-trans-z.induct*)

**case**  $A$ : (*step-trans-t-z Z*)

**from**  $\langle R' \cap (Z^\uparrow \cap \{u. u \vdash inv-of\ A\ l\}) \neq \{\} \rangle$  **obtain**  $u\ d$  **where**  $u$ :

$u \in Z\ u \oplus d \in R'\ u \oplus d \vdash inv-of\ A\ l\ 0 \leq d$

**unfolding** *zone-delay-def* **by** *blast+*

**with**  $alpha.closure-subs[OF\ A(2)]$  **obtain**  $R$  **where**  $R1: u \in R\ R \in \mathcal{R}\ l$

**by** (*simp add: cla-def*) *blast*

**from**  $\langle Z \subseteq V \rangle \langle u \in Z \rangle$  **have**  $\forall x \in X. 0 \leq u\ x$  **unfolding** *V-def* **by** *fastforce*

**from** *region-cover'[OF this]* **have**  $R: [u]_l \in \mathcal{R}\ l\ u \in [u]_l$  **by** *auto*

**from** *SuccI2[OF R-def' this(2,1)  $\langle 0 \leq d \rangle HOL.refl$*   $u(2)$  **have**  $v'1:$

$[u \oplus d]_l \in Succ\ (\mathcal{R}\ l)\ ([u]_l)\ [u \oplus d]_l \in \mathcal{R}\ l$

**by** *auto*

**from**  $\text{alpha.regions-closed}'\text{-spec}[OF\ R(1,2)\ \langle 0 \leq d \rangle]$  **have**  $v'2: u \oplus d \in [u \oplus d]_l$  **by**  $\text{simp}$   
**from**  $\text{valid-abstraction}$  **have**  
 $\forall (x, m) \in \text{clkp-set}\ A\ l.\ m \leq \text{real}\ (k\ l\ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**by**  $(\text{auto elim!}: \text{valid-abstraction.cases})$   
**then have**  
 $\forall (x, m) \in \text{collect-clock-pairs}\ (\text{inv-of}\ A\ l).\ m \leq \text{real}\ (k\ l\ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding**  $\text{clkp-set-def}\ \text{collect-clki-def}\ \text{inv-of-def}$  **by**  $\text{fastforce}$   
**from**  $\text{ccompatible}[OF\ \text{this}, \text{folded}\ \mathcal{R}\text{-def}']\ v'1(2)\ v'2\ u(2,3)$  **have**  
 $[u \oplus d]_l \subseteq \{\!\{ \text{inv-of}\ A\ l \}\!\}$   
**unfolding**  $\text{ccompatible-def}\ \text{ccval-def}$  **by**  $\text{auto}$   
**from**  
 $\text{alpha.valid-regions-distinct-spec}[OF\ v'1(2)\ -\ v'2\ \langle u \oplus d \in R' \rangle\ \langle R' \in \rightarrow \langle l = l' \rangle]$   
 $\text{alpha.region-unique-spec}[OF\ R1]$   
**have**  $[u \oplus d]_l = R'\ [u]_l = R$  **by**  $\text{auto}$   
**from**  $\text{valid-abstraction}\ \langle R \in \rightarrow \langle - \in \text{Succ}\ (\mathcal{R}\ l) \rightarrow \langle - \subseteq \{\!\{ \text{inv-of}\ A\ l \}\!\} \rangle$  **have**  
 $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{\tau} \langle l, R' \rangle$   
**by**  $(\text{auto simp}: \text{comp-def}\ \langle [u \oplus d]_l = R' \rangle \langle - = R \rangle)$   
**with**  $\langle l = l' \rangle\ \langle R \in \rightarrow \langle u \in R \rangle\ \langle u \in Z \rangle$  **show**  $?case$  **by**  $-$   $(\text{rule}\ \text{bexI}[\text{where}\ x = R]; \text{auto})$   
**next**  
**case**  $A: (\text{step-trans-a-z}\ g\ a\ r\ Z)$   
**from**  $A(4)$  **obtain**  $u\ v'$  **where**  
 $u \in Z$  **and**  $v': v' = [r \rightarrow 0]u\ u \vdash g\ v' \vdash \text{inv-of}\ A\ l'\ v' \in R'$   
**unfolding**  $\text{zone-set-def}$  **by**  $\text{blast}$   
**from**  $\langle u \in Z \rangle\ \text{alpha.closure-subs}[OF\ A(2)]\ A(1)$  **obtain**  $u'\ R$  **where**  $u':$   
 $u \in R\ u' \in R\ R \in \mathcal{R}\ l$   
**by**  $(\text{simp add}: \text{cla-def})\ \text{blast}$   
**then have**  $\forall x \in X.\ 0 \leq u\ x$  **unfolding**  $\mathcal{R}\text{-def}$  **by**  $\text{fastforce}$   
**from**  $\text{region-cover}'[OF\ \text{this}]$  **have**  $[u]_l \in \mathcal{R}\ l\ u \in [u]_l$  **by**  $\text{auto}$   
**have**  $*$ :  
 $[u]_l \subseteq \{\!\{ g \}\!\}\ \text{region-set}'\ ([u]_l)\ r\ 0 \subseteq [[r \rightarrow 0]u]_l'$   
 $[[r \rightarrow 0]u]_l' \in \mathcal{R}\ l'\ [[r \rightarrow 0]u]_l' \subseteq \{\!\{ \text{inv-of}\ A\ l' \}\!\}$   
**proof**  $-$   
**from**  $\text{valid-abstraction}$  **have**  $\text{collect-clkvt}\ (\text{trans-of}\ A) \subseteq X$   
 $\forall l\ g\ a\ r\ l'\ c.\ A \vdash l \xrightarrow{g,a,r} l' \wedge c \notin \text{set}\ r \xrightarrow{} k\ l'\ c \leq k\ l\ c$   
**by**  $(\text{auto elim}: \text{valid-abstraction.cases})$   
**with**  $A(1)$  **have**  $\text{set}\ r \subseteq X\ \forall y.\ y \notin \text{set}\ r \xrightarrow{} k\ l'\ y \leq k\ l\ y$   
**unfolding**  $\text{collect-clkvt-def}$  **by**  $(\text{auto}\ 4\ 8)$   
**with**  
 $\text{region-set-subs}[$   
 $\text{of}\ -\ X\ k\ l\ -\ 0,$  **where**  $k' = k\ l',$   $\text{folded}\ \mathcal{R}\text{-def},$   $OF\ \langle [u]_l \in \mathcal{R}\ l \rangle\ \langle u \in$

$[u]_l \rangle$  *finite*  
 $\quad ]$   
**show** *region-set'*  $([u]_l) \ r \ 0 \subseteq [[r \rightarrow 0]u]_{l'} \ [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l'$  **by** *auto*  
**from** *valid-abstraction* **have**  $*$ :  
 $\forall l. \forall (x, m) \in \text{clkp-set } A \ l. \ m \leq \text{real } (k \ l \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**by** (*fastforce elim: valid-abstraction.cases*)  
**with**  $A(1)$  **have**  $\forall (x, m) \in \text{collect-clock-pairs } g. \ m \leq \text{real } (k \ l \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *clkp-set-def collect-clkt-def* **by** *fastforce*  
**from**  $\langle u \in [u]_l \rangle \langle [u]_l \in \mathcal{R} \ l \rangle$  *ccompatible[OF this, folded  $\mathcal{R}$ -def]*  $\langle u \vdash g \rangle$   
**show**  $[u]_l \subseteq \{g\}$   
**unfolding** *ccompatible-def cval-def* **by** *blast*  
**have**  $**$ :  $[r \rightarrow 0]u \in [[r \rightarrow 0]u]_{l'}$   
**using**  $\langle R' \in \mathcal{R} \ l' \rangle \langle v' \in R' \rangle$  *alpha'.region-unique-spec v'(1)* **by** *blast*  
**from**  $*$  **have**  
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l'). \ m \leq \text{real } (k \ l' \ x) \wedge x \in X \wedge m \in \mathbb{N}$   
**unfolding** *inv-of-def clkp-set-def collect-clki-def* **by** *fastforce*  
**from**  $**$   $\langle [[r \rightarrow 0]u]_{l'} \in \mathcal{R} \ l' \rangle$  *ccompatible[OF this, folded  $\mathcal{R}$ -def]*  $\langle v' \vdash - \rangle$   
**show**  
 $[[r \rightarrow 0]u]_{l'} \subseteq \{\text{inv-of } A \ l'\}$   
**unfolding** *ccompatible-def cval-def*  $\langle v' = - \rangle$  **by** *blast*  
**qed**  
**from**  $*$   $\langle v' = - \rangle \langle u \in [u]_l \rangle$  **have**  $v' \in [[r \rightarrow 0]u]_{l'}$  **unfolding** *region-set'-def*  
**by** *auto*  
**from** *alpha'.valid-regions-distinct-spec[OF \*(3)  $\langle R' \in \mathcal{R} \ l' \rangle \langle v' \in [[r \rightarrow 0]u]_{l'} \rangle \langle v' \in R' \rangle$*   
**have**  $[[r \rightarrow 0]u]_{l'} = R'$ .  
**from** *alpha.region-unique-spec[OF u'(1,3)]* **have**  $[u]_l = R$  **by** *auto*  
**from** *A valid-abstraction*  $\langle R \in - \rangle *$  **have**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a^{(g, a, r)} \langle l', R' \rangle$   
**by** (*auto simp: comp-def*  $\langle - = R' \rangle \langle - = R \rangle$ )  
**with**  $\langle R \in - \rangle \langle u \in R \rangle \langle u \in Z \rangle$  **show**  $?case$  **by**  $-$  (*rule* *bestI* [**where**  $x = R$ ]; *auto*)  
**qed**

**lemma** *regions-poststable'*:

**assumes**

$A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \ Z \subseteq V \ R' \in \mathcal{R} \ l' \ R' \cap Z' \neq \{\}$

**shows**  $\exists R \in \mathcal{R} \ l. \ A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle \wedge R \cap Z \neq \{\}$

**using** *assms*

**by** (*cases a*)

(*auto dest!*: *regions-poststable1 dest: step-trans-r-step-r-action step-z-step-trans-z-action simp: step-trans-t-z-iff[symmetric]*)

)

end

**lemma** *regions-poststable2*:

**assumes** *valid-abstraction*: *valid-abstraction*  $A X k$

**and** *prems*:  $A \vdash' \langle l, Z \rangle \rightsquigarrow^a \langle l', Z' \rangle Z \subseteq V R' \in \mathcal{R} l' R' \cap Z' \neq \{\}$

**shows**  $\exists R \in \mathcal{R} l. A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^a \langle l', R' \rangle \wedge R \cap Z \neq \{\}$

**using** *prems*(1) **proof** (*cases*)

**case** *steps*: (*step-trans-z'*  $Z1$ )

**with** *prems* **have**  $Z1 \subseteq V$

**by** (*blast dest*: *step-trans-z-V*)

**from** *regions-poststable1*[*OF valid-abstraction steps*(2)  $\langle Z1 \subseteq V \rangle$  *prems*(3,4)]

**obtain**  $R1$  **where**  $R1$ :

$R1 \in \mathcal{R} l A, \mathcal{R} \vdash \langle l, R1 \rangle \rightsquigarrow^{1a} \langle l', R' \rangle R1 \cap Z1 \neq \{\}$

**by** *auto*

**from** *regions-poststable1*[*OF valid-abstraction steps*(1)  $\langle Z \subseteq V \rangle$   $R1(1,3)$ ]

**obtain**  $R$  **where**

$R \in \mathcal{R} l A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^\tau \langle l, R1 \rangle R \cap Z \neq \{\}$

**by** *auto*

**with**  $R1(2)$  **show** *?thesis*

**by** (*auto intro*: *step-trans-r'.intros*)

qed

Poststability of Closures: For every transition in the zone graph and each region in the closure of the resulting zone, there exists a similar transition in the region graph.

**lemma** *regions-poststable*:

**assumes** *valid-abstraction*: *valid-abstraction*  $A X k$

**and**  $A$ :

$A \vdash \langle l, Z \rangle \rightsquigarrow_\tau \langle l', Z' \rangle A \vdash \langle l', Z' \rangle \rightsquigarrow_{1a} \langle l'', Z'' \rangle$

$Z \subseteq V R'' \in \mathcal{R} l'' R'' \cap Z'' \neq \{\}$

**shows**  $\exists R \in \mathcal{R} l. A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l'', R'' \rangle \wedge R \cap Z \neq \{\}$

**proof** –

**from**  $A(1)$   $\langle Z \subseteq V \rangle$  **have**  $Z' \subseteq V$  **by** (*rule step-z-V*)

**from**  $A(1)$  **have** [*simp*]:  $l' = l$  **by** *auto*

**from** *regions-poststable'*[*OF valid-abstraction A*(2)  $\langle Z' \subseteq V \rangle \langle R'' \in \cdot \rangle \langle R'' \cap Z'' \neq \{\} \rangle$ ] **obtain**  $R'$

**where**  $R'$ :  $R' \in \mathcal{R} l' A, \mathcal{R} \vdash \langle l', R' \rangle \rightsquigarrow_{1a} \langle l'', R'' \rangle R' \cap Z' \neq \{\}$

**by** *auto*

**from** *regions-poststable'*[*OF valid-abstraction A*(1)  $\langle Z \subseteq V \rangle$   $R'(1,3)$ ] **obtain**  $R$  **where**

$R \in \mathcal{R} l A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_\tau \langle l, R' \rangle R \cap Z \neq \{\}$

**by** *auto*

**with**  $R'(2)$  **show** *?thesis* **by** – (*rule bexI*[**where**  $x = R$ ]; *auto intro*:

*step-r'.intros*)  
**qed**

**lemma** *step-t-r-loc*:  
 $l' = l$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_{\tau} \langle l', R' \rangle$   
**using** *that* **by** *cases auto*

**lemma**  $\mathcal{R}$ -*V*:  
 $u \in V$  **if**  $R \in \mathcal{R} \wedge u \in R$   
**using** *that* **unfolding**  $\mathcal{R}$ -*def* *V-def* **by** *auto*

**lemma** *step-r'-complete*:  
**assumes**  $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$  *valid-abstraction*  $A \ X \ (\lambda x. \text{real } o \ k \ x) \ u \in V$   
**shows**  $\exists a \ R'. \ u' \in R' \wedge A, \mathcal{R} \vdash \langle l, [u]_l \rangle \rightsquigarrow_a \langle l', R' \rangle$   
**using** *assms*  
**apply** *cases*  
**apply** (*drule* *step-t-r-complete*, (*rule* *assms*; *fail*), *simp* *add*: *V-def*)  
**apply** *clarify*  
**apply** (*frule* *step-a-r-complete*)  
**by** (*auto* *dest*: *step-t-r-loc* *simp*:  $\mathcal{R}$ -*def* *simp*: *region-unique intro!*: *step-r'.intros*)

**lemma** *step-r- $\mathcal{R}$* :  
 $R' \in \mathcal{R} \wedge l' = l$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$   
**using** *that* **by** (*auto* *elim*: *step-r.cases*)

**lemma** *step-r'- $\mathcal{R}$* :  
 $R' \in \mathcal{R} \wedge l' = l$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$   
**using** *that* **by** (*auto* *intro*: *step-r- $\mathcal{R}$*  *elim*: *step-r'.cases*)

**end**

**context** *Regions*  
**begin**

**lemma** *closure-parts-mono*:  
 $\{R \in \mathcal{R} \mid R \cap Z \neq \{\}\} \subseteq \{R \in \mathcal{R} \mid R \cap Z' \neq \{\}\}$  **if**  $\text{Closure}_{\alpha, l} \ Z \subseteq \text{Closure}_{\alpha, l} \ Z'$   
**proof** (*clarify*, *goal-cases*)  
**case** *prems*: ( $1 \ R$ )  
**with** *that* **have**  $R \subseteq \text{Closure}_{\alpha, l} \ Z'$   
**unfolding** *cla-def* **by** *auto*  
**from**  $\langle - \neq \{\} \rangle$  **obtain**  $u$  **where**  $u \in R \wedge u \in Z$  **by** *auto*  
**with**  $\langle R \subseteq - \rangle$  **obtain**  $R'$  **where**  $R' \in \mathcal{R} \wedge u \in R' \wedge R' \cap Z' \neq \{\}$  **unfolding**

*cla-def* **by force**

**from**  $\mathcal{R}$ -regions-distinct[OF  $\mathcal{R}$ -def' this(1,2)  $\langle R \in \cdot \rangle$ ]  $\langle u \in R \rangle$  **have**  $R = R'$  **by auto**  
**with**  $\langle R' \cap Z' \neq \{\} \rangle$   $\langle R \cap Z' = \{\} \rangle$  **show** ?case **by simp**  
**qed**

**lemma** *closure-parts-id*:

$\{R \in \mathcal{R} \mid R \cap Z \neq \{\}\} = \{R \in \mathcal{R} \mid R \cap Z' \neq \{\}\}$  **if**  
 $\text{Closure}_{\alpha,l} Z = \text{Closure}_{\alpha,l} Z'$   
**using** *closure-parts-mono* **that by blast**

**More lemmas on regions** **context**

**fixes**  $l' :: 's$

**begin**

**interpretation** *regions*: *Regions-global* - -  $k \ l'$

**by standard** (*rule finite clock-numbering not-in-X non-empty*)+

**context**

**fixes**  $A :: ('a, 'c, t, 's) \text{ta}$

**assumes** *valid-abstraction*: *valid-abstraction*  $A \ X \ k$

**begin**

**lemmas** *regions-poststable* = *regions-poststable*[OF *valid-abstraction*]

**lemma** *clkp-set-clkp-set1*:

$\exists l. (c, x) \in \text{clkp-set } A \ l$  **if**  $(c, x) \in \text{Timed-Automata.clkp-set } A$

**using** *that*

**unfolding** *Timed-Automata.clkp-set-def* *Closure.clkp-set-def*

**unfolding** *Timed-Automata.collect-clki-def* *Closure.collect-clki-def*

**unfolding** *Timed-Automata.collect-clkt-def* *Closure.collect-clkt-def*

**by fastforce**

**lemma** *clkp-set-clkp-set2*:

$(c, x) \in \text{Timed-Automata.clkp-set } A$  **if**  $(c, x) \in \text{clkp-set } A \ l$  **for**  $l$

**using** *that*

**unfolding** *Timed-Automata.clkp-set-def* *Closure.clkp-set-def*

**unfolding** *Timed-Automata.collect-clki-def* *Closure.collect-clki-def*

**unfolding** *Timed-Automata.collect-clkt-def* *Closure.collect-clkt-def*

**by fastforce**

**lemma** *clock-numbering-le*:  $\forall c \in \text{clk-set } A. v \ c \leq n$

**proof**

**fix**  $c$  **assume**  $c \in \text{clk-set } A$   
**then have**  $c \in X$   
**proof** (*safe, clarsimp, goal-cases*)  
  **case** (1  $x$ )  
  **then obtain**  $l$  **where**  $(c, x) \in \text{clkp-set } A$  **l by** (*auto dest: clkp-set-clkp-set1*)  
  **with** *valid-abstraction* **show**  $c \in X$  **by** (*auto elim!: valid-abstraction.cases*)  
**next**  
  **case** 2  
  **with** *valid-abstraction* **show**  $c \in X$  **by** (*auto elim!: valid-abstraction.cases*)  
**qed**  
**with** *clock-numbering* **show**  $v \ c \leq n$  **by** *auto*  
**qed**

**lemma** *beta-alpha-step:*

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha(a)} \langle l', \text{Closure}_{\alpha, l'} Z' \rangle$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta(a)} \langle l', Z' \rangle$   $Z \in V'$   
**proof** –  
**from** *that* **obtain**  $Z1'$  **where**  $Z1': Z' = \text{Approx}_{\beta} l' Z1'$   $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z1' \rangle$

**by** (*clarsimp elim!: step-z-beta.cases*)  
  **with**  $\langle Z \in V' \rangle$  **have**  $Z1' \in V'$   
  **using** *valid-abstraction clock-numbering-le* **by** (*auto intro: step-z-V'*)  
  **let**  $?alpha = \text{Closure}_{\alpha, l'} Z1'$  **and**  $?beta = \text{Closure}_{\alpha, l'} (\text{Approx}_{\beta} l' Z1')$   
  **have**  $?beta \subseteq ?alpha$   
  **using** *regions.approx-beta-closure-alpha'[OF  $\langle Z1' \in V' \rangle$  regions.alpha-interp.closure-involutive*  
  **by** (*auto 4 3 dest: regions.alpha-interp.cla-mono*)  
  **moreover have**  $?alpha \subseteq ?beta$   
  **by** (*intro regions.alpha-interp.cla-mono[simplified] regions.beta-interp.apx-subset*)  
  **ultimately have**  $?beta = ?alpha$  ..  
  **with**  $Z1'$  **show** *?thesis* **by** *auto*  
**qed**

**lemma** *beta-alpha-region-step:*

$\exists a. \exists R \in \mathcal{R}. l. R \cap Z \neq \{\}$   $\wedge A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$  **if**  
 $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle$   $Z \in V'$   $R' \in \mathcal{R}$   $l' R' \cap Z' \neq \{\}$

**proof** –

**from** *that(1)* **obtain**  $l''$   $a$   $Z''$  **where** *steps:*  
 $A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l'', Z'' \rangle$   $A \vdash \langle l'', Z'' \rangle \rightsquigarrow_{\beta(1a)} \langle l', Z' \rangle$   
  **unfolding** *step-z-beta'-def* **by** *metis*  
**with**  $\langle Z \in V' \rangle$  *steps(1)* **have**  $Z'' \in V'$   
  **using** *valid-abstraction clock-numbering-le* **by** (*blast intro: step-z-V'*)  
**from** *beta-alpha-step[OF steps(2) this]* **have**  $A \vdash \langle l'', Z'' \rangle \rightsquigarrow_{\alpha 1a} \langle l', \text{Closure}_{\alpha, l'}(Z') \rangle$  .  
**from** *step-z-alpha.cases[OF this]* **obtain**  $Z1$  **where**  $Z1:$

$A \vdash \langle l'', Z'' \rangle \rightsquigarrow_{1a} \langle l', Z1 \rangle \text{ Closure}_{\alpha, l'}(Z') = \text{Closure}_{\alpha, l'}(Z1)$   
 by *metis*  
**from** *closure-parts-id*[*OF this(2)*] *that(3,4)* **have**  $R' \cap Z1 \neq \{\}$  **by** *blast*  
**from** *regions-poststable*[*OF steps(1) Z1(1) - \langle R' \in \cdot \rangle this*]  $\langle Z \in V' \rangle$  **show**  
*?thesis*  
 by (*auto dest: V'-V*)  
**qed**

**lemmas** *step-z-beta'-V' = step-z-beta'-V'*[*OF valid-abstraction clock-numbering-le*]

**lemma** *step-trans-z'-closure-subs*:

**assumes**  
 $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle \ Z \subseteq V \ \forall R \in \mathcal{R} \ l. R \cap Z \neq \{\} \longrightarrow R \cap W \neq \{\}$   
**shows**  
 $\exists W'. A \vdash' \langle l, W \rangle \rightsquigarrow^t \langle l', W' \rangle \wedge (\forall R \in \mathcal{R} \ l'. R \cap Z' \neq \{\} \longrightarrow R \cap W' \neq \{\})$   
**proof** –  
**from** *assms(1)* **obtain**  $W'$  **where** *step*:  $A \vdash' \langle l, W \rangle \rightsquigarrow^t \langle l', W' \rangle$   
 by (*auto elim!: step-trans-z.cases step-trans-z'.cases*)  
**have**  $R' \cap W' \neq \{\}$  **if**  $R' \in \mathcal{R} \ l' \ R' \cap Z' \neq \{\}$  **for**  $R'$   
**proof** –  
**from** *regions-poststable2*[*OF valid-abstraction assms(1) - that*]  $\langle Z \subseteq V \rangle$   
**obtain**  $R$  **where**  $R$ :  
 $R \in \mathcal{R} \ l \ A, \mathcal{R} \vdash' \langle l, R \rangle \rightsquigarrow^t \langle l', R' \rangle \ R \cap Z \neq \{\}$   
 by *auto*  
**with** *assms(3)* **obtain**  $u$  **where**  $u \in R \ u \in W$   
 by *auto*  
**with** *step-trans-r'-sound*[*OF R(2)*] **obtain**  $u'$  **where**  $u' \in R' \ A \vdash' \langle l, u \rangle \rightarrow^t \langle l', u' \rangle$   
 by *auto*  
**with** *step-trans-z'-exact*[*OF this(2) step \langle u \in W \rangle*] **show** *?thesis*  
 by *auto*  
**qed**  
**with** *step* **show** *?thesis*  
 by *auto*  
**qed**

**lemma** *step-trans-z'-closure-eq*:

**assumes**  
 $A \vdash' \langle l, Z \rangle \rightsquigarrow^t \langle l', Z' \rangle \ Z \subseteq V \ W \subseteq V \ \forall R \in \mathcal{R} \ l. R \cap Z \neq \{\} \longleftrightarrow R \cap W \neq \{\}$   
**shows**  
 $\exists W'. A \vdash' \langle l, W \rangle \rightsquigarrow^t \langle l', W' \rangle \wedge (\forall R \in \mathcal{R} \ l'. R \cap Z' \neq \{\} \longleftrightarrow R \cap W' \neq \{\})$

$W' \neq \{\}$   
**proof** –  
**from**  $assms(4)$  **have** \*:  
 $\forall R \in \mathcal{R} \ l. R \cap Z \neq \{\} \longrightarrow R \cap W \neq \{\} \ \forall R \in \mathcal{R} \ l. R \cap W \neq \{\}$   
 $\longrightarrow R \cap Z \neq \{\}$   
**by** *auto*  
**from**  $step-trans-z'-closure-subst[OF \ assms(1,2) \ *(1)]$  **obtain**  $W'$  **where**  
 $W'$ :  
 $A \vdash' \langle l, W \rangle \rightsquigarrow^t \langle l', W' \rangle \ (\forall R \in \mathcal{R} \ l'. R \cap Z' \neq \{\} \longrightarrow R \cap W' \neq \{\})$   
**by** *auto*  
**with**  $step-trans-z'-closure-subst[OF \ W'(1) \ \langle W \subseteq V \rangle \ *(2)] \ assms(1)$  **show**  
*?thesis*  
**by** (*fastforce dest: step-trans-z'-determ*)  
**qed**

**lemma**  $step-z'-closure-subst$ :  
**assumes**  
 $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \ Z \subseteq V \ \forall R \in \mathcal{R} \ l. R \cap Z \neq \{\} \longrightarrow R \cap W \neq \{\}$   
**shows**  
 $\exists W'. A \vdash \langle l, W \rangle \rightsquigarrow \langle l', W' \rangle \wedge (\forall R \in \mathcal{R} \ l'. R \cap Z' \neq \{\} \longrightarrow R \cap W' \neq \{\})$   
**using**  $assms(1)$   
**by** (*auto*  
*dest: step-trans-z'-step-z'*  
*dest!: step-z'-step-trans-z' step-trans-z'-closure-subst[OF - assms(2,3)]*  
*)*

**end**

**lemma**  $apx-finite$ :  
 $finite \ \{Approx_\beta \ l' \ Z \mid Z. Z \subseteq V\}$  (**is**  $finite \ ?S$ )  
**proof** –  
**have**  $finite \ regions.\mathcal{R}_\beta$   
**by** (*simp add: regions.beta-interp.finite- $\mathcal{R}$* )  
**then have**  $finite \ \{S. S \subseteq regions.\mathcal{R}_\beta\}$   
**by** *auto*  
**then have**  $finite \ \{\bigcup S \mid S. S \subseteq regions.\mathcal{R}_\beta\}$   
**by** *auto*  
**moreover have**  $?S \subseteq \{\bigcup S \mid S. S \subseteq regions.\mathcal{R}_\beta\}$   
**by** (*auto dest!: regions.beta-interp.apx-in*)  
**ultimately show** *?thesis* **by** (*rule finite-subset[rotated]*)  
**qed**

**lemmas**  $apx-subset = regions.beta-interp.apx-subset$

**lemma** *step-z-beta'-empty*:

$Z' = \{\}$  **if**  $A \vdash \langle l, \{\} \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle$

**using** *that*

**by** (*auto*

*elim!*: *step-z.cases*

*simp*: *step-z-beta'-def regions.beta-interp.apx-empty zone-delay-def zone-set-def*  
)

**end**

**lemma** *step-z-beta'-complete*:

**assumes**  $A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$   $u \in Z$   $Z \subseteq V$

**shows**  $\exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \wedge u' \in Z'$

**proof** –

**from** *assms(1)* **obtain**  $l''$   $u''$  *d a* **where** *steps*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l'', u'' \rangle$   $A \vdash \langle l'', u'' \rangle \rightarrow_a \langle l', u' \rangle$

**by** (*force elim!*: *step'.cases*)

**then obtain**  $Z''$  **where**

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\tau} \langle l'', Z'' \rangle$   $u'' \in Z''$

**by** (*meson*  $\langle u \in Z \rangle$  *step-t-z-complete*)

**moreover with** *steps(2)* **obtain**  $Z'$  **where**

$A \vdash \langle l'', Z'' \rangle \rightsquigarrow_{1a} \langle l', Z' \rangle$   $u' \in Z'$

**by** (*meson*  $\langle u'' \in Z'' \rangle$  *step-a-z-complete*)

**ultimately show** *?thesis*

**unfolding** *step-z-beta'-def* **using**  $\langle Z \subseteq V \rangle$  *apx-subset* **by** *blast*

**qed**

**end**

## 7.9.2 Instantiation of Double Simulation

### 7.9.3 Auxiliary Definitions

**definition** *state-set* ::  $( 'a, 'c, 'time, 's )$   $ta \Rightarrow 's$  *set* **where**

*state-set*  $A \equiv fst \text{ ` } (fst A) \cup (snd \circ snd \circ snd \circ snd) \text{ ` } (fst A)$

**lemma** *finite-trans-of-finite-state-set*:

*finite* (*state-set*  $A$ ) **if** *finite* (*trans-of*  $A$ )

**using** *that* **unfolding** *state-set-def trans-of-def* **by** *auto*

**lemma** *state-setI1*:

$l \in \text{state-set } A$  **if**  $A \vdash l \rightarrow^{g,a,r} l'$

**using** *that* **unfolding** *state-set-def trans-of-def image-def* **by** (*auto* 4 4)

**lemma** *state-setI2*:

$l' \in \text{state-set } A$  **if**  $A \vdash l \longrightarrow^{g,a,r} l'$

**using** *that unfolding state-set-def trans-of-def image-def* **by** (*auto 4 4*)

**lemma** (**in** *AlphaClosure*) *step-r'-state-set*:

$l' \in \text{state-set } A$  **if**  $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow_a \langle l', R' \rangle$

**using** *that by* (*blast intro: state-setI2 elim: step-r'.cases*)

**lemma** (**in** *Regions*) *step-z-beta'-state-set2*:

$l' \in \text{state-set } A$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle$

**using** *that unfolding step-z-beta'-def* **by** (*force simp: state-set-def trans-of-def*)

#### 7.9.4 Instantiation

**locale** *Regions-TA = Regions*  $X$  - -  $k$  **for**  $X :: 'c$  *set* **and**  $k :: 's \Rightarrow 'c \Rightarrow \text{nat} +$

**fixes**  $A :: ('a, 'c, t, 's)$  *ta*

**assumes** *valid-abstraction: valid-abstraction*  $A$   $X$   $k$

**and** *finite-state-set: finite* (*state-set*  $A$ )

**begin**

**no-notation** *Regions-Beta.part* ( $\langle [-] \rangle$  [*61,61*] *61*)

**notation** *part''* ( $\langle [-] \rangle$  [*61,61*] *61*)

**lemma** *step-z-beta'-state-set1*:

$l \in \text{state-set } A$  **if**  $A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle$

**using** *that unfolding step-z-beta'-def* **by** (*force simp: state-set-def trans-of-def*)

**sublocale** *sim: Double-Simulation-paired*

$\lambda (l, u) (l', u'). A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$  — Concrete step relation

$\lambda (l, Z) (l', Z'). \exists a. A, \mathcal{R} \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the first abstraction layer

$\lambda (l, R). l \in \text{state-set } A \wedge R \in \mathcal{R}$   $l$  — Valid states of the first abstraction layer

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the second abstraction layer

$\lambda (l, Z). l \in \text{state-set } A \wedge Z \in V' \wedge Z \neq \{\}$  — Valid states of the second abstraction layer

**proof** (*standard, goal-cases*)

**case** (*1 S T*)

**then show** *?case*

```

    by (auto dest!: step-r'-sound)
next
case prems: (2 R' l' Z' l Z)
from prems(3) have l ∈ state-set A
  by (blast intro: step-z-beta'-state-set1)
from prems show ?case
  unfolding Double-Simulation-paired-Defs.closure'-def
  by (blast dest: beta-alpha-region-step[OF valid-abstraction] step-z-beta'-state-set1)
next
case prems: (3 l R R')
then show ?case
  using R-regions-distinct[OF R-def'] by auto
next
case 4
have *: finite (R l) for l
  unfolding R-def by (intro finite-R finite)
have
  {(l, R). l ∈ state-set A ∧ R ∈ R l} = (⋃ l ∈ state-set A. ((λ R. (l, R))
  {R. R ∈ R l}))
  by auto
also have finite ...
  by (auto intro: finite-UN-I[OF finite-state-set] *)
finally show ?case by auto
next
case (5 l Z)
then show ?case
  apply safe
  subgoal for u
    using region-cover'[of u l] by (auto dest!: V'-V, auto simp: V-def)
  done
qed

```

**sublocale** *Graph-Defs*

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \wedge Z' \neq \{\}$  .

**lemmas**  $step-z-beta'-V' = step-z-beta'-V'$ [*OF valid-abstraction*]

**lemma** *step-r'-complete-spec*:

**assumes**  $A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle u \in V$

**shows**  $\exists a R'. u' \in R' \wedge A, \mathcal{R} \vdash \langle l, [u]_l \rangle \rightsquigarrow_a \langle l', R' \rangle$

**using** *assms valid-abstraction* **by** (*auto simp: comp-def V-def intro!: step-r'-complete*)

**end**

### 7.9.5 Büchi Runs

**locale** *Regions-TA-Start-State* = *Regions-TA* - - - - *A* **for**  $A :: ('a, 'c, t, 's) ta +$   
**fixes**  $l_0 :: 's$  **and**  $Z_0 :: ('c, t)$  *zone*  
**assumes** *start-state*:  $l_0 \in \text{state-set } A \ Z_0 \in V' \ Z_0 \neq \{\}$   
**begin**

**definition**  $a_0 = \text{from-}R \ l_0 \ Z_0$

**sublocale** *sim-complete'*: *Double-Simulation-Finite-Complete-paired*

$\lambda (l, u) (l', u'). A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$  — Concrete step relation

$\lambda (l, Z) (l', Z'). \exists a. A, \mathcal{R} \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the first abstraction layer

$\lambda (l, R). l \in \text{state-set } A \wedge R \in \mathcal{R} \ l$  — Valid states of the first abstraction

layer

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the second abstraction layer

$\lambda (l, Z). l \in \text{state-set } A \wedge Z \in V' \wedge Z \neq \{\}$  — Valid states of the second

abstraction layer

$l_0 \ Z_0$

**proof** (*standard, goal-cases*)

**case**  $(1 \ x \ y \ S)$

— Completeness

**then show** *?case*

**by** (*force dest: step-z-beta'-complete[rotated 2, OF V'-V]*)

**next**

**case**  $4$

— Finiteness

**have**  $*$ :  $Z \in V'$  **if**  $A \vdash \langle l_0, Z_0 \rangle \rightsquigarrow_{\beta*} \langle l, Z \rangle$  **for**  $l \ Z$

**using** *that start-state step-z-beta'-V'* **by** (*induction rule: rtranclp-induct2*)

*blast+*

**have**  $Z \in \{\text{Approx}_\beta \ l \ Z \mid Z. Z \subseteq V\} \vee (l, Z) = (l_0, Z_0)$

**if reaches**  $(l_0, Z_0) \ (l, Z)$  **for**  $l \ Z$

**using that proof** (*induction rule: rtranclp-induct2*)

**case** *refl*

**then show** *?case*

**by** *simp*

**next**

**case** *prems: (step l Z l' Z')*

**from** *prems(1)* **have**  $A \vdash \langle l_0, Z_0 \rangle \rightsquigarrow_{\beta*} \langle l, Z \rangle$

**by** *induction (auto intro: rtranclp-trans)*

**then have**  $Z \in V'$

**by** (*rule \**)  
**with prems** *show ?case*  
**unfolding** *step-z-beta'-def* **using** *start-state(2)* **by** (*auto 0 1 dest!:*  
*V'-V elim!: step-z-V*)  
**qed**  
**then have**  $\{(l, Z). \text{reaches } (l_0, Z_0) (l, Z) \wedge l \in \text{state-set } A \wedge Z \in V' \wedge Z \neq \{\}\}$   
 $\subseteq \{(l, Z) \mid l \ Z. l \in \text{state-set } A \wedge Z \in \{\text{Approx}_\beta \ l \ Z \mid Z. Z \subseteq V\}\} \cup \{(l_0, Z_0)\}$   
**by** *auto*  
**also have** *finite ... (is finite ?S)*  
**proof** –  
**have**  $?S = \{(l_0, Z_0)\} \cup \bigcup ((\lambda l. (\lambda Z. (l, Z)) \ ' \{\text{Approx}_\beta \ l \ Z \mid Z. Z \subseteq V\}) \ ' (\text{state-set } A))$   
**by** *blast*  
**also have** *finite ...*  
**by** (*blast intro: apx-finite finite-state-set*)  
**finally show** *?thesis .*  
**qed**  
**finally show** *?case*  
**by** *simp*  
**next**  
**case** *prems: (2 a a')*  
**then show** *?case*  
**by** (*auto intro: step-z-beta'-V' step-z-beta'-state-set2*)  
**next**  
**case** *3*  
**from** *start-state* **show** *?case* **unfolding** *a<sub>0</sub>-def* **by** (*auto simp: from-R-fst*)  
**qed**

**sublocale** *sim-complete-bisim'*: *Double-Simulation-Finite-Complete-Bisim-Cover-paired*

$\lambda (l, u) (l', u'). A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$  — Concrete step relation

$\lambda (l, Z) (l', Z'). \exists a. A, \mathcal{R} \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the first abstraction layer

$\lambda (l, R). l \in \text{state-set } A \wedge R \in \mathcal{R} \ l$  — Valid states of the first abstraction layer

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \wedge Z' \neq \{\}$

— Step relation for the second abstraction layer

$\lambda (l, Z). l \in \text{state-set } A \wedge Z \in V' \wedge Z \neq \{\}$  — Valid states of the second abstraction layer

$l_0 \ Z_0$

**proof** (*standard, goal-cases*)

**case** (*1 l x l' y S*)

**then show** *?case*

```

    apply clarify
    apply (drule step-r'-complete-spec, (auto intro:  $\mathcal{R}$ -V; fail))
    by (auto simp:  $\mathcal{R}$ -def region-unique)
next
case ( $\exists l S l' T$ )
then show ?case
  by (auto simp add: step-r'-state-set step-r'- $\mathcal{R}$ )
next
case prems: ( $\exists l Z u$ )
then show ?case
  using region-cover'[of u l] by (auto dest!: V'-V simp: V-def)+
qed

```

### 7.9.6 State Formulas

**context**

fixes  $P :: 's \Rightarrow bool$  — The state property we want to check

**begin**

**definition**  $\varphi = P \circ fst$

**State formulas are compatible with closures.**

**Runs satisfying a formula all the way long interpretation**  $G_\varphi$ :  
*Graph-Start-Defs*

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \wedge Z' \neq \{\} \wedge P l' (l_0, Z_0) .$

**theorem** *Alw-ev-mc1*:

$(\forall x_0 \in a_0. sim.sim.Alw-ev (Not \circ \varphi) x_0) \longleftrightarrow \neg (P l_0 \wedge (\exists a. G_\varphi.reachable a \wedge G_\varphi.reaches1 a a))$

using *sim-complete-bisim'.Alw-ev-mc1*

unfolding  $G_\varphi.reachable-def a_0-def sim-complete-bisim'.\psi-def \varphi-def$

by *auto*

**end**

### 7.9.7 Leads-To Properties

**context**

fixes  $P Q :: 's \Rightarrow bool$  — The state properties we want to check

**begin**

**definition**  $\psi = Q \circ fst$

**interpretation**  $G_\psi$ : *Graph-Defs*

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \wedge Z' \neq \{\} \wedge Q l'$ .

**theorem** *leadsto-mc1*:

$(\forall x_0 \in a_0. \text{sim.sim.leadsto } (\varphi P) (\text{Not} \circ \psi) x_0) \longleftrightarrow$   
 $(\nexists x. \text{reaches } (l_0, Z_0) x \wedge P (\text{fst } x) \wedge Q (\text{fst } x) \wedge (\exists a. G_\psi.\text{reaches } x a \wedge$   
 $G_\psi.\text{reaches1 } a a))$

**if**  $\forall x_0 \in a_0. \neg \text{sim.sim.deadlock } x_0$

**proof** –

**from that have** \*:  $\forall x_0 \in Z_0. \neg \text{sim.sim.deadlock } (l_0, x_0)$

**unfolding**  $a_0$ -def **by** *auto*

**show** *?thesis*

**using** *sim-complete-bisim'.leadsto-mc1*[*OF* \*, *symmetric*, *of P Q*]

**unfolding**  $\psi$ -def  $\varphi$ -def *sim-complete-bisim'.* $\varphi$ -def *sim-complete-bisim'.* $\psi$ -def  
 $a_0$ -def

**by** (*auto dest: from-R-D from-R-loc*)

**qed**

**end**

**lemma** *from-R-reaches*:

**assumes** *sim.sim.Steps.reaches* (*from-R*  $l_0$   $Z_0$ )  $b$

**obtains**  $l$   $Z$  **where**  $b = \text{from-R } l$   $Z$

**using** *assms* **by** *cases* (*fastforce simp: sim.A2'-def dest!: from-R-R-of*) $+$

**lemma** *ta-reaches-ex-iff*:

**assumes** *compatible*:

$\bigwedge l u u' R.$

$u \in R \implies u' \in R \implies R \in \mathcal{R} l \implies l \in \text{state-set } A \implies P (l, u) = P$   
 $(l, u')$

**shows**

$(\exists x_0 \in a_0. \exists l u. \text{sim.sim.reaches } x_0 (l, u) \wedge P (l, u)) \longleftrightarrow$

$(\exists l Z. \exists u \in Z. \text{reaches } (l_0, Z_0) (l, Z) \wedge P (l, u))$

**proof** –

**have** \*:  $(\exists x_0 \in a_0. \exists l u. \text{sim.sim.reaches } x_0 (l, u) \wedge P (l, u))$

$\longleftrightarrow (\exists y. \exists x_0 \in \text{from-R } l_0 Z_0. \text{sim.sim.reaches } x_0 y \wedge P y)$

**unfolding**  $a_0$ -def **by** *auto*

**show** *?thesis*

**unfolding** \*

**apply** (*subst sim-complete-bisim'.sim-reaches-equiv*)

**subgoal**

**by** (*simp add: start-state*)

**apply** (*subst sim-complete-bisim'.reaches-ex'[of P]*)

**unfolding**  $a_0$ -def

```

    apply clarsimp
  subgoal
    unfolding sim.P1'-def by (clarsimp simp: fst-simp) (metis R-ofI
compatible fst-conv)
    apply safe
    apply (rule from-R-reaches, assumption)
    using from-R-fst by (force intro: from-R-val)+
  qed

```

**lemma** *ta-reaches-all-iff*:

```

  assumes compatible:
     $\bigwedge l u u' R. u \in R \implies u' \in R \implies R \in \mathcal{R} \implies l \in \text{state-set } A \implies P(l, u) = P(l, u')$ 
  shows
     $(\forall x_0 \in a_0. \forall l u. \text{sim.sim.reaches } x_0 (l, u) \longrightarrow P(l, u)) \longleftrightarrow (\forall l Z. \text{reaches } (l_0, Z_0) (l, Z) \longrightarrow (\forall u \in Z. P(l, u)))$ 

```

**proof** –

```

  have *:  $(\forall x_0 \in a_0. \forall l u. \text{sim.sim.reaches } x_0 (l, u) \longrightarrow P(l, u)) \longleftrightarrow (\forall y. \forall x_0 \in \text{from-R } l_0 Z_0. \text{sim.sim.reaches } x_0 y \longrightarrow P y)$ 
    unfolding a0-def by auto
  show ?thesis
    unfolding *
    apply (subst sim-complete-bisim'.sim-reaches-equiv)
    subgoal
      by (simp add: start-state)
    apply (subst sim-complete-bisim'.reaches-all'[of P])
    unfolding a0-def
    apply clarsimp
    subgoal
      unfolding sim.P1'-def by (clarsimp simp: fst-simp) (metis R-ofI
compatible fst-conv)
      apply auto
      apply (rule from-R-reaches, assumption)
      using from-R-fst by (force intro: from-R-val)+
    qed
  qed

```

**end**

**end**

## 8 Forward Analysis with DBMs and Widening

**theory** *Normalized-Zone-Semantics*

**imports** *DBM-Zone-Semantics Approx-Beta Simulation-Graphs-TA*

**begin**

**hide-const** (open) *D*

**no-notation** *infinity* ( $\langle \infty \rangle$ )

**lemma** *rtranclp-backwards-invariant-iff*:

**assumes** *invariant*:  $\bigwedge y z. E^{**} x y \implies P z \implies E y z \implies P y$

**and** *E'*:  $E' = (\lambda x y. E x y \wedge P y)$

**shows**  $E^{**} x y \wedge P x \longleftrightarrow E^{**} x y \wedge P y$

**unfolding** *E'*

**by** (*safe; induction rule: rtranclp-induct; auto dest: invariant intro: rtranclp.intros(2)*)

**context** *Bisimulation-Invariant*

**begin**

**context**

**fixes**  $\varphi :: 'a \Rightarrow \text{bool}$  **and**  $\psi :: 'b \Rightarrow \text{bool}$

**assumes** *compatible*:  $a \sim b \implies PA a \implies PB b \implies \varphi a \longleftrightarrow \psi b$

**begin**

**lemma** *reaches-ex-iff*:

$(\exists b. A.\text{reaches } a b \wedge \varphi b) \longleftrightarrow (\exists b. B.\text{reaches } a' b \wedge \psi b)$  **if**  $a \sim a'$  *PA a PB a'*

**using that by** (*force simp: compatible equiv'-def dest: bisim.A-B-reaches bisim.B-A-reaches*)

**lemma** *reaches-all-iff*:

$(\forall b. A.\text{reaches } a b \longrightarrow \varphi b) \longleftrightarrow (\forall b. B.\text{reaches } a' b \longrightarrow \psi b)$  **if**  $a \sim a'$  *PA a PB a'*

**using that by** (*force simp: compatible equiv'-def dest: bisim.A-B-reaches bisim.B-A-reaches*)

**end**

**end**

**lemma** *step-z-dbm-delay-loc*:

$l' = l$  if  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,\tau} \langle l', D' \rangle$   
**using that by** (*auto elim!*: *step-z-dbm.cases*)

**lemma** *step-z-dbm-action-state-set1*:

$l \in \text{state-set } A$  if  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,1a} \langle l', D' \rangle$   
**using that by** (*auto elim!*: *step-z-dbm.cases intro: state-setI1*)

**lemma** *step-z-dbm-action-state-set2*:

$l' \in \text{state-set } A$  if  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,1a} \langle l', D' \rangle$   
**using that by** (*auto elim!*: *step-z-dbm.cases intro: state-setI2*)

**lemma** *step-delay-loc*:

$l' = l$  if  $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$   
**using that by** (*auto elim!*: *step-t.cases*)

**lemma** *step-a-state-set1*:

$l \in \text{state-set } A$  if  $A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle$   
**using that by** (*auto elim!*: *step-a.cases intro: state-setI1*)

**lemma** *step'-state-set1*:

$l \in \text{state-set } A$  if  $A \vdash' \langle l, u \rangle \rightarrow \langle l', u' \rangle$   
**using that by** (*auto elim!*: *step'.cases intro: step-a-state-set1 dest: step-delay-loc*)

## 8.1 DBM-based Semantics with Normalization

### 8.1.1 Single Step

**inductive** *step-z-norm* ::

$(\text{'a}, \text{'c}, t, \text{'s}) \text{ ta}$   
 $\Rightarrow \text{'s} \Rightarrow t \text{ DBM} \Rightarrow (\text{'s} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{'c} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{'a} \text{ action}$   
 $\Rightarrow \text{'s} \Rightarrow t \text{ DBM} \Rightarrow \text{bool}$   
 $(\langle - \vdash \langle -, - \rangle \rightsquigarrow_{-, -, -} \langle -, - \rangle \rangle [61, 61, 61, 61, 61, 61] \ 61)$

**where** *step-z-norm*:

$A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle \Longrightarrow A \vdash \langle l, D \rangle \rightsquigarrow_{k,v,n,a} \langle l', \text{norm } (FW \ D' \ n) \ (k \ l') \ n \rangle$

**inductive** *step-z-norm'* ::

$(\text{'a}, \text{'c}, t, \text{'s}) \text{ ta} \Rightarrow \text{'s} \Rightarrow t \text{ DBM} \Rightarrow (\text{'s} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{'c} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{'s} \Rightarrow t \text{ DBM} \Rightarrow \text{bool}$   
 $(\langle - \vdash'' \langle -, - \rangle \rightsquigarrow_{-, -, -} \langle -, - \rangle \rangle [61, 61, 61, 61, 61] \ 61)$

**where**

*step*:  $A \vdash \langle l', Z' \rangle \rightsquigarrow_{v,n,\tau} \langle l'', Z'' \rangle$   
 $\Longrightarrow A \vdash \langle l'', Z'' \rangle \rightsquigarrow_{k,v,n,1(a)} \langle l''', Z''' \rangle$

$$\Longrightarrow A \vdash' \langle l', Z \rangle \rightsquigarrow_{k,v,n} \langle l''', Z''' \rangle$$

**abbreviation** *steps-z-norm* ::

$(\text{'a, 'c, t, 's}) \text{ta} \Rightarrow \text{'s} \Rightarrow \text{t DBM} \Rightarrow (\text{'s} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{'c} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{'s} \Rightarrow \text{t DBM} \Rightarrow \text{bool}$

$(\langle \vdash \langle -, - \rangle \rightsquigarrow_{-, -, *} \langle -, - \rangle \rangle [61, 61, 61, 61, 61] 61)$  **where**

$A \vdash \langle l, D \rangle \rightsquigarrow_{k,v,n} \langle l', D' \rangle \equiv (\lambda (l, Z) (l', Z'). A \vdash' \langle l, Z \rangle \rightsquigarrow_{k,v,n} \langle l', Z' \rangle)^{**}$   
 $(l, D) (l', D')$

**lemma** *norm-empty-diag-preservation-real*:

**fixes**  $k :: \text{nat} \Rightarrow \text{nat}$

**assumes**  $i \leq n$

**assumes**  $M \ i \ i < \text{Le } 0$

**shows**  $\text{norm } M \ (\text{real } o \ k) \ n \ i \ i < \text{Le } 0$

**using** *assms* **unfolding** *norm-def* **by** (*auto simp: Let-def norm-diag-def DBM.less*)

**context** *Regions-defs*

**begin**

**inductive** *valid-dbm* **where**

$[M]_{v,n} \subseteq V \Longrightarrow \text{dbm-int } M \ n \Longrightarrow \text{valid-dbm } M$

**inductive-cases** *valid-dbm-cases*[*elim*]: *valid-dbm*  $M$

**declare** *valid-dbm.intros*[*intro*]

**end**

**locale** *Regions-common* =

*Regions-defs*  $X \ v \ n$  **for**  $X :: \text{'c set}$  **and**  $v \ n +$

**fixes** *not-in-X*

**assumes** *finite*: *finite*  $X$

**assumes** *clock-numbering*: *clock-numbering'*  $v \ n \ \forall k \leq n. \ k > 0 \longrightarrow (\exists c \in X. \ v \ c = k)$

$\forall c \in X. \ v \ c \leq n$

**assumes** *not-in-X*: *not-in-X*  $\notin X$

**assumes** *non-empty*:  $X \neq \{\}$

**begin**

**lemma** *FW-zone-equiv-spec*:

**shows**  $[M]_{v,n} = [\text{FW } M \ n]_{v,n}$

**apply** (*rule FW-zone-equiv*) **using** *clock-numbering(2)* **by** *auto*

**lemma** *dbm-non-empty-diag*:

**assumes**  $[M]_{v,n} \neq \{\}$   
**shows**  $\forall k \leq n. M k k \geq 0$

**proof** *safe*

**fix**  $k$  **assume**  $k: k \leq n$

**have**  $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k)$  **using** *clock-numbering(2)* **by** *blast*  
**from**  $k$  *not-empty-cyc-free[OF this assms(1)]* **show**  $0 \leq M k k$  **by** (*simp*  
*add: cyc-free-diag-dest'*)

**qed**

**lemma** *cn-weak*:  $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k)$  **using** *clock-numbering(2)*  
**by** *blast*

**lemma** *negative-diag-empty*:

**assumes**  $\exists k \leq n. M k k < 0$   
**shows**  $[M]_{v,n} = \{\}$

**using** *dbm-non-empty-diag assms* **by** *force*

**lemma** *non-empty-cyc-free*:

**assumes**  $[M]_{v,n} \neq \{\}$   
**shows** *cyc-free*  $M n$

**using** *FW-neg-cycle-detect FW-zone-equiv-spec assms negative-diag-empty*  
**by** *blast*

**lemma** *FW-valid-preservation*:

**assumes** *valid-dbm*  $M$   
**shows** *valid-dbm*  $(FW M n)$

**proof** *standard*

**from** *FW-int-preservation assms* **show** *dbm-int*  $(FW M n) n$  **by** *blast*

**next**

**from** *FW-zone-equiv-spec[of M, folded neutral]* *assms* **show**  $[FW M n]_{v,n}$   
 $\subseteq V$  **by** *fastforce*

**qed**

**end**

**context** *Regions-global*

**begin**

**sublocale** *Regions-common* **by** *standard (rule finite clock-numbering not-in-X*  
*non-empty)+*

**abbreviation**  $v' \equiv \text{beta-interp.v}'$

**lemma** *apx-empty-iff''*:

**assumes** *canonical*  $M1\ n\ [M1]_{v,n} \subseteq V\ dbm-int\ M1\ n$

**shows**  $[M1]_{v,n} = \{\} \longleftrightarrow [norm\ M1\ (k\ o\ v')\ n]_{v,n} = \{\}$

**using** *beta-interp.apx-norm-eq[OF assms]* *apx-empty-iff'*[of  $[M1]_{v,n}$ ] *assms*

**unfolding** *V'-def* **by** *blast*

**lemma** *norm-FW-empty*:

**assumes** *valid-dbm*  $M$

**assumes**  $[M]_{v,n} = \{\}$

**shows**  $[norm\ (FW\ M\ n)\ (k\ o\ v')\ n]_{v,n} = \{\}$  (**is**  $[?M]_{v,n} = \{\}$ )

**proof** –

**from** *assms(2)* *cyc-free-not-empty* *clock-numbering(1)* **have**  $\neg\ cyc-free\ M\ n$

**by** *metis*

**from** *FW-neg-cycle-detect[OF this]* **obtain**  $i$  **where**  $i: i \leq n\ FW\ M\ n\ i$   
 $i < 0$  **by** *auto*

**with** *norm-empty-diag-preservation-real[folded neutral]* **have**

$?M\ i\ i < 0$

**unfolding** *comp-def* **by** *auto*

**with**  $\langle i \leq n \rangle$  **show** *?thesis* **using** *beta-interp.neg-diag-empty-spec* **by** *auto*  
**qed**

**lemma** *apx-norm-eq-spec*:

**assumes** *valid-dbm*  $M$

**and**  $[M]_{v,n} \neq \{\}$

**shows** *beta-interp.Approx <sub>$\beta$</sub>*   $([M]_{v,n}) = [norm\ (FW\ M\ n)\ (k\ o\ v')\ n]_{v,n}$

**proof** –

**note** *cyc-free = non-empty-cyc-free[OF assms(2)]*

**from** *assms(1)* *FW-zone-equiv-spec[of M]* **have**  $[M]_{v,n} = [FW\ M\ n]_{v,n}$

**by** (*auto simp: neutral*)

**with** *beta-interp.apx-norm-eq[OF fw-canonical[OF cyc-free] - FW-int-preservation]*  
*dbm-non-empty-diag[OF assms(2)]* *assms(1)*

**show** *Approx <sub>$\beta$</sub>*   $([M]_{v,n}) = [norm\ (FW\ M\ n)\ (k\ o\ v')\ n]_{v,n}$  **by** *auto*  
**qed**

**lemma** *norm-FW-valid-preservation-non-empty*:

**assumes** *valid-dbm*  $M$   $[M]_{v,n} \neq \{\}$

**shows** *valid-dbm*  $(norm\ (FW\ M\ n)\ (k\ o\ v')\ n)$  (**is** *valid-dbm*  $?M$ )

**proof** –

**from** *FW-valid-preservation[OF assms(1)]* **have** *valid: valid-dbm*  $(FW\ M\ n)$  .

**show** *?thesis*

**proof** *standard*

**from** *valid beta-interp.norm-int-preservation* **show** *dbm-int*  $?M\ n$  **by**

*blast*  
**next**  
**from** *fw-canonical*[*OF non-empty-cyc-free*] *assms* **have** *canonical* (*FW M n*) *n* **by** *auto*  
**from** *beta-interp.norm-V-preservation*[*OF - this*] *valid* **show**  $[?M]_{v,n} \subseteq V$  **by** *fast*  
**qed**  
**qed**

**lemma** *norm-int-all-preservation*:  
**fixes**  $M :: \text{real DBM}$   
**assumes** *dbm-int-all*  $M$   
**shows** *dbm-int-all* (*norm M (k o v')*  $n$ )  
**using** *assms* **unfolding** *norm-def norm-diag-def* **by** (*auto simp: Let-def*)

**lemma** *norm-FW-valid-preservation-empty*:  
**assumes** *valid-dbm*  $M$   $[M]_{v,n} = \{\}$   
**shows** *valid-dbm* (*norm (FW M n) (k o v')*  $n$ ) (**is** *valid-dbm*  $?M$ )  
**proof** –  
**from** *FW-valid-preservation*[*OF assms(1)*] **have** *valid: valid-dbm* (*FW M n*) .  
**show** *?thesis*  
**proof** *standard*  
**from** *valid beta-interp.norm-int-preservation* **show** *dbm-int*  $?M$   $n$  **by**  
*blast*  
**next**  
**from** *norm-FW-empty*[*OF assms(1,2)*] **show**  $[?M]_{v,n} \subseteq V$  **by** *fast*  
**qed**  
**qed**

**lemma** *norm-FW-valid-preservation*:  
**assumes** *valid-dbm*  $M$   
**shows** *valid-dbm* (*norm (FW M n) (k o v')*  $n$ )  
**using** *assms norm-FW-valid-preservation-empty norm-FW-valid-preservation-non-empty*  
**by** *metis*

**lemma** *norm-FW-equiv*:  
**assumes** *valid: dbm-int*  $D$   $n$  *dbm-int*  $M$   $n$   $[D]_{v,n} \subseteq V$   
**and** *equiv*:  $[D]_{v,n} = [M]_{v,n}$   
**shows**  $[norm (FW D n) (k o v') n]_{v,n} = [norm (FW M n) (k o v') n]_{v,n}$   
**proof** (*cases*  $[D]_{v,n} = \{\}$ )  
**case** *False*  
**with** *equiv fw-shortest*[*OF non-empty-cyc-free*] *FW-zone-equiv-spec* **have**  
*canonical (FW D n) n canonical (FW M n) n*  $[FW D n]_{v,n} = [D]_{v,n}$

$[FW M n]_{v,n} = [M]_{v,n}$   
**by** *blast+*  
**with** *valid equiv* **show** *?thesis*  
**apply** –  
**apply** (*subst beta-interp.apx-norm-eq[symmetric]*)  
**prefer** 4  
**apply** (*subst beta-interp.apx-norm-eq[symmetric]*)  
**by** (*simp add: FW-int-preservation*)+  
**next**  
**case** *True*  
**show** *?thesis*  
**apply** (*subst norm-FW-empty*)  
**prefer** 3  
**apply** (*subst norm-FW-empty*)  
**using** *valid equiv True* **by** *blast+*  
**qed**  
**end**

**context** *Regions*  
**begin**

**sublocale** *Regions-common* **by** *standard (rule finite clock-numbering not-in-X non-empty)*+

**definition**  $v' \equiv \lambda i. \text{if } 0 < i \wedge i \leq n \text{ then } (THE c. c \in X \wedge v c = i) \text{ else not-in-}X$

**abbreviation**  $\text{step-z-norm}' (\langle \cdot \vdash \langle \cdot, \cdot \rangle \rightsquigarrow_{\mathcal{N}(\cdot)} \langle \cdot, \cdot \rangle) [61,61,61,61] 61$

**where**

$$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle \equiv A \vdash \langle l, D \rangle \rightsquigarrow_{(\lambda l. k l o v'), v, n, a} \langle l', D' \rangle$$

**definition**  $\text{step-z-norm}'' (\langle \cdot \vdash'' \langle \cdot, \cdot \rangle \rightsquigarrow_{\mathcal{N}(\cdot)} \langle \cdot, \cdot \rangle) [61,61,61,61] 61$

**where**

$$A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l'', D'' \rangle \equiv \\ \exists l' D'. A \vdash \langle l, D \rangle \rightsquigarrow_{v, n, \tau} \langle l', D' \rangle \wedge A \vdash \langle l', D' \rangle \rightsquigarrow_{\mathcal{N}(1a)} \langle l'', D'' \rangle$$

**abbreviation**  $\text{steps-z-norm}' (\langle \cdot \vdash \langle \cdot, \cdot \rangle \rightsquigarrow_{\mathcal{N}^*} \langle \cdot, \cdot \rangle) [61,61,61] 61$

**where**

$$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \equiv (\lambda (l,D) (l',D'). \exists a. A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle) ** (l,D) (l',D')$$

**inductive-cases** *step-z-norm'-elims*[*elim!*]:  $A \vdash \langle l, u \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', u' \rangle$

**declare** *step-z-norm.intros*[*intro*]

**lemma** *step-z-valid-dbm*:

**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$

**and** *global-clock-numbering*  $A \ v \ n \ \text{valid-abstraction} \ A \ X \ k \ \text{valid-dbm} \ D$

**shows** *valid-dbm*  $D'$

**proof** –

**from** *step-z-V step-z-dbm-sound*[*OF assms(1,2)*] *step-z-dbm-preserves-int*[*OF assms(1,2)*]

*assms(3,4)*

**have**

*dbm-int*  $D' \ n \ A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', [D']_{v,n} \rangle$

**by** (*fastforce dest!*: *valid-abstraction-pairsD*)+

**with** *step-z-V*[*OF this(2)*] *assms(4)* **show** *?thesis* **by** *auto*

**qed**

**lemma** *step-z-norm-induct*[*case-names - step-z-norm step-z-refl*]:

**assumes**  $x1 \vdash \langle x2, x3 \rangle \rightsquigarrow_{(\lambda l. k \ l \ o \ v'),v,n,a} \langle x7, x8 \rangle$

**and** *step-z-norm*:

$\bigwedge A \ l \ D \ l' \ D'.$

$A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle \implies$

$P \ A \ l \ D \ l' \ (\text{norm} \ (FW \ D' \ n) \ (k \ l' \ o \ v') \ n)$

**shows**  $P \ x1 \ x2 \ x3 \ x7 \ x8$

**using** *assms* **by** (*induction rule: step-z-norm.inducts*) *auto*

**context**

**fixes**  $l' :: 's$

**begin**

**interpretation** *regions*: *Regions-global - - k l'*

**by** *standard* (*rule finite clock-numbering not-in-X non-empty*)+

**lemma** *regions-v'-eq*[*simp*]:

*regions.v' = v'*

**unfolding** *v'-def regions.beta-interp.v'-def* **by** *simp*

**lemma** *step-z-norm-int-all-preservation*:

**assumes**

$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *global-clock-numbering*  $A \ v \ n$

$\forall (x, m) \in \text{Timed-Automata.clkp-set} \ A. \ m \in \mathbb{N} \ \text{dbm-int-all} \ D$

**shows** *dbm-int-all*  $D'$

**using** *assms*  
**apply** *cases*  
**apply** *simp*  
**apply** (*rule regions.norm-int-all-preservation[simplified]*)  
**apply** (*rule FW-int-all-preservation*)  
**apply** (*erule step-z-dbm-preserves-int-all*)  
**by** *fast+*

**lemma** *step-z-norm-valid-dbm-preservation:*

**assumes**  
 $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *global-clock-numbering*  $A$   $v$   $n$  *valid-abstraction*  
 $A$   $X$   $k$  *valid-dbm*  $D$   
**shows** *valid-dbm*  $D'$   
**using** *assms*  
**by** *cases (simp; rule regions.norm-FW-valid-preservation[simplified]; erule step-z-valid-dbm; fast)*

**lemma** *norm-beta-sound:*

**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *global-clock-numbering*  $A$   $v$   $n$  *valid-abstraction*  
 $A$   $X$   $k$   
**and** *valid-dbm*  $D$   
**shows**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta(a)} \langle l', [D']_{v,n} \rangle$  **using** *assms(2-)*  
**apply** (*induction*  $A$   $l$   $D$   $l' \equiv l'$   $D'$  *rule: step-z-norm-induct, (subst assms(1); blast)*)  
**proof** *goal-cases*  
**case** *step-z-norm: (1 A l D D')*  
**from** *step-z-dbm-sound[OF step-z-norm(1,2)]* **have**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', [D']_{v,n} \rangle$  **by** *blast*  
**then have**  $*$ :  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta(a)} \langle l', \text{Approx}_{\beta} l' ([D']_{v,n}) \rangle$  **by** *force*  
**show** *?case*  
**proof** (*cases*  $[D']_{v,n} = \{\}$ )  
**case** *False*  
**from** *regions.apx-norm-eq-spec[OF step-z-valid-dbm[OF step-z-norm] False] \**  
**show** *?thesis* **by** *auto*  
**next**  
**case** *True*  
**with**  
 $\text{regions.norm-FW-empty}[OF \text{step-z-valid-dbm}[OF \text{step-z-norm}] \text{this}]$   
 $\text{regions.beta-interp.apx-empty} *$   
**show** *?thesis* **by** *auto*  
**qed**  
**qed**

**lemma** *step-z-norm-valid-dbm*:

**assumes**

$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *global-clock-numbering*  $A \ v \ n$   
*valid-abstraction*  $A \ X \ k$  *valid-dbm*  $D$

**shows** *valid-dbm*  $D'$  **using** *assms(2-)*

**apply** (*induction*  $A \ l \ D \ l' \equiv l' \ D'$  *rule*: *step-z-norm-induct*, (*subst* *assms(1)*;  
*blast*))

**proof** *goal-cases*

**case** *step-z-norm*: ( $1 \ A \ l \ D \ D'$ )

**with** *regions.norm-FW-valid-preservation*[*OF* *step-z-valid-dbm*[*OF* *step-z-norm*]]

**show** *?case* **by** *auto*

**qed**

**lemma** *norm-beta-complete*:

**assumes**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta(a)} \langle l', Z \rangle$  *global-clock-numbering*  $A \ v \ n$  *valid-abstraction*  
 $A \ X \ k$

**and** *valid-dbm*  $D$

**obtains**  $D'$  **where**  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$   $[D']_{v,n} = Z$  *valid-dbm*  $D'$

**proof** –

**from** *assms(3)* **have** *ta-int*:  $\forall (x, m) \in \text{Timed-Automata.clkp-set} \ A. \ m \in \mathbb{N}$

**by** (*fastforce* *dest!*: *valid-abstraction-pairsD*)

**from** *assms(1)* **obtain**  $Z'$  **where**  $Z'$ :  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_a \langle l', Z' \rangle$   $Z = \text{Approx}_\beta \ l' \ Z'$  **by** *auto*

**from** *assms(4)* **have** *dbm-int*  $D \ n$  **by** *auto*

**with** *step-z-dbm-DBM*[*OF*  $Z'(1)$  *assms(2)*] *step-z-dbm-preserves-int*[*OF*  
– *assms(2)* *ta-int*] **obtain**  $D'$

**where**  $D'$ :  $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,a} \langle l', D' \rangle$   $Z' = [D']_{v,n}$  *dbm-int*  $D' \ n$

**by** *auto*

**note** *valid-D'* = *step-z-valid-dbm*[*OF*  $D'(1)$  *assms(2,3)*]

**obtain**  $D''$  **where**  $D''$ :  $D'' = \text{norm} \ (FW \ D' \ n) \ (k \ l' \circ \ v')$   $n$  **by** *auto*

**show** *?thesis*

**proof** (*cases*  $Z' = \{\}$ )

**case** *False*

**with**  $D'$  **have**  $*$ :  $[D']_{v,n} \neq \{\}$  **by** *auto*

**from** *regions.apx-norm-eq-spec*[*OF* *valid-D'* *this*]  $D'' \ D'(2) \ Z'(2)$  *assms(4)*

**have**  $Z = [D'']_{v,n}$

**by** *auto*

**with** *regions.norm-FW-valid-preservation*[*OF* *valid-D'*]  $D' \ D'' \ *$  *assms(4)*

**show** *thesis*

**apply** –

**apply** (*rule* *that*[*of*  $D''$ ])

**by** (*drule* *step-z-norm.intros*[**where**  $k = \lambda \ l. \ k \ l \circ \ v'$ ]) *simp+*

**next**  
**case** *True*  
**with** *regions.norm-FW-empty*[*OF valid-D'*[*OF assms(4)*]] *D'' D' Z'(2)*  
*regions.norm-FW-valid-preservation*[*OF valid-D'*[*OF assms(4)*]] *re-*  
*gions.beta-interp.apx-empty*  
**show** *thesis*  
**apply** –  
**apply** (*rule that*[*of D''*])  
**apply** *blast*  
**by** *fastforce+*  
**qed**  
**qed**

**lemma** *step-z-norm-mono*:

**assumes**  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *global-clock-numbering A v n valid-abstraction*  
*A X k*  
**and** *valid-dbm D valid-dbm M*  
**and**  $[D]_{v,n} \subseteq [M]_{v,n}$   
**shows**  $\exists M'. A \vdash \langle l, M \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', M' \rangle \wedge [D']_{v,n} \subseteq [M']_{v,n}$   
**proof** –  
**from** *norm-beta-sound*[*OF assms(1,2,3,4)*] **have**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta(a)}$   
 $\langle l', [D']_{v,n} \rangle$  .  
**from** *step-z-beta-mono*[*OF this assms(6)*] *assms(5)* **obtain** *Z* **where**  
 $A \vdash \langle l, [M]_{v,n} \rangle \rightsquigarrow_{\beta(a)} \langle l', Z \rangle$   $[D']_{v,n} \subseteq Z$   
**by** *auto*  
**with** *norm-beta-complete*[*OF this(1) assms(2,3,5)*] **show** *?thesis* **by** *metis*  
**qed**

**lemma** *step-z-norm-equiv*:

**assumes** *step*:  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$   
**and** *prems*: *global-clock-numbering A v n valid-abstraction A X k*  
**and** *valid*: *valid-dbm D valid-dbm M*  
**and** *equiv*:  $[D]_{v,n} = [M]_{v,n}$   
**shows**  $\exists M'. A \vdash \langle l, M \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', M' \rangle \wedge [D']_{v,n} = [M']_{v,n}$   
**using** *step*  
**apply** *cases*  
**apply** (*frule step-z-dbm-equiv*[*OF prems(1)*])  
**apply** (*rule equiv*)  
**apply** *clarify*  
**apply** (*drule regions.norm-FW-equiv*[*rotated 3*])  
**prefer** 4  
**apply** *force*

**using** *step-z-valid-dbm*[*OF - prems*] **valid by** (*simp add: valid-dbm.simps*)+  
**end**

### 8.1.2 Multi Step

**lemma** *valid-dbm-V'*:  
**assumes** *valid-dbm M*  
**shows**  $[M]_{v,n} \in V'$   
**using** *assms unfolding V'-def* **by force**

**lemma** *step-z-empty*:  
**assumes**  $A \vdash \langle l, Z \rangle \rightsquigarrow_a \langle l', Z' \rangle$   $Z = \{\}$   
**shows**  $Z' = \{\}$   
**using** *assms*  
**apply** *cases*  
**unfolding** *zone-delay-def zone-set-def*  
**by** *auto*

### 8.1.3 Connecting with Correctness Results for Approximating Semantics

**context**  
**fixes**  $A :: ('a, 'c, \text{real}, 's) \text{ta}$   
**assumes** *gcn: global-clock-numbering A v n*  
**and** *va: valid-abstraction A X k*  
**begin**

**context**  
**notes** [*intro*] = *step-z-valid-dbm*[*OF - gcn va*]  
**begin**

**lemma** *valid-dbm-step-z-norm''*:  
*valid-dbm D' if*  $A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle$  *valid-dbm D*  
**using** *that unfolding step-z-norm''-def* **by** (*auto intro: step-z-norm-valid-dbm*[*OF - gcn va*])

**lemma** *steps-z-norm'-valid-dbm-invariant*:  
*valid-dbm D' if*  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle$  *valid-dbm D*  
**using** *that by (induction rule: rtranclp-induct2)* (*auto intro: valid-dbm-step-z-norm''*)

**lemma** *norm-beta-sound''*:  
**assumes**  $A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l'', D'' \rangle$   
**and** *valid-dbm D*

**shows**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta} \langle l'', [D'']_{v,n} \rangle$   
**proof** –  
**from** *assms(1)* **obtain**  $l' D'$  **where**  
 $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,\tau} \langle l', D' \rangle$   $A \vdash \langle l', D' \rangle \rightsquigarrow_{\mathcal{N}(1a)} \langle l'', D'' \rangle$   
**by** (*auto simp: step-z-norm''-def*)  
**moreover with**  $\langle \text{valid-dbm } D \rangle$  **have** *valid-dbm*  $D'$   
**by** *auto*  
**ultimately have**  $A \vdash \langle l', [D']_{v,n} \rangle \rightsquigarrow_{\beta \upharpoonright a} \langle l'', [D'']_{v,n} \rangle$   
**by** – (*rule norm-beta-sound[OF - gcn va]*)  
**with** *step-z-dbm-sound[OF  $\langle A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,\tau} \langle l', D' \rangle \rangle$  gcn]* **show** *?thesis*  
**unfolding** *step-z-beta'-def* **by** – (*frule step-z.cases[where  $P = l' = l$ ];*  
*force*)  
**qed**

**lemma** *norm-beta-complete1*:  
**assumes**  $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta} \langle l'', Z'' \rangle$   
**and** *valid-dbm*  $D$   
**obtains**  $a D''$  **where**  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l'', D'' \rangle$   $[D'']_{v,n} = Z''$  *valid-dbm*  
 $D''$   
**proof** –  
**from** *assms(1)* **obtain**  $l' Z'$  **where** *steps*:  
 $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\tau} \langle l', Z' \rangle$   $A \vdash \langle l', Z' \rangle \rightsquigarrow_{\beta(1a)} \langle l'', Z'' \rangle$   
**by** (*auto simp: step-z-beta'-def*)  
**from** *step-z-dbm-DBM[OF this(1) gcn]* **obtain**  $D'$  **where**  $D'$ :  
 $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n,\tau} \langle l', D' \rangle$   $Z' = [D']_{v,n}$   
**by** *auto*  
**with**  $\langle \text{valid-dbm } D \rangle$  **have** *valid-dbm*  $D'$   
**by** *auto*  
**from** *steps*  $D'$  **show** *?thesis*  
**by** (*auto*  
*intro!*: *that[unfolded step-z-norm''-def]*  
*elim!*: *norm-beta-complete[OF - gcn va  $\langle \text{valid-dbm } D' \rangle$ ]*  
)

**qed**

**lemma** *bisim*:

*Bisimulation-Invariant*

$(\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \wedge Z' \neq \{\})$

$(\lambda (l, D) (l', D'). \exists a. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle \wedge [D']_{v,n} \neq \{\})$

$(\lambda (l, Z) (l', D). l = l' \wedge Z = [D]_{v,n})$

$(\lambda -. \text{True}) (\lambda (l, D). \text{valid-dbm } D)$

**proof** (*standard, goal-cases*)

–  $\beta \Rightarrow \mathcal{N}$

**case** (1 a b a')  
**then show** ?case  
**by** (blast elim: norm-beta-complete1)  
**next**  
—  $\mathcal{N} \Rightarrow \beta$   
**case** (2 a a' b')  
**then show** ?case  
**by** (blast intro: norm-beta-sound'')  
**next**  
—  $\beta$  invariant  
**case** (3 a b)  
**then show** ?case  
**by** simp  
**next**  
—  $\mathcal{N}$  invariant  
**case** (4 a b)  
**then show** ?case  
**unfolding** step-z-norm''-def  
**by** (auto intro: step-z-norm-valid-dbm[OF - gcn va])  
**qed**

**end**

**interpretation** Bisimulation-Invariant

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \wedge Z' \neq \{\}$   
 $\lambda (l, D) (l', D'). \exists a. A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle \wedge [D']_{v,n} \neq \{\}$   
 $\lambda (l, Z) (l', D). l = l' \wedge Z = [D]_{v,n}$   
 $\lambda -. \text{True } \lambda (l, D). \text{valid-dbm } D$   
**by** (rule bisim)

**lemma** step-z-norm''-non-empty:

$[D]_{v,n} \neq \{\}$  **if**  $A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle [D']_{v,n} \neq \{\}$  *valid-dbm*  $D$

**proof** —

**from** that B-A-step[of (l, D) (l', D') (l, [D]<sub>v,n</sub>)] **have**

$A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta} \langle l', [D']_{v,n} \rangle$

**by** auto

**with**  $\langle - \neq \{\} \rangle$  **show** ?thesis

**by** (auto 4 3 dest: step-z-beta'-empty)

**qed**

**lemma** norm-steps-empty:

$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge [D']_{v,n} \neq \{\} \iff B.\text{reaches } (l, D) (l', D') \wedge [D]_{v,n} \neq \{\}$

**if** *valid-dbm D*  
**apply** (*subst rtranclp-backwards-invariant-iff* [  
*of*  $\lambda(l, D) (l', D'). \exists a. A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle (l, D) \lambda(l, D). [D]_{v,n}$   
 $\neq \{\}$ ,  
*simplified*  
 $\rangle$ )  
**using**  $\langle \text{valid-dbm } D \rangle$   
**by** (*auto dest!: step-z-norm''-non-empty intro: steps-z-norm'-valid-dbm-invariant*)

**context**

**fixes**  $P Q :: 's \Rightarrow \text{bool}$  — The state property we want to check

**begin**

**interpretation** *bisim-ψ: Bisimulation-Invariant*

$\lambda (l, Z) (l', Z'). A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \wedge Z' \neq \{\} \wedge Q l'$

$\lambda (l, D) (l', D'). \exists a. A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}(a)} \langle l', D' \rangle \wedge [D]_{v,n} \neq \{\} \wedge Q l'$

$\lambda (l, Z) (l', D). l = l' \wedge Z = [D]_{v,n}$

$\lambda -. \text{True } \lambda (l, D). \text{valid-dbm } D$

**by** (*rule Bisimulation-Invariant-filter[OF bisim, of  $\lambda (l, -)$ .  $Q l \lambda (l, -)$ .  $Q l$ ]*) *auto*

**end**

**context**

**assumes** *finite-state-set: finite (state-set A)*

**begin**

**interpretation** *R: Regions-TA*

**by** (*standard; rule va finite-state-set*)

**lemma** *A-reaches-non-empty:*

$Z' \neq \{\}$  **if** *A.reaches (l, Z) (l', Z')*  $Z \neq \{\}$

**using** *that by cases auto*

**lemma** *A-reaches-start-non-empty-iff:*

$(\exists Z'. (\exists u. u \in Z') \wedge \text{A.reaches } (l, Z) (l', Z')) \longleftrightarrow (\exists Z'. \text{A.reaches } (l, Z) (l', Z')) \wedge Z \neq \{\}$

**apply** *safe*

**apply** *blast*

**subgoal**

**by** (*auto dest: step-z-beta'-empty elim: converse-rtranclpE2*)

**by** (*auto dest: A-reaches-non-empty*)

**lemma** *step-z-norm''-state-set1*:

$l \in \text{state-set } A$  **if**  $A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}a} \langle l', D' \rangle$

**using** *that unfolding step-z-norm''-def*

**by** (*auto dest: step-z-dbm-delay-loc intro: step-z-dbm-action-state-set1*)

**lemma** *step-z-norm''-state-set2*:

$l' \in \text{state-set } A$  **if**  $A \vdash' \langle l, D \rangle \rightsquigarrow_{\mathcal{N}a} \langle l', D' \rangle$

**using** *that unfolding step-z-norm''-def* **by** (*auto intro: step-z-dbm-action-state-set2*)

**theorem** *steps-z-norm-decides-emptiness*:

**assumes** *valid-dbm D*

**shows**  $(\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge [D']_{v,n} \neq \{\})$

$\longleftrightarrow (\exists u \in [D]_{v,n}. (\exists u'. A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u' \rangle))$

**proof** (*cases*  $[D]_{v,n} = \{\}$ )

**case** *True*

**then show** *?thesis*

**unfolding** *norm-steps-empty[OF <valid-dbm D>]* **by** *auto*

**next**

**case** *F: False*

**show** *?thesis*

**proof** (*cases*  $l \in \text{state-set } A$ )

**case** *True*

**interpret** *Regions-TA-Start-State v n not-in-X X k A l [D]\_{v,n}*

**using** *assms F True* **by**  $-$  (*standard, auto elim!: valid-dbm-V'*)

**show** *?thesis*

**unfolding** *steps'-iff[symmetric] norm-steps-empty[OF <valid-dbm D>]*

**using**

*reaches-ex-iff[of  $\lambda (l, -). l = l' \lambda (l, -). l = l' (l, [D]_{v,n}) (l, D)$ ]*

*<valid-dbm D> ta-reaches-ex-iff[of  $\lambda (l, -). l = l'$ ]*

**by** (*auto simp: A-reaches-start-non-empty-iff from-R-def a0-def*)

**next**

**case** *False*

**have**  $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \longleftrightarrow (D' = D \wedge l' = l)$  **for**  $D'$

**using** *False* **by** (*blast dest: step-z-norm''-state-set1 elim: converse-rtranclpE2*)

**moreover have**  $A \vdash' \langle l, u \rangle \rightarrow^* \langle l', u' \rangle \longleftrightarrow (u' = u \wedge l' = l)$  **for**  $u u'$

**unfolding** *steps'-iff[symmetric]* **using** *False*

**by** (*blast dest: step'-state-set1 elim: converse-rtranclpE2*)

**ultimately show** *?thesis*

**using** *F* **by** *auto*

**qed**

**qed**

**end**

**end**

**context**

**fixes**  $A :: ('a, 'c, \text{real}, 's) \text{ta}$   
**assumes**  $gcn: \text{global-clock-numbering } A \ v \ n$   
**and**  $va: \text{valid-abstraction } A \ X \ k$

**begin**

**lemmas**

$\text{step-z-norm-valid-dbm}' = \text{step-z-norm-valid-dbm}[OF - gcn \ va]$

**lemmas**

$\text{step-z-valid-dbm}' = \text{step-z-valid-dbm}[OF - gcn \ va]$

**lemmas**  $\text{norm-beta-sound}' = \text{norm-beta-sound}[OF - gcn \ va]$

**lemma**  $v\text{-bound}$ :

$\forall c \in \text{clk-set } A. \ v \ c \leq n$   
**using**  $gcn$  **by**  $\text{blast}$

**lemmas**  $\text{alpha-beta-step}'' = \text{alpha-beta-step}'[OF - va \ v\text{-bound}]$

**lemmas**  $\text{step-z-dbm-sound}' = \text{step-z-dbm-sound}[OF - gcn]$

**lemmas**  $\text{step-z-V}'' = \text{step-z-V}'[OF - va \ v\text{-bound}]$

**end**

**end**

## 8.2 Additional Useful Properties of the Normalized Semantics

Obsolete

**lemma**  $\text{norm-diag-alt-def}$ :

$\text{norm-diag } e = (\text{if } e < 0 \text{ then } Lt \ 0 \ \text{else if } e = 0 \ \text{then } e \ \text{else } \infty)$   
**unfolding**  $\text{norm-diag-def } DBM.\text{neutral } DBM.\text{less } ..$

**lemma**  $\text{norm-diag-preservation}$ :

**assumes**  $\forall l \leq n. \ M1 \ ll \leq 0$   
**shows**  $\forall l \leq n. \ (\text{norm } M1 \ (k :: \text{nat} \Rightarrow \text{nat}) \ n) \ ll \leq 0$   
**using**  $\text{assms}$  **unfolding**  $\text{norm-def } \text{norm-diag-alt-def}$  **by**  $(\text{auto simp: } DBM.\text{neutral})$

### 8.3 Appendix: Standard Clock Numberings for Concrete Models

```

locale Regions' =
  fixes  $X$  and  $k :: 'c \Rightarrow \text{nat}$  and  $v :: 'c \Rightarrow \text{nat}$  and  $n :: \text{nat}$  and not-in-X
  assumes finite: finite X
  assumes clock-numbering':  $\forall c \in X. v\ c > 0 \ \forall c. c \notin X \longrightarrow v\ c > n$ 
  assumes bij: bij-betw v X {1..n}
  assumes non-empty:  $X \neq \{\}$ 
  assumes not-in-X: not-in-X  $\notin X$ 

begin

lemma inj: inj-on v X using bij-betw-imp-inj-on bij by simp

lemma cn-weak:  $\forall c. v\ c > 0$  using clock-numbering' by force

lemma in-X: assumes  $v\ x \leq n$  shows  $x \in X$  using assms clock-numbering'(2)
by force

end

sublocale Regions'  $\subseteq$  Regions-global
proof (unfold-locales, auto simp: finite clock-numbering' non-empty cn-weak
not-in-X, goal-cases)
  case (1  $x\ y$ ) with inj in-X show ?case unfolding inj-on-def by auto
next
  case (2  $k$ )
  from bij have  $v\ 'X = \{1..n\}$  unfolding bij-betw-def by auto
  from 2 have  $k \in \{1..n\}$  by simp
  then obtain  $x$  where  $x \in X \ v\ x = k$  unfolding image-def
  by (metis (no-types, lifting) v 'X = {1..n} imageE)
  then show ?case by blast
next
  case (3  $x$ ) with bij show ?case unfolding bij-betw-def by auto
qed

lemma standard-abstraction:
  assumes
    finite (Timed-Automata.clkp-set A) finite (Timed-Automata.collect-clkvt
(trans-of A))
     $\forall (-, m :: \text{real}) \in \text{Timed-Automata.clkp-set } A. m \in \mathbb{N}$ 
  obtains  $k :: 'c \Rightarrow \text{nat}$  where Timed-Automata.valid-abstraction A (clk-set

```

A)  $k$   
**proof** –  
**from** *assms* **have** 1: *finite (clk-set A)* **by** *auto*  
**have** 2: *Timed-Automata.collect-clkvt (trans-of A)  $\subseteq$  clk-set A* **by** *auto*  
**from** *assms* **obtain**  $L$  **where**  $L$ : *distinct L set L = Timed-Automata.clkp-set A*  
**by** (*meson finite-distinct-list*)  
**let**  $?M = \lambda c. \{m . (c, m) \in \text{Timed-Automata.clkp-set } A\}$   
**let**  $?X = \text{clk-set } A$   
**let**  $?m = \text{map-of } L$   
**let**  $?k = \lambda x. \text{if } ?M x = \{\} \text{ then } 0 \text{ else } \text{nat (floor (Max (?M x)) + 1)}$   
**{** **fix**  $c\ m$  **assume**  $A: (c, m) \in \text{Timed-Automata.clkp-set } A$   
**from** *assms(1)* **have** *finite (snd ‘ Timed-Automata.clkp-set A)* **by** *auto*  
**moreover** **have**  $?M\ c \subseteq (\text{snd ‘ Timed-Automata.clkp-set } A)$  **by** *force*  
**ultimately** **have** *fin: finite (?M c)* **by** (*blast intro: finite-subset*)  
**then** **have**  $\text{Max} (?M\ c) \in \{m . (c, m) \in \text{Timed-Automata.clkp-set } A\}$   
**using** *Max-in A* **by** *auto*  
**with** *assms(3)* **have**  $\text{Max} (?M\ c) \in \mathbb{N}$  **by** *auto*  
**then** **have**  $\text{floor} (\text{Max} (?M\ c)) = \text{Max} (?M\ c)$  **by** (*metis Nats-cases floor-of-nat of-int-of-nat-eq*)  
**have**  $*$ :  $?k\ c = \text{Max} (?M\ c) + 1$   
**proof** –  
**have**  $\text{real} (\text{nat} (n + 1)) = \text{real-of-int } n + 1$   
**if**  $\text{Max} \{m. (c, m) \in \text{Timed-Automata.clkp-set } A\} = \text{real-of-int } n$   
**for**  $n :: \text{int}$  **and**  $x :: \text{real}$   
**proof** –  
**from** *that* **have**  $\text{real-of-int} (n + 1) \in \mathbb{N}$   
**using**  $\langle \text{Max} \{m. (c, m) \in \text{Timed-Automata.clkp-set } A\} \in \mathbb{N} \rangle$  **by**  
*auto*  
**then** **show** *?thesis*  
**by** (*metis Nats-cases ceiling-of-int nat-int of-int-1 of-int-add of-int-of-nat-eq*)  
**qed**  
**with**  $A \langle \text{floor} (\text{Max} (?M\ c)) = \text{Max} (?M\ c) \rangle$  **show** *?thesis*  
**by** *auto*  
**qed**  
**from** *fin A* **have**  $\text{Max} (?M\ c) \geq m$  **by** *auto*  
**moreover** **from**  $A$  *assms(3)* **have**  $m \in \mathbb{N}$  **by** *auto*  
**ultimately** **have**  $m \leq ?k\ c\ m \in \mathbb{N}\ c \in \text{clk-set } A$  **using**  $A\ *$  **by** *force+*  
**}**  
**then** **have**  $\forall (x, m) \in \text{Timed-Automata.clkp-set } A. m \leq ?k\ x \wedge x \in \text{clk-set } A \wedge m \in \mathbb{N}$  **by** *blast*  
**with** 1 2 **have** *Timed-Automata.valid-abstraction A ?X ?k* **by** – (*standard, assumption+*)

then show *thesis* ..  
 qed

**definition**

*finite-ta*  $A \equiv$   
 $\text{finite}(\text{Timed-Automata.clk-set } A) \wedge \text{finite}(\text{Timed-Automata.collect-clkvt}(\text{trans-of } A))$   
 $\wedge (\forall (-, m) \in \text{Timed-Automata.clk-p-set } A. m \in \mathbb{N}) \wedge \text{clk-set } A \neq \{\} \wedge$   
 $\neg \text{clk-set } A \neq \{\}$

**lemma** *finite-ta-Regions'*:

**fixes**  $A :: ('a, 'c, \text{real}, 's) \text{ ta}$

**assumes** *finite-ta*  $A$

**obtains**  $v \ n \ x$  **where** *Regions'* ( $\text{clk-set } A$ )  $v \ n \ x$

**proof** –

**from** *assms* **obtain**  $x$  **where**  $x \notin \text{clk-set } A$  **unfolding** *finite-ta-def* **by** *auto*

**from** *assms*(1) **have**  $\text{finite}(\text{clk-set } A)$  **unfolding** *finite-ta-def* **by** *auto*

**with** *standard-numbering*[of  $\text{clk-set } A$ ] *assms* **obtain**  $v$  **and**  $n :: \text{nat}$  **where**  
 $\text{bij-betw } v \ (\text{clk-set } A) \ \{1..n\}$

$\forall c \in \text{clk-set } A. 0 < v \ c \ \forall c. c \notin \text{clk-set } A \longrightarrow n < v \ c$

**by** *auto*

**then have** *Regions'* ( $\text{clk-set } A$ )  $v \ n \ x$  **using**  $x$  *assms* **unfolding** *finite-ta-def* **by** – (*standard*, *auto*)

**then show** *?thesis* ..

qed

**lemma** *finite-ta-RegionsD*:

**fixes**  $A :: ('a, 'c, t, 's) \text{ ta}$

**assumes** *finite-ta*  $A$

**obtains**  $k :: 'c \Rightarrow \text{nat}$  **and**  $v \ n \ x$  **where**

*Regions'* ( $\text{clk-set } A$ )  $v \ n \ x$  *Timed-Automata.valid-abstraction*  $A$  ( $\text{clk-set } A$ )  $k$

*global-clock-numbering*  $A \ v \ n$

**proof** –

**from** *standard-abstraction* *assms* **obtain**  $k :: 'c \Rightarrow \text{nat}$  **where**  $k$ :

*Timed-Automata.valid-abstraction*  $A$  ( $\text{clk-set } A$ )  $k$

**unfolding** *finite-ta-def* **by** *blast*

**from** *finite-ta-Regions'*[OF *assms*] **obtain**  $v \ n \ x$  **where**  $*$ : *Regions'* ( $\text{clk-set } A$ )  $v \ n \ x$ .

**then interpret** *interp*: *Regions'*  $\text{clk-set } A \ k \ v \ n \ x$ .

**from** *interp.clock-numbering* **have** *global-clock-numbering*  $A \ v \ n$  **by** *blast*

**with**  $*$   $k$  **show** *?thesis* ..

qed

**definition** *valid-dbm* **where** *valid-dbm*  $M\ n \equiv \text{dbm-int } M\ n \wedge (\forall\ i \leq n. M\ 0\ i \leq 0)$

**lemma** *dbm-positive*:

**assumes**  $M\ 0\ (v\ c) \leq 0\ v\ c \leq n$  *DBM-val-bounded*  $v\ u\ M\ n$   
**shows**  $u\ c \geq 0$

**proof** –

**from** *assms* **have** *dbm-entry-val*  $u\ \text{None}\ (\text{Some}\ c)\ (M\ 0\ (v\ c))$  **unfolding** *DBM-val-bounded-def* **by** *auto*

**with** *assms*(1) **show** *?thesis*

**proof** (*cases*  $M\ 0\ (v\ c)$ , *goal-cases*)

**case** 1

**then show** *?case* **unfolding** *less-eq neutral* **using** *order-trans* **by** (*fastforce dest!*: *le-dbm-le*)

**next**

**case** 2

**then show** *?case* **unfolding** *less-eq neutral*

**by** (*auto dest!*: *lt-dbm-le*) (*meson less-trans neg-0-less-iff-less not-less*)

**next**

**case** 3

**then show** *?case* **unfolding** *neutral less-eq dbm-le-def* **by** *auto*

**qed**

**qed**

**lemma** *valid-dbm-pos*:

**assumes** *valid-dbm*  $M\ n$

**shows**  $[M]_{v,n} \subseteq \{u. \forall\ c. v\ c \leq n \longrightarrow u\ c \geq 0\}$

**using** *dbm-positive assms* **unfolding** *valid-dbm-def* **unfolding** *DBM-zone-repr-def* **by** *fast*

**lemma** (*in* *Regions'*) *V-alt-def*:

**shows**  $\{u. \forall\ c. v\ c > 0 \wedge v\ c \leq n \longrightarrow u\ c \geq 0\} = V$

**unfolding** *V-def* **using** *clock-numbering* **by** *metis*

**end**

## References

- [AD90] Rajeev Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
- [BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, pages 87–124, 2003.
- [HHWt97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.
- [LPY97] G. Kim Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.
- [Yov97] Sergio Yovine. KRONOS: A verification tool for real-time systems. *STTT*, 1(1-2):123–133, 1997.