Büchi Complementation

Julian Brunner

June 25, 2026

Abstract

This entry provides a verified implementation of rank-based Büchi Complementation [1]. The verification is done in three steps:

1. Definition of odd rankings and proof that an automaton rejects a word iff there exists an odd ranking for it.
2. Definition of the complement automaton and proof that it accepts exactly those words for which there is an odd ranking.
3. Verified implementation of the complement automaton using the Isabelle Collections Framework.

Contents

1 Alternating Function Iteration **2**

2 Run Graphs **3**

3 Rankings **7**
3.1 Rankings 8
3.2 Ranking Implies Word not in Language 8
3.3 Word not in Language Implies Ranking 10
3.3.1 Removal of Endangered Nodes 10
3.3.2 Removal of Safe Nodes 10
3.3.3 Run Graph Interation 11
3.4 Node Ranks 16
3.5 Correctness Theorem 18

4 Complementation **18**
4.1 Level Rankings and Complementation States 18
4.2 Word in Complement Language Implies Ranking 21
4.3 Ranking Implies Word in Complement Language 25
4.4 Correctness Theorem 32

5 Complementation Implementation **32**
5.1 Phase 1 . . . 32
5.2 Phase 2 . . . 37
5.3 Phase 3 . . . 39
5.4 Phase 4 . . . 41
5.5 Phase 5 . . . 47
5.6 Phase 6 . . . 49
5.7 Phase 7 . . . 50

6 Boolean Formulae **54**

7 Final Instantiation of Algorithms Related to Complementation **55**
7.1 Syntax . . . 55
7.2 Hashcodes on Complement States . . . 55
7.3 Complementation . . . 56
7.4 Language Subset . . . 57
7.5 Language Equality . . . 58

8 Build and test exported program with MLton **59**

1 Alternating Function Iteration

theory *Alternate*
imports *Main*
begin

primrec *alternate* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow nat \Rightarrow ('a \Rightarrow 'a)$ **where**
alternate f g 0 = *id* | *alternate f g* (*Suc k*) = *alternate g f k* ∘ *f*

lemma *alternate-Suc*[*simp*]: *alternate f g* (*Suc k*) = (*if even k then f else g*) ∘ *alternate f g k*
proof (*induct k arbitrary*: *f g*)
case (*0*)
show *?case* **by** *simp*
next
case (*Suc k*)
have *alternate f g* (*Suc* (*Suc k*)) = *alternate g f* (*Suc k*) ∘ *f* **by** *auto*
also have ... = (*if even k then g else f*) ∘ (*alternate g f k* ∘ *f*) **unfolding** *Suc* **by** *auto*
also have ... = (*if even* (*Suc k*) *then f else g*) ∘ *alternate f g* (*Suc k*) **by** *auto*
finally show *?case* **by** *this*
qed

declare *alternate.simps*(*2*)[*simp del*]

lemma *alternate-antimono*:

assumes $\bigwedge x.\ f\ x \leq x \bigwedge x.\ g\ x \leq x$
shows *antimono* (*alternate f g*)
proof
fix *k l* :: *nat*
assume *1*: $k \leq l$
obtain *n* **where** *2*: $l = k + n$ **using** *le-Suc-ex 1* **by** *auto*
have *3*: *alternate f g* $(k + n) \leq$ *alternate f g k*
proof (*induct n*)
case (*0*)
show *?case* **by** *simp*
next
case (*Suc n*)
have *alternate f g* $(k +$ *Suc n*$) \leq$ *alternate f g* $(k + n)$ **using** *assms* **by** (*auto intro*: *le-funI*)
also have $\ldots \leq$ *alternate f g k* **using** *Suc* **by** *this*
finally show *?case* **by** *this*
qed
show *alternate f g l* $\leq$ *alternate f g k* **using** *3* **unfolding** *2* **by** *this*
qed

end

2 Run Graphs

theory *Graph*
imports *Transition-Systems-and-Automata.NBA*
begin

type-synonym *'state node* = *nat* × *'state*

abbreviation *ginitial A* ≡ $\{0\}$ × *initial A*
abbreviation *gaccepting A* ≡ *accepting A* ∘ *snd*

global-interpretation *graph*: *transition-system-initial*
const
$\lambda\ u\ (k,\ p).\ w\ !!\ k \in$ *alphabet A* $\wedge\ u \in \{$*Suc k*$\}$ × *transition A* $(w\ !!\ k)\ p \cap V$
$\lambda\ v.\ v \in$ *ginitial A* $\cap\ V$
for *A w V*
defines
gpath = *graph.path* **and** *grun* = *graph.run* **and**
greachable = *graph.reachable* **and** *gnodes* = *graph.nodes*
by *this*

We disable rules that are degenerate due to *execute* = $(\lambda x\ \text{-}.\ x)$.

declare *graph.reachable.execute*[*rule del*]
declare *graph.nodes.execute*[*rule del*]

abbreviation *gtarget* ≡ *graph.target*
abbreviation *gstates* ≡ *graph.states*

abbreviation *gtrace* ≡ *graph.trace*

abbreviation *gsuccessors* :: ('*label*, '*state*) *nba* ⇒ '*label stream* ⇒
'*state node set* ⇒ '*state node* ⇒ '*state node set* **where**
gsuccessors A w V ≡ *graph.successors TYPE*('*label*) *w A V*

abbreviation *gusuccessors A w* ≡ *gsuccessors A w UNIV*
abbreviation *gupath A w* ≡ *gpath A w UNIV*
abbreviation *gurun A w* ≡ *grun A w UNIV*
abbreviation *gureachable A w* ≡ *greachable A w UNIV*
abbreviation *gunodes A w* ≡ *gnodes A w UNIV*

lemma *gtarget-alt-def*: *gtarget r v* = *last* (*v* # *r*) **using** *fold-const* **by** *this*
lemma *gstates-alt-def*: *gstates r v* = *r* **by** *simp*
lemma *gtrace-alt-def*: *gtrace r v* = *r* **by** *simp*

lemma *gpath-elim*[*elim?*]:
assumes *gpath A w V s v*
obtains *r k p*
where *s* = [*Suc k* ..< *Suc k* + *length r*] || *r v* = (*k*, *p*)
proof −
obtain *t r* **where** *1*: *s* = *t* || *r length t* = *length r*
using *zip-map-fst-snd*[*of s*] **by** (*metis length-map*)
obtain *k p* **where** *2*: *v* = (*k*, *p*) **by** *force*
have *3*: *t* = [*Suc k* ..< *Suc k* + *length r*]
using *assms 1 2*
proof (*induct arbitrary*: *t r k p*)
case (*nil v*)
then show *?case* **by** (*metis add-0-right le-add1 length-0-conv length-zip min.idem upt-conv-Nil*)
next
case (*cons u v s*)
have *1*: *t* || *r* = (*hd t*, *hd r*) # (*tl t* || *tl r*)
by (*metis cons.prems*(*1*) *hd-Cons-tl neq-Nil-conv zip.simps*(*1*) *zip-Cons-Cons zip-Nil*)
have *2*: *s* = *tl t* || *tl r* **using** *cons 1* **by** *simp*
have *t* = *hd t* # *tl t* **using** *cons*(*4*) **by** (*metis hd-Cons-tl list.simps*(*3*) *zip-Nil*)
also have *hd t* = *Suc k* **using** *1 cons.hyps*(*1*) *cons.prems*(*1*) *cons.prems*(*3*) **by** *auto*
also have *tl t* = [*Suc* (*Suc k*) ..< *Suc* (*Suc k*) + *length* (*tl r*)]
using *cons*(*3*)[*OF 2*] **using** *1* ‹*hd t* = *Suc k*› *cons.prems*(*1*) *cons.prems*(*2*) **by** *auto*
finally show *?case* **using** *cons.prems*(*2*) *upt-rec* **by** *auto*
qed
show *?thesis* **using** *that 1 2 3* **by** *simp*
qed

lemma *gpath-path*[*symmetric*]: *path A* (*stake* (*length r*) (*sdrop k w*) || *r*) *p* ⟷
gpath A w UNIV ([*Suc k* ..< *Suc k* + *length r*] || *r*) (*k*, *p*)

```
  proof (induct r arbitrary: k p)
    case (Nil)
    show ?case by auto
  next
    case (Cons q r)
    have 1: path A (stake (length r) (sdrop (Suc k) w) || r) q ⟷
      gpath A w UNIV ([Suc (Suc k) ..< Suc k + length (q # r)] || r) (Suc k, q)
      using Cons[of Suc k q] by simp
    have stake (length (q # r)) (sdrop k w) || q # r =
      (w !! k, q) # (stake (length r) (sdrop (Suc k) w) || r) by simp
    also have path A ... p ⟷
      gpath A w UNIV ((Suc k, q) # ([Suc (Suc k) ..< Suc k + length (q # r)] ||
r)) (k, p)
      using 1 by auto
    also have (Suc k, q) # ([Suc (Suc k) ..< Suc k + length (q # r)] || r) =
        Suc k # [Suc (Suc k) ..< Suc k + length (q # r)] || q # r unfolding
zip-Cons-Cons by rule
    also have Suc k # [Suc (Suc k) ..< Suc k + length (q # r)] = [Suc k ..< Suc
k + length (q # r)]
      by (simp add: upt-rec)
    finally show ?case by this
  qed

  lemma grun-elim[elim?]:
    assumes grun A w V s v
    obtains r k p
    where s = fromN (Suc k) ||| r v = (k, p)
  proof −
    obtain t r where 1: s = t ||| r using szip-smap by metis
    obtain k p where 2: v = (k, p) by force
    have 3: t = fromN (Suc k)
      using assms unfolding 1 2
      by (coinduction arbitrary: t r k p) (force iff: eq-scons elim: graph.run.cases)
    show ?thesis using that 1 2 3 by simp
  qed

  lemma run-grun:
    assumes run A (sdrop k w ||| r) p
    shows gurun A w (fromN (Suc k) ||| r) (k, p)
    using assms by (coinduction arbitrary: k p r) (auto elim: nba.run.cases)

  lemma grun-run:
    assumes grun A w V (fromN (Suc k) ||| r) (k, p)
    shows run A (sdrop k w ||| r) p
  proof −
    have 2: ∃ ka wa. sdrop k (stl w :: 'a stream) = sdrop ka wa ∧ P ka wa if P
(Suc k) w for P k w
      using that by (metis sdrop.simps(2))
    show ?thesis using assms by (coinduction arbitrary: k p w r) (auto intro!: 2
```

elim: *graph.run.cases*)
qed

lemma *greachable-reachable*:
fixes $l\ q\ k\ p$
defines $u \equiv (l,\ q)$
defines $v \equiv (k,\ p)$
assumes $u \in greachable\ A\ w\ V\ v$
shows $q \in reachable\ A\ p$
using *assms*(*3*, *1*, *2*)
proof (*induct arbitrary*: $l\ q\ k\ p$)
case *reflexive*
then show *?case* **by** *auto*
next
case (*execute u*)
have *1*: $q \in successors\ A\ (snd\ u)$ **using** *execute* **by** *auto*
have $snd\ u \in reachable\ A\ p$ **using** *execute* **by** *auto*
also have $q \in reachable\ A\ (snd\ u)$ **using** *1* **by** *blast*
finally show *?case* **by** *this*
qed

lemma *gnodes-nodes*: $gnodes\ A\ w\ V \subseteq UNIV \times nodes\ A$
proof
fix *v*
assume $v \in gnodes\ A\ w\ V$
then show $v \in UNIV \times nodes\ A$ **by** *induct auto*
qed

lemma *gpath-subset*:
assumes $gpath\ A\ w\ V\ r\ v$
assumes $set\ (gstates\ r\ v) \subseteq U$
shows $gpath\ A\ w\ U\ r\ v$
using *assms* **by** *induct auto*
lemma *grun-subset*:
assumes $grun\ A\ w\ V\ r\ v$
assumes $sset\ (gtrace\ r\ v) \subseteq U$
shows $grun\ A\ w\ U\ r\ v$
using *assms*
proof (*coinduction arbitrary*: $r\ v$)
case (*run a s r v*)
have *1*: $grun\ A\ w\ V\ s\ a$ **using** *run*(*1*, *2*) **by** *fastforce*
have *2*: $a \in gusuccessors\ A\ w\ v$ **using** *run*(*1*, *2*) **by** *fastforce*
show *?case* **using** *1 2 run*(*1*, *3*) **by** *force*
qed

lemma *greachable-subset*: $greachable\ A\ w\ V\ v \subseteq insert\ v\ V$
proof
fix *u*
assume $u \in greachable\ A\ w\ V\ v$

then show *u* ∈ *insert v V* **by** *induct auto*
qed

lemma *gtrace-infinite*:
assumes *grun A w V r v*
shows *infinite* (*sset* (*gtrace r v*))
using *assms* **by** (*metis grun-elim gtrace-alt-def infinite-Ici sset-fromN sset-szip-finite*)

lemma *infinite-greachable-gtrace*:
assumes *grun A w V r v*
assumes *u* ∈ *sset* (*gtrace r v*)
shows *infinite* (*greachable A w V u*)
proof −
obtain *i* **where** *1*: *u* = *gtrace r v* !! *i* **using** *sset-range imageE assms*(*2*) **by** *metis*
have *2*: *gtarget* (*stake* (*Suc i*) *r*) *v* = *u* **unfolding** *1 sscan-snth* **by** *rule*
have *infinite* (*sset* (*sdrop* (*Suc i*) (*gtrace r v*)))
using *gtrace-infinite*[*OF assms*(*1*)]
by (*metis List.finite-set finite-Un sset-shift stake-sdrop*)
also have *sdrop* (*Suc i*) (*gtrace r v*) = *gtrace* (*sdrop* (*Suc i*) *r*) (*gtarget* (*stake* (*Suc i*) *r*) *v*)
by *simp*
also have *sset* ... ⊆ *greachable A w V u*
using *assms*(*1*) *2* **by** (*metis graph.reachable.reflexive graph.reachable-trace graph.run-sdrop*)
finally show *?thesis* **by** *this*
qed

lemma *finite-nodes-gsuccessors*:
assumes *finite* (*nodes A*)
assumes *v* ∈ *gunodes A w*
shows *finite* (*gusuccessors A w v*)
proof −
have *gusuccessors A w v* ⊆ *gureachable A w v* **by** *rule*
also have ... ⊆ *gunodes A w* **using** *assms*(*2*) **by** *blast*
also have ... ⊆ *UNIV* × *nodes A* **using** *gnodes-nodes* **by** *this*
finally have *3*: *gusuccessors A w v* ⊆ *UNIV* × *nodes A* **by** *this*
have *gusuccessors A w v* ⊆ {*Suc* (*fst v*)} × *nodes A* **using** *3* **by** *auto*
also have *finite* ... **using** *assms*(*1*) **by** *simp*
finally show *?thesis* **by** *this*
qed

end

3 Rankings

theory *Ranking*
imports
Alternate

Graph
begin

3.1 Rankings

type-synonym *'state ranking = 'state node ⇒ nat*

definition *ranking* :: *('label, 'state) nba ⇒ 'label stream ⇒ 'state ranking ⇒ bool*
where
ranking A w f ≡
(∀ v ∈ gunodes A w. f v ≤ 2 ∗ card (nodes A)) ∧
(∀ v ∈ gunodes A w. ∀ u ∈ gusuccessors A w v. f u ≤ f v) ∧
(∀ v ∈ gunodes A w. gaccepting A v ⟶ even (f v)) ∧
(∀ v ∈ gunodes A w. ∀ r k. gurun A w r v ⟶ smap f (gtrace r v) = sconst k ⟶ odd k)

3.2 Ranking Implies Word not in Language

lemma *ranking-stuck*:
assumes *ranking A w f*
assumes *v ∈ gunodes A w gurun A w r v*
obtains *n k*
where *smap f (gtrace (sdrop n r) (gtarget (stake n r) v)) = sconst k*
proof −
have *0*: *f u ≤ f v* **if** *v ∈ gunodes A w u ∈ gusuccessors A w v* **for** *v u*
using *assms(1) that* **unfolding** *ranking-def* **by** *auto*
have *1*: *shd (v ## gtrace r v) ∈ gunodes A w* **using** *assms(2)* **by** *auto*
have *2*: *sdescending (smap f (v ## gtrace r v))*
using *1 assms(3)*
proof (*coinduction arbitrary*: *r v rule*: *sdescending.coinduct*)
case *sdescending*
obtain *u s* **where** *1*: *r = u ## s* **using** *stream.exhaust* **by** *blast*
have *2*: *v ∈ gunodes A w* **using** *sdescending(1)* **by** *simp*
have *3*: *gurun A w (u ## s) v* **using** *sdescending(2) 1* **by** *auto*
have *4*: *u ∈ gusuccessors A w v* **using** *3* **by** *auto*
have *5*: *u ∈ gureachable A w v* **using** *graph.reachable-successors 4* **by** *blast*
show *?case*
unfolding *1*
proof (*intro exI conjI disjI1*)
show *f u ≤ f v* **using** *0 2 4* **by** *this*
show *shd (u ## gtrace s u) ∈ gunodes A w* **using** *2 5* **by** *auto*
show *gurun A w s u* **using** *3* **by** *auto*
qed *auto*
qed
obtain *s k* **where** *3*: *smap f (v ## gtrace r v) = s @− sconst k*
using *sdescending-stuck*[*OF 2*] **by** *metis*
have *gtrace (sdrop (Suc (length s)) r) (gtarget (stake (Suc (length s)) r) v) = sdrop (Suc (length s)) (gtrace r v)*
using *sscan-sdrop* **by** *rule*
also have *smap f ... = sdrop (length s) (smap f (v ## gtrace r v))*

by (*metis 3 id-apply sdrop-simps*(*2*) *sdrop-smap sdrop-stl shift-eq siterate.simps*(*2*) *stream.sel*(*2*))
also have ... = *sconst k* **unfolding** *3* **using** *shift-eq* **by** *metis*
finally show *?thesis* **using** *that* **by** *blast*
qed

lemma *ranking-stuck-odd*:
assumes *ranking A w f*
assumes *v* ∈ *gunodes A w gurun A w r v*
obtains *n*
where *Ball* (*sset* (*smap f* (*gtrace* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)))) *odd*
proof −
obtain *n k* **where** *1*: *smap f* (*gtrace* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)) = *sconst k*
using *ranking-stuck assms* **by** *this*
have *2*: *gtarget* (*stake n r*) *v* ∈ *gunodes A w*
using *assms*(*2*, *3*) **by** (*simp add*: *graph.nodes-target graph.run-stake*)
have *3*: *gurun A w* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)
using *assms*(*2*, *3*) **by** (*simp add*: *graph.run-sdrop*)
have *4*: *odd k* **using** *1 2 3 assms*(*1*) **unfolding** *ranking-def* **by** *meson*
have *5*: *Ball* (*sset* (*smap f* (*gtrace* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)))) *odd*
unfolding *1* **using** *4* **by** *simp*
show *?thesis* **using** *that 5* **by** *this*
qed

lemma *ranking-language*:
assumes *ranking A w f*
shows *w* ∉ *language A*
proof
assume *1*: *w* ∈ *language A*
obtain *r p* **where** *2*: *run A* (*w* ||| *r*) *p p* ∈ *initial A infs* (*accepting A*) (*p* ## *r*) **using** *1* **by** *rule*
let *?r* = *fromN 1* ||| *r*
let *?v* = (*0*, *p*)
have *3*: *?v* ∈ *gunodes A w gurun A w ?r ?v* **using** *2*(*1*, *2*) **by** (*auto intro*: *run-grun*)

obtain *n* **where** *4*: *Ball* (*sset* (*smap f* (*gtrace* (*sdrop n ?r*) (*gtarget* (*stake n ?r*) *?v*)))) *odd*
using *ranking-stuck-odd assms 3* **by** *this*
let *?s* = *stake n ?r*
let *?t* = *sdrop n ?r*
let *?u* = *gtarget ?s ?v*

have *sset* (*gtrace ?t ?u*) ⊆ *gureachable A w ?v*
proof (*intro graph.reachable-trace graph.reachable-target graph.reachable.reflexive*)
show *gupath A w ?s ?v* **using** *graph.run-stake 3*(*2*) **by** *this*
show *gurun A w ?t ?u* **using** *graph.run-sdrop 3*(*2*) **by** *this*
qed

also have ... $\subseteq$ *gunodes A w* **using** *3(1)* **by** *blast*
finally have *7*: *sset* (*gtrace ?t ?u*) $\subseteq$ *gunodes A w* **by** *this*
have *8*: $\bigwedge$ *p*. *p* $\in$ *gunodes A w* $\Longrightarrow$ *gaccepting A p* $\Longrightarrow$ *even* (*f p*)
using *assms* **unfolding** *ranking-def* **by** *auto*
have *9*: $\bigwedge$ *p*. *p* $\in$ *sset* (*gtrace ?t ?u*) $\Longrightarrow$ *gaccepting A p* $\Longrightarrow$ *even* (*f p*) **using** *7 8* **by** *auto*

have *19*: *infs* (*accepting A*) (*smap snd ?r*) **using** *2(3)* **by** *simp*
have *18*: *infs* (*gaccepting A*) *?r* **using** *19* **by** *simp*
have *17*: *infs* (*gaccepting A*) (*gtrace ?r ?v*) **using** *18* **unfolding** *gtrace-alt-def* **by** *this*
have *16*: *infs* (*gaccepting A*) (*gtrace* (*?s* @− *?t*) *?v*) **using** *17* **unfolding** *stake-sdrop* **by** *this*
have *15*: *infs* (*gaccepting A*) (*gtrace ?t ?u*) **using** *16* **by** *simp*
have *13*: *infs* (*even* $\circ$ *f*) (*gtrace ?t ?u*) **using** *infs-mono*[*OF - 15*] *9* **by** *simp*
have *12*: *infs even* (*smap f* (*gtrace ?t ?u*)) **using** *13* **by** (*simp add*: *comp-def*)
have *11*: *Bex* (*sset* (*smap f* (*gtrace ?t ?u*))) *even* **using** *12 infs-any* **by** *metis*

show *False* **using** *4 11* **by** *auto*
qed

3.3 Word not in Language Implies Ranking

3.3.1 Removal of Endangered Nodes

definition *clean* :: ('*label*, '*state*) *nba* $\Rightarrow$ '*label stream* $\Rightarrow$ '*state node set* $\Rightarrow$ '*state node set* **where**
clean A w V $\equiv$ {*v* $\in$ *V*. *infinite* (*greachable A w V v*)}

lemma *clean-decreasing*: *clean A w V* $\subseteq$ *V* **unfolding** *clean-def* **by** *auto*
lemma *clean-successors*:
assumes *v* $\in$ *V u* $\in$ *gusuccessors A w v*
shows *u* $\in$ *clean A w V* $\Longrightarrow$ *v* $\in$ *clean A w V*
proof −
assume *1*: *u* $\in$ *clean A w V*
have *2*: *u* $\in$ *V infinite* (*greachable A w V u*) **using** *1* **unfolding** *clean-def* **by** *auto*
have *3*: *u* $\in$ *greachable A w V v* **using** *graph.reachable.execute assms(2) 2(1)* **by** *blast*
have *4*: *greachable A w V u* $\subseteq$ *greachable A w V v* **using** *3* **by** *blast*
have *5*: *infinite* (*greachable A w V v*) **using** *2(2) 4* **by** (*simp add*: *infinite-super*)
show *v* $\in$ *clean A w V* **unfolding** *clean-def* **using** *assms(1) 5* **by** *simp*
qed

3.3.2 Removal of Safe Nodes

definition *prune* :: ('*label*, '*state*) *nba* $\Rightarrow$ '*label stream* $\Rightarrow$ '*state node set* $\Rightarrow$ '*state node set* **where**
prune A w V $\equiv$ {*v* $\in$ *V*. $\exists$ *u* $\in$ *greachable A w V v*. *gaccepting A u*}

lemma *prune-decreasing*: *prune A w V* $\subseteq$ *V* **unfolding** *prune-def* **by** *auto*
lemma *prune-successors*:
assumes *v* $\in$ *V u* $\in$ *gusuccessors A w v*
shows *u* $\in$ *prune A w V* $\Longrightarrow$ *v* $\in$ *prune A w V*
proof −
assume *1*: *u* $\in$ *prune A w V*
have *2*: *u* $\in$ *V* $\exists$ *x* $\in$ *greachable A w V u. gaccepting A x* **using** *1* **unfolding** *prune-def* **by** *auto*
have *3*: *u* $\in$ *greachable A w V v* **using** *graph.reachable.execute assms(2) 2(1)* **by** *blast*
have *4*: *greachable A w V u* $\subseteq$ *greachable A w V v* **using** *3* **by** *blast*
show *v* $\in$ *prune A w V* **unfolding** *prune-def* **using** *assms(1) 2(2) 4* **by** *auto*
qed

3.3.3 Run Graph Interation

definition *graph* :: (*'label*, *'state*) *nba* $\Rightarrow$ *'label stream* $\Rightarrow$ *nat* $\Rightarrow$ *'state node set* **where**
graph A w k $\equiv$ *alternate* (*clean A w*) (*prune A w*) *k* (*gunodes A w*)

abbreviation *level A w k l* $\equiv$ {*v* $\in$ *graph A w k. fst v* = *l*}

lemma *graph-0*[*simp*]: *graph A w 0* = *gunodes A w* **unfolding** *graph-def* **by** *simp*
lemma *graph-Suc*[*simp*]: *graph A w* (*Suc k*) = (*if even k then clean A w else prune A w*) (*graph A w k*)
unfolding *graph-def* **by** *simp*

lemma *graph-antimono*: *antimono* (*graph A w*)
using *alternate-antimono clean-decreasing prune-decreasing*
unfolding *monotone-def le-fun-def graph-def*
by *metis*
lemma *graph-nodes*: *graph A w k* $\subseteq$ *gunodes A w* **using** *graph-0 graph-antimono le0 antimonoD* **by** *metis*
lemma *graph-successors*:
assumes *v* $\in$ *gunodes A w u* $\in$ *gusuccessors A w v*
shows *u* $\in$ *graph A w k* $\Longrightarrow$ *v* $\in$ *graph A w k*
using *assms*
proof (*induct k arbitrary*: *u v*)
case *0*
show *?case* **using** *0(2)* **by** *simp*
next
case (*Suc k*)
have *1*: *v* $\in$ *graph A w k* **using** *Suc* **using** *antimono-iff-le-Suc graph-antimono rev-subsetD* **by** *blast*
show *?case* **using** *Suc(2) clean-successors*[*OF 1 Suc(4)*] *prune-successors*[*OF 1 Suc(4)*] **by** *auto*
qed

lemma *graph-level-finite*:
assumes *finite* (*nodes A*)
shows *finite* (*level A w k l*)
proof −
have *level A w k l* ⊆ {*v* ∈ *gunodes A w*. *fst v* = *l*} **by** (*simp add*: *graph-nodes subset-CollectI*)
also have {*v* ∈ *gunodes A w*. *fst v* = *l*} ⊆ {*l*} × *nodes A* **using** *gnodes-nodes* **by** *force*
also have *finite* ({*l*} × *nodes A*) **using** *assms*(*1*) **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *find-safe*:
assumes *w* ∉ *language A*
assumes *V* ≠ {} *V* ⊆ *gunodes A w*
assumes ⋀ *v*. *v* ∈ *V* ⟹ *gsuccessors A w V v* ≠ {}
obtains *v*
where *v* ∈ *V* ∀ *u* ∈ *greachable A w V v*. ¬ *gaccepting A u*
proof (*rule ccontr*)
assume *1*: ¬ *thesis*
have *2*: ⋀ *v*. *v* ∈ *V* ⟹ ∃ *u* ∈ *greachable A w V v*. *gaccepting A u* **using** *that 1* **by** *auto*
have *3*: ⋀ *r v*. *v* ∈ *initial A* ⟹ *run A* (*w* ||| *r*) *v* ⟹ *fins* (*accepting A*) *r* **using** *assms*(*1*) **by** *auto*
obtain *v* **where** *4*: *v* ∈ *V* **using** *assms*(*2*) **by** *force*
obtain *x* **where** *5*: *x* ∈ *greachable A w V v gaccepting A x* **using** *2 4* **by** *blast*
obtain *y* **where** *50*: *gpath A w V y v x* = *gtarget y v* **using** *5*(*1*) **by** *rule*
obtain *r* **where** *6*: *grun A w V r x infs* (λ *x*. *x* ∈ *V* ∧ *gaccepting A x*) *r*
proof (*rule graph.recurring-condition*)
show *x* ∈ *V* ∧ *gaccepting A x* **using** *greachable-subset 4 5* **by** *blast*
next
fix *v*
assume *1*: *v* ∈ *V* ∧ *gaccepting A v*
obtain *v*′ **where** *20*: *v*′ ∈ *gsuccessors A w V v* **using** *assms*(*4*) *1* **by** (*meson IntE equals0I*)
have *21*: *v*′ ∈ *V* **using** *20* **by** *auto*
have *22*: ∃ *u* ∈ *greachable A w V v*′. *u* ∈ *V* ∧ *gaccepting A u*
using *greachable-subset 2 21* **by** *blast*
obtain *r* **where** *30*: *gpath A w V r v*′ *gtarget r v*′ ∈ *V* ∧ *gaccepting A* (*gtarget r v*′)
using *22* **by** *blast*
show ∃ *r*. *r* ≠ [] ∧ *gpath A w V r v* ∧ *gtarget r v* ∈ *V* ∧ *gaccepting A* (*gtarget r v*)
proof (*intro exI conjI*)
show *v*′ # *r* ≠ [] **by** *simp*
show *gpath A w V* (*v*′ # *r*) *v* **using** *20 30* **by** *auto*
show *gtarget* (*v*′ # *r*) *v* ∈ *V* **using** *30* **by** *simp*
show *gaccepting A* (*gtarget* (*v*′ # *r*) *v*) **using** *30* **by** *simp*
qed

qed *auto*
obtain *u* **where** *100*: *u* ∈ *ginitial A v* ∈ *gureachable A w u* **using** *4 assms*(*3*) **by** *blast*
have *101*: *gupath A w y v* **using** *gpath-subset 50*(*1*) *subset-UNIV* **by** *this*
have *102*: *gurun A w r x* **using** *grun-subset 6*(*1*) *subset-UNIV* **by** *this*
obtain *t* **where** *103*: *gupath A w t u v* = *gtarget t u* **using** *100*(*2*) **by** *rule*
have *104*: *gurun A w* (*t* @− *y* @− *r*) *u* **using** *101 102 103 50*(*2*) **by** *auto*
obtain *s q* **where** *7*: *t* @− *y* @− *r* = *fromN* (*Suc 0*) ||| *s u* = (*0*, *q*)
using *grun-elim*[*OF 104*] *100*(*1*) **by** *blast*
have *8*: *run A* (*w* ||| *s*) *q* **using** *grun-run*[*OF 104*[*unfolded 7*]] **by** *simp*
have *9*: *q* ∈ *initial A* **using** *100*(*1*) *7*(*2*) **by** *auto*
have *91*: *sset* (*trace* (*w* ||| *s*) *q*) ⊆ *reachable A q*
using *nba.reachable-trace nba.reachable.reflexive 8* **by** *this*
have *10*: *fins* (*accepting A*) *s* **using** *3 9 8* **by** *this*
have *12*: *infs* (*gaccepting A*) *r* **using** *infs-mono*[*OF - 6*(*2*)] **by** *simp*
have *s* = *smap snd* (*t* @− *y* @− *r*) **unfolding** *7*(*1*) **by** *simp*
also have *infs* (*accepting A*) ... **using** *12* **by** (*simp add*: *comp-def*)
finally have *13*: *infs* (*accepting A*) *s* **by** *this*
show *False* **using** *10 13* **by** *simp*
qed

lemma *remove-run*:
assumes *finite* (*nodes A*) *w* ∉ *language A*
assumes *V* ⊆ *gunodes A w clean A w V* ≠ {}
obtains *v r*
where
grun A w V r v
sset (*gtrace r v*) ⊆ *clean A w V*
sset (*gtrace r v*) ⊆ − *prune A w* (*clean A w V*)
proof −
obtain *u* **where** *1*: *u* ∈ *clean A w V* ∀ *x* ∈ *greachable A w* (*clean A w V*) *u*. ¬ *gaccepting A x*
proof (*rule find-safe*)
show *w* ∉ *language A* **using** *assms*(*2*) **by** *this*
show *clean A w V* ≠ {} **using** *assms*(*4*) **by** *this*
show *clean A w V* ⊆ *gunodes A w* **using** *assms*(*3*) **by** (*meson clean-decreasing subset-iff*)
next
fix *v*
assume *1*: *v* ∈ *clean A w V*
have *2*: *v* ∈ *V* **using** *1 clean-decreasing* **by** *blast*
have *3*: *infinite* (*greachable A w V v*) **using** *1 clean-def* **by** *auto*
have *gsuccessors A w V v* ⊆ *gusuccessors A w v* **by** *auto*
also have *finite* ... **using** *2 assms*(*1*, *3*) *finite-nodes-gsuccessors* **by** *blast*
finally have *4*: *finite* (*gsuccessors A w V v*) **by** *this*
have *5*: *infinite* (*insert v* (⋃((*greachable A w V*) ‘ (*gsuccessors A w V v*))))
using *graph.reachable-step 3* **by** *metis*
obtain *u* **where** *6*: *u* ∈ *gsuccessors A w V v infinite* (*greachable A w V u*) **using** *4 5* **by** *auto*

```
    have 7: u ∈ clean A w V using 6 unfolding clean-def by auto
    show gsuccessors A w (clean A w V) v ≠ {} using 6(1) 7 by auto
   qed auto
   have 2: u ∈ V using 1(1) unfolding clean-def by auto
  have 3: infinite (greachable A w V u) using 1(1) unfolding clean-def by simp
   have 4: finite (gsuccessors A w V v) if v ∈ greachable A w V u for v
   proof −
    have 1: v ∈ V using that greachable-subset 2 by blast
    have gsuccessors A w V v ⊆ gusuccessors A w v by auto
    also have finite ... using 1 assms(1, 3) finite-nodes-gsuccessors by blast
    finally show ?thesis by this
   qed
   obtain r where 5: grun A w V r u using graph.koenig[OF 3 4] by this
   have 6: greachable A w V u ⊆ V using 2 greachable-subset by blast
   have 7: sset (gtrace r u) ⊆ V
    using graph.reachable-trace[OF graph.reachable.reflexive 5(1)] 6 by blast
   have 8: sset (gtrace r u) ⊆ clean A w V
    unfolding clean-def using 7 infinite-greachable-gtrace[OF 5(1)] by auto
   have 9: sset (gtrace r u) ⊆ greachable A w (clean A w V) u
   using 5 8 by (metis graph.reachable.reflexive graph.reachable-trace grun-subset)
   show ?thesis
   proof
    show grun A w V r u using 5(1) by this
    show sset (gtrace r u) ⊆ clean A w V using 8 by this
    show sset (gtrace r u) ⊆ − prune A w (clean A w V)
    proof (intro subsetI ComplI)
     fix p
     assume 10: p ∈ sset (gtrace r u) p ∈ prune A w (clean A w V)
     have 20: ∃ x ∈ greachable A w (clean A w V) p. gaccepting A x
      using 10(2) unfolding prune-def by auto
     have 30: greachable A w (clean A w V) p ⊆ greachable A w (clean A w V)
u
      using 10(1) 9 by blast
     show False using 1(2) 20 30 by force
    qed
   qed
  qed

  lemma level-bounded:
   assumes finite (nodes A) w ∉ language A
   obtains n
   where ⋀ l. l ≥ n ⟹ card (level A w (2 * k) l) ≤ card (nodes A) − k
  proof (induct k arbitrary: thesis)
   case (0)
   show ?case
   proof (rule 0)
    fix l :: nat
    have finite ({l} × nodes A) using assms(1) by simp
    also have level A w 0 l ⊆ {l} × nodes A using gnodes-nodes by force
```

also (*card-mono*) **have** *card* ... = *card* (*nodes A*) **using** *assms*(*1*) **by** *simp*
finally show *card* (*level A w* (*2* ∗ *0*) *l*) ≤ *card* (*nodes A*) − *0* **by** *simp*
qed
next
case (*Suc k*)
show *?case*
proof (*cases graph A w* (*Suc* (*2* ∗ *k*)) = {})
case *True*
have *3*: *graph A w* (*2* ∗ *Suc k*) = {} **using** *True prune-decreasing* **by** *simp blast*
show *?thesis* **using** *Suc*(*2*) *3* **by** *simp*
next
case *False*
obtain *v r* **where** *1*:
grun A w (*graph A w* (*2* ∗ *k*)) *r v*
sset (*gtrace r v*) ⊆ *graph A w* (*Suc* (*2* ∗ *k*))
sset (*gtrace r v*) ⊆ − *graph A w* (*Suc* (*Suc* (*2* ∗ *k*)))
proof (*rule remove-run*)
show *finite* (*nodes A*) *w* ∉ *language A* **using** *assms* **by** *this*
show *clean A w* (*graph A w* (*2* ∗ *k*)) ≠ {} **using** *False* **by** *simp*
show *graph A w* (*2* ∗ *k*) ⊆ *gunodes A w* **using** *graph-nodes* **by** *this*
qed *auto*
obtain *l q* **where** *2*: *v* = (*l*, *q*) **by** *force*
obtain *n* **where** *90*: ⋀ *l*. *n* ≤ *l* ⟹ *card* (*level A w* (*2* ∗ *k*) *l*) ≤ *card* (*nodes A*) − *k*
using *Suc*(*1*) **by** *blast*
show *?thesis*
proof (*rule Suc*(*2*))
fix *j*
assume *100*: *n* + *Suc l* ≤ *j*
have *6*: *graph A w* (*Suc* (*Suc* (*2* ∗ *k*))) ⊆ *graph A w* (*Suc* (*2* ∗ *k*))
using *graph-antimono antimono-iff-le-Suc* **by** *blast*
have *101*: *gtrace r v* !! (*j* − *Suc l*) ∈ *graph A w* (*Suc* (*2* ∗ *k*)) **using** *1*(*2*) *snth-sset* **by** *auto*
have *102*: *gtrace r v* !! (*j* − *Suc l*) ∉ *graph A w* (*Suc* (*Suc* (*2* ∗ *k*))) **using** *1*(*3*) *snth-sset* **by** *blast*
have *103*: *gtrace r v* !! (*j* − *Suc l*) ∈ *level A w* (*Suc* (*2* ∗ *k*)) *j*
using *1*(*1*) *100 101 2* **by** (*auto elim*: *grun-elim*)
have *104*: *gtrace r v* !! (*j* − *Suc l*) ∉ *level A w* (*Suc* (*Suc* (*2* ∗ *k*))) *j* **using** *100 102* **by** *simp*
have *level A w* (*2* ∗ *Suc k*) *j* = *level A w* (*Suc* (*Suc* (*2* ∗ *k*))) *j* **by** *simp*
also have ... ⊂ *level A w* (*Suc* (*2* ∗ *k*)) *j* **using** *103 104 6* **by** *blast*
also have ... ⊆ *level A w* (*2* ∗ *k*) *j* **by** (*simp add*: *Collect-mono clean-def*)
finally have *105*: *level A w* (*2* ∗ *Suc k*) *j* ⊂ *level A w* (*2* ∗ *k*) *j* **by** *this*
have *card* (*level A w* (*2* ∗ *Suc k*) *j*) < *card* (*level A w* (*2* ∗ *k*) *j*)
using *assms*(*1*) *105* **by** (*simp add*: *graph-level-finite psubset-card-mono*)
also have ... ≤ *card* (*nodes A*) − *k* **using** *90 100* **by** *simp*
finally show *card* (*level A w* (*2* ∗ *Suc k*) *j*) ≤ *card* (*nodes A*) − *Suc k* **by** *simp*

```
    qed
  qed
qed
lemma graph-empty:
  assumes finite (nodes A) w ∉ language A
  shows graph A w (Suc (2 * card (nodes A))) = {}
proof −
  obtain n where 1: ⋀ l. l ≥ n ⟹ card (level A w (2 * card (nodes A)) l) = 0
    using level-bounded[OF assms(1, 2), of card (nodes A)] by auto
  have graph A w (2 * card (nodes A)) =
    (⋃ l ∈ {..< n}. level A w (2 * card (nodes A)) l) ∪
    (⋃ l ∈ {n ..}. level A w (2 * card (nodes A)) l)
    by auto
  also have (⋃ l ∈ {n ..}. level A w (2 * card (nodes A)) l) = {}
    using graph-level-finite assms(1) 1 by fastforce
  also have finite ((⋃ l ∈ {..< n}. level A w (2 * card (nodes A)) l) ∪ {})
    using graph-level-finite assms(1) by auto
  finally have 100: finite (graph A w (2 * card (nodes A))) by this
  have 101: finite (greachable A w (graph A w (2 * card (nodes A))) v) for v
    using 100 greachable-subset[of A w graph A w (2 * card (nodes A)) v]
    using finite-insert infinite-super by auto
  show ?thesis using 101 by (simp add: clean-def)
qed
lemma graph-le:
  assumes finite (nodes A) w ∉ language A
  assumes v ∈ graph A w k
  shows k ≤ 2 * card (nodes A)
  using graph-empty graph-antimono assms
  by (metis Suc-leI empty-iff monotone-def not-le-imp-less rev-subsetD)
```

3.4 Node Ranks

```
definition rank :: ('label, 'state) nba ⇒ 'label stream ⇒ 'state node ⇒ nat where
  rank A w v ≡ GREATEST k. v ∈ graph A w k

lemma rank-member:
  assumes finite (nodes A) w ∉ language A v ∈ gunodes A w
  shows v ∈ graph A w (rank A w v)
unfolding rank-def
proof (rule GreatestI-nat)
  show v ∈ graph A w 0 using assms(3) by simp
  show k ≤ 2 * card (nodes A) if v ∈ graph A w k for k
    using graph-le assms(1, 2) that by blast
qed
lemma rank-removed:
  assumes finite (nodes A) w ∉ language A
  shows v ∉ graph A w (Suc (rank A w v))
proof
  assume v ∈ graph A w (Suc (rank A w v))
```

```
    then have 2: Suc (rank A w v) ≤ rank A w v
      unfolding rank-def using Greatest-le-nat graph-le assms by metis
    then show False by auto
  qed
  lemma rank-le:
    assumes finite (nodes A) w ∉ language A
    assumes v ∈ gunodes A w u ∈ gusuccessors A w v
    shows rank A w u ≤ rank A w v
  unfolding rank-def
  proof (rule Greatest-le-nat)
    have 1: u ∈ gureachable A w v using graph.reachable-successors assms(4) by
blast
    have 2: u ∈ gunodes A w using assms(3) 1 by auto
    show v ∈ graph A w (GREATEST k. u ∈ graph A w k)
    unfolding rank-def[symmetric]
    proof (rule graph-successors)
      show v ∈ gunodes A w using assms(3) by this
      show u ∈ gusuccessors A w v using assms(4) by this
     show u ∈ graph A w (rank A w u) using rank-member assms(1, 2) 2 by this
    qed
    show k ≤ 2 * card (nodes A) if v ∈ graph A w k for k
      using graph-le assms(1, 2) that by blast
  qed

  lemma language-ranking:
    assumes finite (nodes A) w ∉ language A
    shows ranking A w (rank A w)
  unfolding ranking-def
  proof (intro conjI ballI allI impI)
    fix v
    assume 1: v ∈ gunodes A w
    have 2: v ∈ graph A w (rank A w v) using rank-member assms 1 by this
    show rank A w v ≤ 2 * card (nodes A) using graph-le assms 2 by this
  next
    fix v u
    assume 1: v ∈ gunodes A w u ∈ gusuccessors A w v
    show rank A w u ≤ rank A w v using rank-le assms 1 by this
  next
    fix v
    assume 1: v ∈ gunodes A w gaccepting A v
    have 2: v ∈ graph A w (rank A w v) using rank-member assms 1(1) by this
    have 3: v ∉ graph A w (Suc (rank A w v)) using rank-removed assms by this
     have 4: v ∈ prune A w (graph A w (rank A w v)) using 2 1(2) unfolding
prune-def by auto
    have 5: graph A w (Suc (rank A w v)) ≠ prune A w (graph A w (rank A w v))
using 3 4 by blast
    show even (rank A w v) using 5 by auto
  next
    fix v r k
```

assume *1*: $v \in$ *gunodes A w gurun A w r v smap* (*rank A w*) (*gtrace r v*) = *sconst k*
have *sset* (*gtrace r v*) ⊆ *gureachable A w v*
using *1*(*2*) **by** (*metis graph.reachable.reflexive graph.reachable-trace*)
then have *6*: *sset* (*gtrace r v*) ⊆ *gunodes A w* **using** *1*(*1*) **by** *blast*
have *60*: *rank A w* ‘ *sset* (*gtrace r v*) ⊆ {*k*}
using *1*(*3*) **by** (*metis equalityD1 sset-sconst stream.set-map*)
have *50*: *sset* (*gtrace r v*) ⊆ *graph A w k*
using *rank-member*[*OF assms*] *subsetD*[*OF 6*] *60* **unfolding** *image-subset-iff* **by** *auto*
have *70*: *grun A w* (*graph A w k*) *r v* **using** *grun-subset 1*(*2*) *50* **by** *this*
have *7*: *sset* (*gtrace r v*) ⊆ *clean A w* (*graph A w k*)
unfolding *clean-def* **using** *50 infinite-greachable-gtrace*[*OF 70*] **by** *auto*
have *8*: *sset* (*gtrace r v*) ∩ *graph A w* (*Suc k*) = {} **using** *rank-removed*[*OF assms*] *60* **by** *blast*
have *9*: *sset* (*gtrace r v*) ≠ {} **using** *stream.set-sel*(*1*) **by** *auto*
have *10*: *graph A w* (*Suc k*) ≠ *clean A w* (*graph A w k*) **using** *7 8 9* **by** *blast*
show *odd k* **using** *10* **unfolding** *graph-Suc* **by** *auto*
qed

3.5 Correctness Theorem

theorem *language-ranking-iff*:
assumes *finite* (*nodes A*)
shows *w* ∉ *language A* ⟷ (∃ *f*. *ranking A w f*)
using *ranking-language language-ranking assms* **by** *blast*

end

4 Complementation

theory *Complementation*
imports
Transition-Systems-and-Automata.Maps
Ranking
begin

4.1 Level Rankings and Complementation States

type-synonym *'state lr* = *'state* ⇀ *nat*

definition *lr-succ* :: (*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state lr* ⇒ *'state lr set* **where**
lr-succ A a f ≡ {*g*.
dom g = ⋃ (*transition A a* ‘ *dom f*) ∧
(∀ *p* ∈ *dom f*. ∀ *q* ∈ *transition A a p*. *the* (*g q*) ≤ *the* (*f p*)) ∧
(∀ *q* ∈ *dom g*. *accepting A q* ⟶ *even* (*the* (*g q*)))}

type-synonym *'state st* = *'state set*

definition *st-succ* :: (*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state lr* ⇒ *'state st* ⇒ *'state st* **where**
st-succ A a g P ≡ {*q* ∈ *if P* = {} *then dom g else* ⋃ (*transition A a* ‘ *P*). *even* (*the* (*g q*))}

type-synonym *'state cs* = *'state lr* × *'state st*

definition *complement-succ* :: (*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state cs* ⇒ *'state cs set* **where**
complement-succ A a ≡ λ (*f*, *P*). {(*g*, *st-succ A a g P*) |*g*. *g* ∈ *lr-succ A a f*}

definition *complement* :: (*'label*, *'state*) *nba* ⇒ (*'label*, *'state cs*) *nba* **where**
complement A ≡ *nba*
(*alphabet A*)
({*const* (*Some* (*2* ∗ *card* (*nodes A*))) |‘ *initial A*} × {{}})
(*complement-succ A*)
(λ (*f*, *P*). *P* = {})

lemma *dom-nodes*:
assumes *fP* ∈ *nodes* (*complement A*)
shows *dom* (*fst fP*) ⊆ *nodes A*
using *assms* **unfolding** *complement-def complement-succ-def lr-succ-def* **by** (*induct*) (*auto*, *blast*)
lemma *ran-nodes*:
assumes *fP* ∈ *nodes* (*complement A*)
shows *ran* (*fst fP*) ⊆ {*0* .. *2* ∗ *card* (*nodes A*)}
using *assms*
proof *induct*
case (*initial fP*)
show *?case*
using *initial* **unfolding** *complement-def* **by** (*auto*) (*metis eq-refl option.inject ran-restrictD*)
next
case (*execute fP agQ*)
obtain *f P* **where** *1*: *fP* = (*f*, *P*) **by** *force*
have *2*: *ran f* ⊆ {*0* .. *2* ∗ *card* (*nodes A*)} **using** *execute*(*2*) **unfolding** *1* **by** *auto*
obtain *a g Q* **where** *3*: *agQ* = (*a*, (*g*, *Q*)) **using** *prod-cases3* **by** *this*
have *4*: *p* ∈ *dom f* ⟹ *q* ∈ *transition A a p* ⟹ *the* (*g q*) ≤ *the* (*f p*) **for** *p q*
using *execute*(*3*)
unfolding *1 3 complement-def nba.simps complement-succ-def lr-succ-def*
by *simp*
have *8*: *dom g* = ⋃((*transition A a*) ‘ (*dom f*))
using *execute*(*3*)
unfolding *1 3 complement-def nba.simps complement-succ-def lr-succ-def*
by *simp*
show *?case*
unfolding *1 3 ran-def*
proof *safe*

```
      fix q k
      assume 5: fst (snd (a, (g, Q))) q = Some k
      have 6: q ∈ dom g using 5 by auto
     obtain p where 7: p ∈ dom f q ∈ transition A a p using 6 unfolding 8 by
auto
      have k = the (g q) using 5 by auto
      also have ... ≤ the (f p) using 4 7 by this
      also have ... ≤ 2 * card (nodes A) using 2 7(1) by (simp add: domD ranI
subset-eq)
      finally show k ∈ {0 .. 2 * card (nodes A)} by auto
    qed
  qed
  lemma states-nodes:
    assumes fP ∈ nodes (complement A)
    shows snd fP ⊆ nodes A
  using assms
  proof induct
    case (initial fP)
    show ?case using initial unfolding complement-def by auto
  next
    case (execute fP agQ)
    obtain f P where 1: fP = (f, P) by force
    have 2: P ⊆ nodes A using execute(2) unfolding 1 by auto
    obtain a g Q where 3: agQ = (a, (g, Q)) using prod-cases3 by this
     have 11: a ∈ alphabet A using execute(3) unfolding 3 complement-def by
auto
    have 10: (g, Q) ∈ nodes (complement A) using execute(1, 3) unfolding 1 3
by auto
    have 4: dom g ⊆ nodes A using dom-nodes[OF 10] by simp
    have 5: ⋃ (transition A a ' P) ⊆ nodes A using 2 11 by auto
    have 6: Q ⊆ nodes A
      using execute(3)
      unfolding 1 3 complement-def nba.simps complement-succ-def st-succ-def
      using 4 5
      by (auto split: if-splits)
    show ?case using 6 unfolding 3 by auto
  qed

  theorem complement-finite:
    assumes finite (nodes A)
    shows finite (nodes (complement A))
  proof −
    let ?lrs = {f. dom f ⊆ nodes A ∧ ran f ⊆ {0 .. 2 * card (nodes A)}}
    have 1: finite ?lrs using finite-set-of-finite-maps' assms by auto
    let ?states = Pow (nodes A)
    have 2: finite ?states using assms by simp
     have nodes (complement A) ⊆ ?lrs × ?states by (force dest: dom-nodes
ran-nodes states-nodes)
    also have finite ... using 1 2 by simp
```

```
  finally show ?thesis by this
 qed

 lemma complement-trace-snth:
  assumes run (complement A) (w ||| r) p
  defines m ≡ p ## trace (w ||| r) p
  obtains
   fst (m !! Suc k) ∈ lr-succ A (w !! k) (fst (m !! k))
   snd (m !! Suc k) = st-succ A (w !! k) (fst (m !! Suc k)) (snd (m !! k))
 proof
  have 1: r !! k ∈ transition (complement A) (w !! k) (m !! k) using nba.run-snth
assms by force
  show fst (m !! Suc k) ∈ lr-succ A (w !! k) (fst (m !! k))
   using assms(2) 1 unfolding complement-def complement-succ-def nba.trace-alt-def
by auto
  show snd (m !! Suc k) = st-succ A (w !! k) (fst (m !! Suc k)) (snd (m !! k))
   using assms(2) 1 unfolding complement-def complement-succ-def nba.trace-alt-def
by auto
 qed
```

4.2 Word in Complement Language Implies Ranking

```
 lemma complement-ranking:
  assumes w ∈ language (complement A)
  obtains f
  where ranking A w f
 proof −
  obtain r p where 1:
   run (complement A) (w ||| r) p
   p ∈ initial (complement A)
   infs (accepting (complement A)) (p ## r)
   using assms by rule
  let ?m = p ## r
  obtain 100:
   fst (?m !! Suc k) ∈ lr-succ A (w !! k) (fst (?m !! k))
   snd (?m !! Suc k) = st-succ A (w !! k) (fst (?m !! Suc k)) (snd (?m !! k))
  for k using complement-trace-snth 1(1) unfolding nba.trace-alt-def szip-smap-snd
by metis
  define f where f ≡ λ (k, q). the (fst (?m !! k) q)
  define P where P k ≡ snd (?m !! k) for k
  have 2: snd v ∈ dom (fst (?m !! fst v)) if v ∈ gunodes A w for v
  using that
  proof induct
   case (initial v)
   then show ?case using 1(2) unfolding complement-def by auto
  next
   case (execute v u)
   have snd u ∈ ⋃ (transition A (w !! fst v) ‘ dom (fst (?m !! fst v)))
    using execute(2, 3) by auto
```

```
    also have ... = dom (fst (?m !! Suc (fst v)))
      using 100 unfolding lr-succ-def by simp
    also have Suc (fst v) = fst u using execute(3) by auto
    finally show ?case by this
  qed
  have 3: f u ≤ f v if 10: v ∈ gunodes A w and 11: u ∈ gusuccessors A w v for
u v
  proof −
    have 15: snd u ∈ transition A (w !! fst v) (snd v) using 11 by auto
    have 16: snd v ∈ dom (fst (?m !! fst v)) using 2 10 by this
     have f u = the (fst (?m !! fst u) (snd u)) unfolding f-def by (simp add:
case-prod-beta)
    also have fst u = Suc (fst v) using 11 by auto
    also have the (fst (?m !! ...) (snd u)) ≤ the (fst (?m !! fst v) (snd v))
      using 100 15 16 unfolding lr-succ-def by auto
    also have ... = f v unfolding f-def by (simp add: case-prod-beta)
    finally show f u ≤ f v by this
  qed
  have 4: ∃ l ≥ k. P l = {} for k
  proof −
    have 15: infs (λ (k, P). P = {}) ?m using 1(3) unfolding complement-def
by auto
    obtain l where 17: l ≥ k snd (?m !! l) = {} using 15 unfolding infs-snth
by force
    have 19: P l = {} unfolding P-def using 17 by auto
    show ?thesis using 19 17(1) by auto
  qed
  show ?thesis
  proof (rule that, unfold ranking-def, intro conjI ballI impI allI)
    fix v
    assume v ∈ gunodes A w
    then show f v ≤ 2 ∗ card (nodes A)
    proof induct
      case (initial v)
      then show ?case using 1(2) unfolding complement-def f-def by auto
    next
      case (execute v u)
      have f u ≤ f v using 3[OF execute(1)] execute(3) by simp
      also have ... ≤ 2 ∗ card (nodes A) using execute(2) by this
      finally show ?case by this
    qed
  next
    fix v u
    assume 10: v ∈ gunodes A w
    assume 11: u ∈ gusuccessors A w v
    show f u ≤ f v using 3 10 11 by this
  next
    fix v
    assume 10: v ∈ gunodes A w
```

assume *11*: *gaccepting A v*
show *even* (*f v*)
using *10*
proof *cases*
case (*initial*)
then show *?thesis* **using** *1*(*2*) **unfolding** *complement-def f-def* **by** *auto*
next
case (*execute u*)
have *12*: *snd v* ∈ *dom* (*fst* (*?m* !! *fst v*)) **using** *execute graph.nodes.execute 2* **by** *blast*
have *12*: *snd v* ∈ *dom* (*fst* (*?m* !! *Suc* (*fst u*))) **using** *12 execute*(*2*) **by** *auto*
have *13*: *accepting A* (*snd v*) **using** *11* **by** *auto*
have *f v* = *the* (*fst* (*?m* !! *fst v*) (*snd v*)) **unfolding** *f-def* **by** (*simp add*: *case-prod-beta*)
also have *fst v* = *Suc* (*fst u*) **using** *execute*(*2*) **by** *auto*
also have *even* (*the* (*fst* (*?m* !! *Suc* (*fst u*)) (*snd v*)))
using *100 12 13* **unfolding** *lr-succ-def* **by** *simp*
finally show *?thesis* **by** *this*
qed
next
fix *v s k*
assume *10*: *v* ∈ *gunodes A w*
assume *11*: *gurun A w s v*
assume *12*: *smap f* (*gtrace s v*) = *sconst k*
show *odd k*
proof
assume *13*: *even k*
obtain *t u* **where** *14*: *u* ∈ *ginitial A gupath A w t u v* = *gtarget t u* **using** *10* **by** *auto*
obtain *l* **where** *15*: *l* ≥ *length t P l* = {} **using** *4* **by** *auto*
have *30*: *gurun A w* (*t* @− *s*) *u* **using** *11 14* **by** *auto*
have *21*: *fst* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*) = *Suc l* **for** *l*
unfolding *sscan-snth*[*symmetric*] **using** *30 14*(*1*) **by** (*auto elim*!: *grun-elim*)
have *17*: *snd* (*gtarget* (*stake* (*Suc l* + *i*) (*t* @− *s*)) *u*) ∈ *P* (*Suc l* + *i*) **for** *i*
proof (*induct i*)
case (*0*)
have *20*: *gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u* ∈ *gunodes A w*
using *14 11* **by** (*force simp add*: *15*(*1*) *le-SucI graph.run-stake stake-shift*)
have *snd* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*) ∈
dom (*fst* (*?m* !! *fst* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*)))
using *2*[*OF 20*] **by** *this*
also have *fst* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*) = *Suc l* **using** *21* **by** *this*
finally have *22*: *snd* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*) ∈ *dom* (*fst* (*?m* !! *Suc l*)) **by** *this*
have *gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u* = *gtrace* (*t* @− *s*) *u* !! *l* **unfolding** *sscan-snth* **by** *rule*
also have ... = *gtrace s v* !! (*l* − *length t*) **using** *15*(*1*) **by** *simp*
also have *f* ... = *smap f* (*gtrace s v*) !! (*l* − *length t*) **by** *simp*

also have *smap f* (*gtrace s v*) = *sconst k* **unfolding** *12* **by** *rule*
also have *sconst k* !! (*l* − *length t*) = *k* **by** *simp*
finally have *23*: *even* (*f* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*)) **using** *13* **by** *simp*
have *snd* (*gtarget* (*stake* (*Suc l*) (*t* @− *s*)) *u*) ∈
{*p* ∈ *dom* (*fst* (*?m* !! *Suc l*)). *even* (*f* (*Suc l*, *p*))}
using *21 22 23* **by** (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq prod.collapse*)
also have ... = *st-succ A* (*w* !! *l*) (*fst* (*?m* !! *Suc l*)) (*P l*)
unfolding *15*(*2*) *st-succ-def f-def* **by** *simp*
also have ... = *P* (*Suc l*) **using** *100*(*2*) **unfolding** *P-def* **by** *rule*
finally show *?case* **by** *auto*
next
case (*Suc i*)
have *20*: *P* (*Suc l* + *i*) ≠ {} **using** *Suc* **by** *auto*
have *21*: *fst* (*gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u*) = *Suc l* + *Suc i*
using *21* **by** (*simp add*: *stake-shift*)
have *gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u* = *gtrace* (*t* @− *s*) *u* !! (*l* + *Suc i*)
unfolding *sscan-snth* **by** *simp*
also have ... ∈ *gusuccessors A w* (*gtarget* (*stake* (*Suc* (*l* + *i*)) (*t* @− *s*)) *u*)
using *graph.run-snth*[*OF 30*, *of l* + *Suc i*] **by** *simp*
finally have *220*: *snd* (*gtarget* (*stake* (*Suc* (*Suc l* + *i*)) (*t* @− *s*)) *u*) ∈
transition A (*w* !! (*Suc l* + *i*)) (*snd* (*gtarget* (*stake* (*Suc* (*l* + *i*)) (*t* @− *s*)) *u*))
using *21* **by** *auto*
have *22*: *snd* (*gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u*) ∈
⋃ (*transition A* (*w* !! (*Suc l* + *i*)) ‘ *P* (*Suc l* + *i*)) **using** *220 Suc* **by** *auto*
have *gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u* = *gtrace* (*t* @− *s*) *u* !! (*l* + *Suc i*)
unfolding *sscan-snth* **by** *simp*
also have ... = *gtrace s v* !! (*l* + *Suc i* − *length t*) **using** *15*(*1*)
by (*metis add.commute shift-snth-ge sscan-const trans-le-add2*)
also have *f* ... = *smap f* (*gtrace s v*) !! (*l* + *Suc i* − *length t*) **by** *simp*
also have *smap f* (*gtrace s v*) = *sconst k* **unfolding** *12* **by** *rule*
also have *sconst k* !! (*l* + *Suc i* − *length t*) = *k* **by** *simp*
finally have *23*: *even* (*f* (*gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u*)) **using** *13* **by** *auto*
have *snd* (*gtarget* (*stake* (*Suc l* + *Suc i*) (*t* @− *s*)) *u*) ∈
{*p* ∈ ⋃ (*transition A* (*w* !! (*Suc l* + *i*)) ‘ *P* (*Suc l* + *i*)). *even* (*f* (*Suc* (*Suc l* + *i*), *p*))}
using *21 22 23* **by** (*metis* (*mono-tags*) *add-Suc-right mem-Collect-eq prod.collapse*)
also have ... = *st-succ A* (*w* !! (*Suc l* + *i*)) (*fst* (*?m* !! *Suc* (*Suc l* + *i*))) (*P* (*Suc l* + *i*))
unfolding *st-succ-def f-def* **using** *20* **by** *simp*
also have ... = *P* (*Suc* (*Suc l* + *i*)) **unfolding** *100*(*2*)[*folded P-def*] **by** *rule*

```
      also have ... = P (Suc l + Suc i) by simp
      finally show ?case by this
    qed
    obtain l' where 16: l' ≥ Suc l P l' = {} using 4 by auto
    show False using 16 17 using nat-le-iff-add by auto
   qed
  qed
 qed
```

4.3 Ranking Implies Word in Complement Language

```
 definition reach where
  reach A w i ≡ {target r p |r p. path A r p ∧ p ∈ initial A ∧ map fst r = stake
i w}

 lemma reach-0[simp]: reach A w 0 = initial A unfolding reach-def by auto
 lemma reach-Suc-empty:
  assumes w !! n ∉ alphabet A
  shows reach A w (Suc n) = {}
 proof safe
  fix q
  assume 1: q ∈ reach A w (Suc n)
  obtain r p where 2: q = target r p path A r p p ∈ initial A map fst r = stake
(Suc n) w
    using 1 unfolding reach-def by blast
  have 3: path A (take n r @ drop n r) p using 2(2) by simp
  have 4: map fst r = stake n w @ [w !! n] using 2(4) stake-Suc by auto
  have 5: map snd r = take n (map snd r) @ [q] using 2(1, 4) 4
   by (metis One-nat-def Suc-inject Suc-neq-Zero Suc-pred append.right-neutral
       append-eq-conv-conj drop-map id-take-nth-drop last-ConsR last-conv-nth
length-0-conv
    length-map length-stake lessI nba.target-alt-def nba.states-alt-def zero-less-Suc)
  have 6: drop n r = [(w !! n, q)] using 4 5
  by (metis append-eq-conv-conj append-is-Nil-conv append-take-drop-id drop-map
     length-greater-0-conv length-stake stake-cycle-le stake-invert-Nil
     take-map zip-Cons-Cons zip-map-fst-snd)
  show q ∈ {} using assms 3 unfolding 6 by auto
 qed
 lemma reach-Suc-succ:
  assumes w !! n ∈ alphabet A
  shows reach A w (Suc n) = ⋃ (transition A (w !! n) ‘ reach A w n)
 proof safe
  fix q
  assume 1: q ∈ reach A w (Suc n)
  obtain r p where 2: q = target r p path A r p p ∈ initial A map fst r = stake
(Suc n) w
    using 1 unfolding reach-def by blast
  have 3: path A (take n r @ drop n r) p using 2(2) by simp
  have 4: map fst r = stake n w @ [w !! n] using 2(4) stake-Suc by auto
```

have *5*: *map snd r = take n (map snd r) @ [q]* **using** *2(1, 4) 4*
by (*metis One-nat-def Suc-inject Suc-neq-Zero Suc-pred append.right-neutral append-eq-conv-conj drop-map id-take-nth-drop last-ConsR last-conv-nth length-0-conv length-map length-stake lessI nba.target-alt-def nba.states-alt-def zero-less-Suc*)
have *6*: *drop n r = [(w !! n, q)]* **using** *4 5*
by (*metis append-eq-conv-conj append-is-Nil-conv append-take-drop-id drop-map length-greater-0-conv length-stake stake-cycle-le stake-invert-Nil take-map zip-Cons-Cons zip-map-fst-snd*)
show *q ∈ ⋃((transition A (w !! n) ‘ (reach A w n)))*
unfolding *reach-def*
proof (*intro UN-I CollectI exI conjI*)
show *target (take n r) p = target (take n r) p* **by** *rule*
show *path A (take n r) p* **using** *3* **by** *blast*
show *p ∈ initial A* **using** *2(3)* **by** *this*
show *map fst (take n r) = stake n w* **using** *2* **by** (*metis length-stake lessI nat.distinct(1) stake-cycle-le stake-invert-Nil take-map take-stake*)
show *q ∈ transition A (w !! n) (target (take n r) p)* **using** *3* **unfolding** *6* **by** *auto*
qed
next
fix *p q*
assume *1*: *p ∈ reach A w n q ∈ transition A (w !! n) p*
obtain *r x* **where** *2*: *p = target r x path A r x x ∈ initial A map fst r = stake n w*
using *1(1)* **unfolding** *reach-def* **by** *blast*
show *q ∈ reach A w (Suc n)*
unfolding *reach-def*
proof (*intro CollectI exI conjI*)
show *q = target (r @ [(w !! n, q)]) x* **using** *1 2* **by** *auto*
show *path A (r @ [(w !! n, q)]) x* **using** *assms 1(2) 2(1, 2)* **by** *auto*
show *x ∈ initial A* **using** *2(3)* **by** *this*
show *map fst (r @ [(w !! n, q)]) = stake (Suc n) w* **using** *1 2*
by (*metis eq-fst-iff list.simps(8) list.simps(9) map-append stake-Suc*)
qed
qed
lemma *reach-Suc*[*simp*]: *reach A w (Suc n) = (if w !! n ∈ alphabet A then ⋃ (transition A (w !! n) ‘ reach A w n) else {})*
using *reach-Suc-empty reach-Suc-succ* **by** *metis*
lemma *reach-nodes*: *reach A w i ⊆ nodes A* **by** (*induct i*) (*auto*)
lemma *reach-gunodes*: *{i} × reach A w i ⊆ gunodes A w*
by (*induct i*) (*auto intro*: *graph.nodes.execute*)

lemma *ranking-complement*:
assumes *finite (nodes A) w ∈ streams (alphabet A) ranking A w f*
shows *w ∈ language (complement A)*
proof −
define *f′* **where** *f′ ≡ λ (k, p). if k = 0 then 2 ∗ card (nodes A) else f (k, p)*

```
  have 0: ranking A w f'
  unfolding ranking-def
  proof (intro conjI ballI impI allI)
    show ⋀ v. v ∈ gunodes A w ⟹ f' v ≤ 2 * card (nodes A)
      using assms(3) unfolding ranking-def f'-def by auto
    show ⋀ v u. v ∈ gunodes A w ⟹ u ∈ gusuccessors A w v ⟹ f' u ≤ f' v
      using assms(3) unfolding ranking-def f'-def by fastforce
    show ⋀ v. v ∈ gunodes A w ⟹ gaccepting A v ⟹ even (f' v)
      using assms(3) unfolding ranking-def f'-def by auto
  next
   have 1: v ∈ gunodes A w ⟹ gurun A w r v ⟹ smap f (gtrace r v) = sconst
k ⟹ odd k
     for v r k using assms(3) unfolding ranking-def by meson
    fix v r k
    assume 2: v ∈ gunodes A w gurun A w r v smap f' (gtrace r v) = sconst k
    have 20: shd r ∈ gureachable A w v using 2
     by (auto) (metis graph.reachable.reflexive graph.reachable-trace gtrace-alt-def
subsetD shd-sset)
    obtain 3:
      shd r ∈ gunodes A w
      gurun A w (stl r) (shd r)
      smap f' (gtrace (stl r) (shd r)) = sconst k
    using 2 20 by (metis (no-types, lifting) eq-id-iff graph.nodes-trans graph.run-scons-elim
       siterate.simps(2) sscan.simps(2) stream.collapse stream.map-sel(2))
    have 4: k ≠ 0 if (k, p) ∈ sset r for k p
    proof −
      obtain ra ka pa where 1: r = fromN (Suc ka) ||| ra v = (ka, pa)
        using grun-elim[OF 2(2)] by this
      have 2: k ∈ sset (fromN (Suc ka)) using 1(1) that
        by (metis image-eqI prod.sel(1) szip-smap-fst stream.set-map)
      show ?thesis using 2 by simp
    qed
    have 5: smap f' (gtrace (stl r) (shd r)) = smap f (gtrace (stl r) (shd r))
    proof (rule stream.map-cong)
      show gtrace (stl r) (shd r) = gtrace (stl r) (shd r) by rule
    next
      fix z
      assume 1: z ∈ sset (gtrace (stl r) (shd r))
      have 2: fst z ≠ 0 using 4 1 by (metis gtrace-alt-def prod.collapse stl-sset)
      show f' z = f z using 2 unfolding f'-def by (auto simp: case-prod-beta)
    qed
    show odd k using 1 3 5 by simp
  qed

  define g where g i p ≡ if p ∈ reach A w i then Some (f' (i, p)) else None for
i p
  have g-dom[simp]: dom (g i) = reach A w i for i
    unfolding g-def by (auto) (metis option.simps(3))
  have g-0[simp]: g 0 = const (Some (2 * card (nodes A))) |` initial A
```

unfolding *g-def f'-def* **by** *auto*
have *g-Suc*[*simp*]: *g* (*Suc n*) ∈ *lr-succ A* (*w* !! *n*) (*g n*) **for** *n*
unfolding *lr-succ-def*
proof (*intro CollectI conjI ballI impI*)
show *dom* (*g* (*Suc n*)) = ⋃ (*transition A* (*w* !! *n*) ‘ *dom* (*g n*)) **using** *snth-in assms*(*2*) **by** *auto*
next
fix *p q*
assume *100*: *p* ∈ *dom* (*g n*) *q* ∈ *transition A* (*w* !! *n*) *p*
have *101*: *q* ∈ *reach A w* (*Suc n*) **using** *snth-in assms*(*2*) *100* **by** *auto*
have *102*: (*n*, *p*) ∈ *gunodes A w* **using** *100*(*1*) *reach-gunodes g-dom* **by** *blast*
have *103*: (*Suc n*, *q*) ∈ *gusuccessors A w* (*n*, *p*) **using** *snth-in assms*(*2*) *102 100*(*2*) **by** *auto*
have *104*: *p* ∈ *reach A w n* **using** *100*(*1*) **by** *simp*
have *g* (*Suc n*) *q* = *Some* (*f'* (*Suc n*, *q*)) **using** *101* **unfolding** *g-def* **by** *simp*
also have *the* ... = *f'* (*Suc n*, *q*) **by** *simp*
also have ... ≤ *f'* (*n*, *p*) **using** *0* **unfolding** *ranking-def* **using** *102 103* **by** *simp*
also have ... = *the* (*Some* (*f'* (*n*, *p*))) **by** *simp*
also have *Some* (*f'* (*n*, *p*)) = *g n p* **using** *104* **unfolding** *g-def* **by** *simp*
finally show *the* (*g* (*Suc n*) *q*) ≤ *the* (*g n p*) **by** *this*
next
fix *p*
assume *100*: *p* ∈ *dom* (*g* (*Suc n*)) *accepting A p*
have *101*: *p* ∈ *reach A w* (*Suc n*) **using** *100*(*1*) **by** *simp*
have *102*: (*Suc n*, *p*) ∈ *gunodes A w* **using** *101 reach-gunodes* **by** *blast*
have *103*: *gaccepting A* (*Suc n*, *p*) **using** *100*(*2*) **by** *simp*
have *the* (*g* (*Suc n*) *p*) = *f'* (*Suc n*, *p*) **using** *101* **unfolding** *g-def* **by** *simp*
also have *even* ... **using** *0* **unfolding** *ranking-def* **using** *102 103* **by** *auto*
finally show *even* (*the* (*g* (*Suc n*) *p*)) **by** *this*
qed

define *P* **where** *P* ≡ *rec-nat* {} (λ *n*. *st-succ A* (*w* !! *n*) (*g* (*Suc n*)))
have *P-0*[*simp*]: *P 0* = {} **unfolding** *P-def* **by** *simp*
have *P-Suc*[*simp*]: *P* (*Suc n*) = *st-succ A* (*w* !! *n*) (*g* (*Suc n*)) (*P n*) **for** *n*
unfolding *P-def* **by** *simp*
have *P-reach*: *P n* ⊆ *reach A w n* **for** *n*
using *snth-in assms*(*2*) **by** (*induct n*) (*auto simp add*: *st-succ-def*)
have *P n* ⊆ *reach A w n* **for** *n* **using** *P-reach* **by** *auto*
also have ... *n* ⊆ *nodes A* **for** *n* **using** *reach-nodes* **by** *this*
also have *finite* (*nodes A*) **using** *assms*(*1*) **by** *this*
finally have *P-finite*: *finite* (*P n*) **for** *n* **by** *this*

define *s* **where** *s* ≡ *smap g nats* ||| *smap P nats*

show *?thesis*
proof
show *run* (*complement A*) (*w* ||| *stl s*) (*shd s*)

proof (*intro nba.snth-run conjI*, *simp-all del*: *stake.simps stake-szip*)
fix *k*
show *w* !! *k* ∈ *alphabet* (*complement A*) **using** *snth-in assms*(*2*) **unfolding** *complement-def* **by** *auto*
have *stl s* !! *k* = *s* !! *Suc k* **by** *simp*
also have ... ∈ *complement-succ A* (*w* !! *k*) (*s* !! *k*)
unfolding *complement-succ-def s-def* **using** *P-Suc* **by** *simp*
also have ... = *complement-succ A* (*w* !! *k*) (*target* (*stake k* (*w* ||| *stl s*)) (*shd s*))
unfolding *sscan-scons-snth*[*symmetric*] *nba.trace-alt-def* **by** *simp*
also have ... = *transition* (*complement A*) (*w* !! *k*) (*target* (*stake k* (*w* ||| *stl s*)) (*shd s*))
unfolding *complement-def nba.sel* **by** *rule*
finally show *stl s* !! *k* ∈
transition (*complement A*) (*w* !! *k*) (*target* (*stake k* (*w* ||| *stl s*)) (*shd s*)) **by** *this*
qed
show *shd s* ∈ *initial* (*complement A*) **unfolding** *complement-def s-def* **using** *P-0* **by** *simp*
show *infs* (*accepting* (*complement A*)) (*shd s* ## *stl s*)
proof −
have *10*: ∀ *n*. ∃ *k* ≥ *n*. *P k* = {}
proof (*rule ccontr*)
assume *20*: ¬ (∀ *n*. ∃ *k* ≥ *n*. *P k* = {})
obtain *k* **where** *22*: *P* (*k* + *n*) ≠ {} **for** *n* **using** *20* **using** *le-add1* **by** *blast*
define *m* **where** *m n S* ≡ {*p* ∈ ⋃ (*transition A* (*w* !! *n*) ‘ *S*). *even* (*the* (*g* (*Suc n*) *p*))} **for** *n S*
define *R* **where** *R i n S* ≡ *rec-nat S* (λ *i*. *m* (*n* + *i*)) *i* **for** *i n S*
have *R-0*[*simp*]: *R 0 n* = *id* **for** *n* **unfolding** *R-def* **by** *auto*
have *R-Suc*[*simp*]: *R* (*Suc i*) *n* = *m* (*n* + *i*) ∘ *R i n* **for** *i n* **unfolding** *R-def* **by** *auto*
have *R-Suc'*: *R* (*Suc i*) *n* = *R i* (*Suc n*) ∘ *m n* **for** *i n* **unfolding** *R-Suc* **by** (*induct i*) (*auto*)
have *R-reach*: *R i n S* ⊆ *reach A w* (*n* + *i*) **if** *S* ⊆ *reach A w n* **for** *i n S*
using *snth-in assms*(*2*) *that m-def* **by** (*induct i*) (*auto*)
have *P-R*: *P* (*k* + *i*) = *R i k* (*P k*) **for** *i*
using *22* **by** (*induct i*) (*auto simp add*: *case-prod-beta' m-def st-succ-def*)

have *50*: *R i n S* = (⋃ *p* ∈ *S*. *R i n* {*p*}) **for** *i n S*
by (*induct i*) (*auto simp add*: *m-def prod.case-eq-if*)
have *51*: *R* (*i* + *j*) *n S* = {} **if** *R i n S* = {} **for** *i j n S*
using *that* **by** (*induct j*) (*auto simp add*: *m-def prod.case-eq-if*)
have *52*: *R j n S* = {} **if** *i* ≤ *j R i n S* = {} **for** *i j n S*
using *51* **by** (*metis le-add-diff-inverse that*(*1*) *that*(*2*))

have *1*: ∃ *p* ∈ *S*. ∀ *i*. *R i n* {*p*} ≠ {}
if *assms*: *finite S* ⋀ *i*. *R i n S* ≠ {} **for** *n S*
proof (*rule ccontr*)

assume *1*: $\neg\ (\exists\ p \in S.\ \forall\ i.\ R\ i\ n\ \{p\} \neq \{\})$
obtain *f* **where** *3*: $\bigwedge p.\ p \in S \Longrightarrow R\ (f\ p)\ n\ \{p\} = \{\}$ **using** *1* **by** *metis*
have *4*: $R\ (Sup\ (f\ `\ S))\ n\ \{p\} = \{\}$ **if** $p \in S$ **for** *p*
proof (*rule 52*)
show $f\ p \leq Sup\ (f\ `\ S)$ **using** *assms(1) that* **by** (*auto intro*: *le-cSup-finite*)
show $R\ (f\ p)\ n\ \{p\} = \{\}$ **using** *3 that* **by** *this*
qed
have $R\ (Sup\ (f\ `\ S))\ n\ S = (\bigcup\ p \in S.\ R\ (Sup\ (f\ `\ S))\ n\ \{p\})$ **using** *50* **by** *this*
also have $\ldots = \{\}$ **using** *4* **by** *simp*
finally have *5*: $R\ (Sup\ (f\ `\ S))\ n\ S = \{\}$ **by** *this*
show *False* **using** *that(2) 5* **by** *auto*
qed
have *2*: $\bigwedge i.\ R\ i\ (k + 0)\ (P\ k) \neq \{\}$ **using** *22 P-R* **by** *simp*
obtain *p* **where** *3*: $p \in P\ k \bigwedge i.\ R\ i\ k\ \{p\} \neq \{\}$ **using** *1*[*OF P-finite 2*] **by** *auto*

define *Q* **where** $Q\ n\ p \equiv (\forall\ i.\ R\ i\ (k + n)\ \{p\} \neq \{\}) \wedge p \in P\ (k + n)$ **for** *n p*
have *5*: $\exists\ q \in transition\ A\ (w\ !!\ (k + n))\ p.\ Q\ (Suc\ n)\ q$ **if** $Q\ n\ p$ **for** *n p*
proof −
have *11*: $p \in P\ (k + n) \bigwedge i.\ R\ i\ (k + n)\ \{p\} \neq \{\}$ **using** *that* **unfolding** *Q-def* **by** *auto*
have *12*: $R\ (Suc\ i)\ (k + n)\ \{p\} \neq \{\}$ **for** *i* **using** *11(2)* **by** *this*
have *13*: $R\ i\ (k + Suc\ n)\ (m\ (k + n)\ \{p\}) \neq \{\}$ **for** *i* **using** *12* **unfolding** *R-Suc′* **by** *simp*
have $\{p\} \subseteq P\ (k + n)$ **using** *11(1)* **by** *auto*
also have $\ldots \subseteq reach\ A\ w\ (k + n)$ **using** *P-reach* **by** *this*
finally have $R\ 1\ (k + n)\ \{p\} \subseteq reach\ A\ w\ (k + n + 1)$ **using** *R-reach* **by** *blast*
also have $\ldots \subseteq nodes\ A$ **using** *reach-nodes* **by** *this*
also have *finite* $(nodes\ A)$ **using** *assms(1)* **by** *this*
finally have *14*: *finite* $(m\ (k + n)\ \{p\})$ **by** *simp*
obtain *q* **where** *14*: $q \in m\ (k + n)\ \{p\} \bigwedge i.\ R\ i\ (k + Suc\ n)\ \{q\} \neq \{\}$
using *1*[*OF 14 13*] **by** *auto*
show *?thesis*
unfolding *Q-def prod.case*
proof (*intro bexI conjI allI*)
show $\bigwedge i.\ R\ i\ (k + Suc\ n)\ \{q\} \neq \{\}$ **using** *14(2)* **by** *this*
show $q \in P\ (k + Suc\ n)$
using *14(1) 11(1) 22* **unfolding** *m-def* **by** (*auto simp add*: *st-succ-def*)
show $q \in transition\ A\ (w\ !!\ (k + n))\ p$ **using** *14(1)* **unfolding** *m-def* **by** *simp*
qed
qed
obtain *r* **where** *23*:
$run\ A\ r\ p \bigwedge i.\ Q\ i\ ((p\ \#\#\ trace\ r\ p)\ !!\ i) \bigwedge i.\ fst\ (r\ !!\ i) = w\ !!\ (k + i)$
proof (*rule nba.invariant-run-index*[*of Q 0 p A* λ *n p a. fst a* $= w\ !!\ (k + n)$])

show *Q 0 p* **unfolding** *Q-def* **using** *3* **by** *auto*
show ∃ *a*. (*fst a* ∈ *alphabet A* ∧ *snd a* ∈ *transition A* (*fst a*) *p*) ∧
Q (*Suc n*) (*snd a*) ∧ *fst a* = *w* !! (*k* + *n*) **if** *Q n p* **for** *n p*
using *snth-in assms*(*2*) *5 that* **by** *fastforce*
qed *auto*
have *20*: *smap fst r* = *sdrop k w* **using** *23*(*3*) **by** (*intro eqI-snth*) (*simp add*: *case-prod-beta*)
have *21*: (*p ## smap snd r*) !! *i* ∈ *P* (*k* + *i*) **for** *i*
using *23*(*2*) **unfolding** *Q-def* **unfolding** *nba.trace-alt-def* **by** *simp*
obtain *r* **where** *23*: *run A* (*sdrop k w* ||| *stl r*) (*shd r*) ⋀ *i*. *r* !! *i* ∈ *P* (*k* + *i*)
using *20 21 23*(*1*) **by** (*metis stream.sel*(*1*) *stream.sel*(*2*) *szip-smap*)
let *?v* = (*k*, *shd r*)
let *?r* = *fromN* (*Suc k*) ||| *stl r*
have *shd r* = *r* !! *0* **by** *simp*
also have ... ∈ *P k* **using** *23*(*2*)[*of 0*] **by** *simp*
also have ... ⊆ *reach A w k* **using** *P-reach* **by** *this*
finally have *24*: *?v* ∈ *gunodes A w* **using** *reach-gunodes* **by** *blast*
have *25*: *gurun A w ?r ?v* **using** *run-grun 23*(*1*) **by** *this*
obtain *l* **where** *26*: *Ball* (*sset* (*smap f'* (*gtrace* (*sdrop l ?r*) (*gtarget* (*stake l ?r*) *?v*)))) *odd*
using *ranking-stuck-odd 0 24 25* **by** *this*
have *27*: *f'* (*Suc* (*k* + *l*), *r* !! *Suc l*) =
shd (*smap f'* (*gtrace* (*sdrop l ?r*) (*gtarget* (*stake l ?r*) *?v*))) **by** (*simp add*: *algebra-simps*)
also have ... ∈ *sset* (*smap f'* (*gtrace* (*sdrop l ?r*) (*gtarget* (*stake l ?r*) *?v*)))
using *shd-sset* **by** *this*
finally have *28*: *odd* (*f'* (*Suc* (*k* + *l*), *r* !! *Suc l*)) **using** *26* **by** *auto*
have *r* !! *Suc l* ∈ *P* (*Suc* (*k* + *l*)) **using** *23*(*2*) **by** (*metis add-Suc-right*)
also have ... = {*p* ∈ ⋃ (*transition A* (*w* !! (*k* + *l*)) ‘ *P* (*k* + *l*)).
even (*the* (*g* (*Suc* (*k* + *l*)) *p*))} **using** *23*(*2*) **by** (*auto simp*: *st-succ-def*)
also have ... ⊆ {*p*. *even* (*the* (*g* (*Suc* (*k* + *l*)) *p*))} **by** *auto*
finally have *29*: *even* (*the* (*g* (*Suc* (*k* + *l*)) (*r* !! *Suc l*))) **by** *auto*
have *30*: *r* !! *Suc l* ∈ *reach A w* (*Suc* (*k* + *l*))
using *23*(*2*) *P-reach* **by** (*metis add-Suc-right subsetCE*)
have *31*: *even* (*f'* (*Suc* (*k* + *l*), *r* !! *Suc l*)) **using** *29 30* **unfolding** *g-def* **by** *simp*
show *False* **using** *28 31* **by** *simp*
qed
have *11*: *infs* (λ *k*. *P k* = {}) *nats* **using** *10* **unfolding** *infs-snth* **by** *simp*
have *infs* (λ *S*. *S* = {}) (*smap snd* (*smap g nats* ||| *smap P nats*))
using *11* **by** (*simp add*: *comp-def*)
then have *infs* (λ *x*. *snd x* = {}) (*smap g nats* ||| *smap P nats*)
by (*simp add*: *comp-def del*: *szip-smap-snd*)
then have *infs* (λ (*f*, *P*). *P* = {}) (*smap g nats* ||| *smap P nats*)
by (*simp add*: *case-prod-beta'*)
then have *infs* (λ (*f*, *P*). *P* = {}) (*stl* (*smap g nats* ||| *smap P nats*)) **by** *blast*

then have *infs* (λ (*f*, *P*). *P* = {}) (*smap snd* (*w* ||| *stl* (*smap g nats* ||| *smap P nats*))) **by** *simp*
then have *infs* (λ (*f*, *P*). *P* = {}) (*stl s*) **unfolding** *s-def* **by** *simp*
then show *?thesis* **unfolding** *complement-def* **by** *auto*
qed
qed
qed

4.4 Correctness Theorem

theorem *complement-language*:
assumes *finite* (*nodes A*)
shows *language* (*complement A*) = *streams* (*alphabet A*) − *language A*
proof (*safe del*: *notI*)
have *1*: *alphabet* (*complement A*) = *alphabet A* **unfolding** *complement-def nba.sel* **by** *rule*
show *w* ∈ *streams* (*alphabet A*) **if** *w* ∈ *language* (*complement A*) **for** *w*
using *nba.language-alphabet that 1* **by** *force*
show *w* ∉ *language A* **if** *w* ∈ *language* (*complement A*) **for** *w*
using *complement-ranking ranking-language that* **by** *metis*
show *w* ∈ *language* (*complement A*) **if** *w* ∈ *streams* (*alphabet A*) *w* ∉ *language A* **for** *w*
using *language-ranking ranking-complement assms that* **by** *blast*
qed

end

5 Complementation Implementation

theory *Complementation-Implement*
imports
Transition-Systems-and-Automata.NBA-Implement
Complementation
begin

unbundle *lattice-syntax*

type-synonym *item* = *nat* × *bool*
type-synonym *'state items* = *'state* ⇀ *item*

type-synonym *state* = (*nat* × *item*) *list*
abbreviation *item-rel* ≡ *nat-rel* ×$_r$ *bool-rel*
abbreviation *state-rel* ≡ ⟨*nat-rel*, *item-rel*⟩ *list-map-rel*

abbreviation *pred A a q* ≡ {*p*. *q* ∈ *transition A a p*}

5.1 Phase 1

definition *cs-lr* :: *'state items* ⇒ *'state lr* **where**

cs-lr f ≡ *map-option fst* ∘ *f*
definition *cs-st* :: *'state items* ⇒ *'state st* **where**
cs-st f ≡ *f* −‘ *Some* ‘ *snd* −‘ {*True*}
abbreviation *cs-abs* :: *'state items* ⇒ *'state cs* **where**
cs-abs f ≡ (*cs-lr f*, *cs-st f*)
definition *cs-rep* :: *'state cs* ⇒ *'state items* **where**
cs-rep ≡ λ (*g*, *P*) *p*. *map-option* (λ *k*. (*k*, *p* ∈ *P*)) (*g p*)

lemma *cs-abs-rep*[*simp*]: *cs-rep* (*cs-abs f*) = *f*
proof
show *cs-rep* (*cs-abs f*) *x* = *f x* **for** *x*
unfolding *cs-lr-def cs-st-def cs-rep-def* **by** (*cases f x*) (*force*+)
qed
lemma *cs-rep-lr*[*simp*]: *cs-lr* (*cs-rep* (*g*, *P*)) = *g*
proof
show *cs-lr* (*cs-rep* (*g*, *P*)) *x* = *g x* **for** *x*
unfolding *cs-rep-def cs-lr-def* **by** (*cases g x*) (*auto*)
qed
lemma *cs-rep-st*[*simp*]: *cs-st* (*cs-rep* (*g*, *P*)) = *P* ∩ *dom g*
unfolding *cs-rep-def cs-st-def* **by** *force*

lemma *cs-lr-dom*[*simp*]: *dom* (*cs-lr f*) = *dom f* **unfolding** *cs-lr-def* **by** *simp*
lemma *cs-lr-apply*[*simp*]:
assumes *p* ∈ *dom f*
shows *the* (*cs-lr f p*) = *fst* (*the* (*f p*))
using *assms* **unfolding** *cs-lr-def* **by** *auto*

lemma *cs-rep-dom*[*simp*]: *dom* (*cs-rep* (*g*, *P*)) = *dom g* **unfolding** *cs-rep-def* **by** *auto*
lemma *cs-rep-apply*[*simp*]:
assumes *p* ∈ *dom f*
shows *fst* (*the* (*cs-rep* (*f*, *P*) *p*)) = *the* (*f p*)
using *assms* **unfolding** *cs-rep-def* **by** *auto*

abbreviation *cs-rel* :: (*'state items* × *'state cs*) *set* **where**
cs-rel ≡ *br cs-abs top*

lemma *cs-rel-inv-single-valued*: *single-valued* (*cs-rel*$^{-1}$)
by (*auto intro*!: *inj-onI*) (*metis cs-abs-rep*)

definition *refresh-1* :: *'state items* ⇒ *'state items* **where**
refresh-1 f ≡ *if True* ∈ *snd* ‘ *ran f then f else map-option* (*apsnd top*) ∘ *f*
definition *ranks-1* ::
(*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state items* ⇒ *'state items set* **where**
ranks-1 A a f ≡ {*g*.
dom g = ⋃((*transition A a*) ‘ (*dom f*)) ∧
(∀ *p* ∈ *dom f*. ∀ *q* ∈ *transition A a p*. *fst* (*the* (*g q*)) ≤ *fst* (*the* (*f p*))) ∧
(∀ *q* ∈ *dom g*. *accepting A q* ⟶ *even* (*fst* (*the* (*g q*)))) ∧
cs-st g = {*q* ∈ ⋃((*transition A a*) ‘ (*cs-st f*)). *even* (*fst* (*the* (*g q*)))}}

definition *complement-succ-1* ::
(*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state items* ⇒ *'state items set* **where**
complement-succ-1 A a = *ranks-1 A a* ∘ *refresh-1*
definition *complement-1* :: (*'label*, *'state*) *nba* ⇒ (*'label*, *'state items*) *nba* **where**
complement-1 A ≡ *nba*
(*alphabet A*)
({*const* (*Some* (2 ∗ *card* (*nodes A*), *False*)) |‘ *initial A*})
(*complement-succ-1 A*)
(λ *f*. *cs-st f* = {})

lemma *refresh-1-dom*[*simp*]: *dom* (*refresh-1 f*) = *dom f* **unfolding** *refresh-1-def* **by** *simp*
lemma *refresh-1-apply*[*simp*]: *fst* (*the* (*refresh-1 f p*)) = *fst* (*the* (*f p*))
unfolding *refresh-1-def* **by** (*cases f p*) (*auto*)
lemma *refresh-1-cs-st*[*simp*]: *cs-st* (*refresh-1 f*) = (*if cs-st f* = {} *then dom f else cs-st f*)
unfolding *refresh-1-def cs-st-def ran-def image-def vimage-def* **by** *auto*

lemma *complement-succ-1-abs*:
assumes *g* ∈ *complement-succ-1 A a f*
shows *cs-abs g* ∈ *complement-succ A a* (*cs-abs f*)
unfolding *complement-succ-def*
proof (*simp*, *rule*)
have *1*:
dom g = ⋃((*transition A a*) ‘ (*dom f*))
∀ *p* ∈ *dom f*. ∀ *q* ∈ *transition A a p*. *fst* (*the* (*g q*)) ≤ *fst* (*the* (*f p*))
∀ *p* ∈ *dom g*. *accepting A p* ⟶ *even* (*fst* (*the* (*g p*)))
using *assms* **unfolding** *complement-succ-1-def ranks-1-def* **by** *simp-all*
show *cs-lr g* ∈ *lr-succ A a* (*cs-lr f*)
unfolding *lr-succ-def*
proof (*intro CollectI conjI ballI impI*)
show *dom* (*cs-lr g*) = ⋃ (*transition A a* ‘ *dom* (*cs-lr f*)) **using** *1* **by** *simp*
next
fix *p q*
assume *2*: *p* ∈ *dom* (*cs-lr f*) *q* ∈ *transition A a p*
have *3*: *q* ∈ *dom* (*cs-lr g*) **using** *1 2* **by** *auto*
show *the* (*cs-lr g q*) ≤ *the* (*cs-lr f p*) **using** *1 2 3* **by** *simp*
next
fix *p*
assume *2*: *p* ∈ *dom* (*cs-lr g*) *accepting A p*
show *even* (*the* (*cs-lr g p*)) **using** *1 2* **by** *auto*
qed
have *2*: *cs-st g* = {*q* ∈ ⋃ (*transition A a* ‘ *cs-st* (*refresh-1 f*)). *even* (*fst* (*the* (*g q*)))}
using *assms* **unfolding** *complement-succ-1-def ranks-1-def* **by** *simp*
show *cs-st g* = *st-succ A a* (*cs-lr g*) (*cs-st f*)
proof (*cases cs-st f* = {})
case *True*
have *3*: *the* (*cs-lr g q*) = *fst* (*the* (*g q*)) **if** *q* ∈ ⋃((*transition A a*) ‘ (*dom f*))

for q
using *that 1(1)* **by** *simp*
show *?thesis* **using** *2 3* **unfolding** *st-succ-def refresh-1-cs-st True cs-lr-dom 1(1)* **by** *force*
next
case *False*
have *3*: *the* (*cs-lr g q*) = *fst* (*the* (*g q*)) **if** $q \in \bigcup$((*transition A a*) ‘ (*cs-st f*)) **for** q
using *that 1(1)* **by**
(*auto intro*!: *cs-lr-apply*)
(*metis IntE UN-iff cs-abs-rep cs-lr-dom cs-rep-st domD prod.collapse*)
have *cs-st g* = {$q \in \bigcup$ (*transition A a* ‘ *cs-st* (*refresh-1 f*)). *even* (*fst* (*the* (*g q*)))}
using *2* **by** *this*
also have *cs-st* (*refresh-1 f*) = *cs-st f* **using** *False* **by** *simp*
also have {$q \in \bigcup$((*transition A a*) ‘ (*cs-st f*)). *even* (*fst* (*the* (*g q*)))} =
{$q \in \bigcup$((*transition A a*) ‘ (*cs-st f*)). *even* (*the* (*cs-lr g q*))} **using** *3* **by** *metis*
also have ... = *st-succ A a* (*cs-lr g*) (*cs-st f*) **unfolding** *st-succ-def* **using** *False* **by** *simp*
finally show *?thesis* **by** *this*
qed
qed
lemma *complement-succ-1-rep*:
assumes $P \subseteq$ *dom f* (*g*, *Q*) ∈ *complement-succ A a* (*f*, *P*)
shows *cs-rep* (*g*, *Q*) ∈ *complement-succ-1 A a* (*cs-rep* (*f*, *P*))
unfolding *complement-succ-1-def ranks-1-def comp-apply*
proof (*intro CollectI conjI ballI impI*)
have *1*:
dom g = $\bigcup$((*transition A a*) ‘ (*dom f*))
$\forall$ *p* ∈ *dom f*. $\forall$ *q* ∈ *transition A a p*. *the* (*g q*) ≤ *the* (*f p*)
$\forall$ *p* ∈ *dom g*. *accepting A p* ⟶ *even* (*the* (*g p*))
using *assms(2)* **unfolding** *complement-succ-def lr-succ-def* **by** *simp-all*
have *2*: *Q* = {*q* ∈ *if P* = {} *then dom g else* $\bigcup$((*transition A a*) ‘ *P*). *even* (*the* (*g q*))}
using *assms(2)* **unfolding** *complement-succ-def st-succ-def* **by** *simp*
have *3*: *Q* ⊆ *dom g* **unfolding** *2 1(1)* **using** *assms(1)* **by** *auto*
show *dom* (*cs-rep* (*g*, *Q*)) = $\bigcup$ (*transition A a* ‘ *dom* (*refresh-1* (*cs-rep* (*f*, *P*)))) **using** *1* **by** *simp*
show $\bigwedge$ *p q*. *p* ∈ *dom* (*refresh-1* (*cs-rep* (*f*, *P*))) ⟹ *q* ∈ *transition A a p* ⟹
fst (*the* (*cs-rep* (*g*, *Q*) *q*)) ≤ *fst* (*the* (*refresh-1* (*cs-rep* (*f*, *P*)) *p*))
using *1(1*, *2)* **by** (*auto*) (*metis UN-I cs-rep-apply domI option.sel*)
show $\bigwedge$ *p*. *p* ∈ *dom* (*cs-rep* (*g*, *Q*)) ⟹ *accepting A p* ⟹ *even* (*fst* (*the* (*cs-rep* (*g*, *Q*) *p*)))
using *1(1*, *3)* **by** *auto*
show *cs-st* (*cs-rep* (*g*, *Q*)) = {*q* ∈ $\bigcup$ (*transition A a* ‘ *cs-st* (*refresh-1* (*cs-rep* (*f*, *P*)))).
even (*fst* (*the* (*cs-rep* (*g*, *Q*) *q*)))}
proof (*cases P* = {})

```
      case True
      have cs-st (cs-rep (g, Q)) = Q using 3 by auto
      also have ... = {q ∈ dom g. even (the (g q))} unfolding 2 using True by
auto
       also have ... = {q ∈ dom g. even (fst (the (cs-rep (g, Q) q)))} using
cs-rep-apply by metis
      also have dom g = ⋃((transition A a) ‘ (dom f)) using 1(1) by this
      also have dom f = cs-st (refresh-1 (cs-rep (f, P))) using True by simp
      finally show ?thesis by this
    next
      case False
     have 4: fst (the (cs-rep (g, Q) q)) = the (g q) if q ∈ ⋃((transition A a) ‘ P)
for q
       using 1(1) that assms(1) by (fast intro: cs-rep-apply)
      have cs-st (cs-rep (g, Q)) = Q using 3 by auto
      also have ... = {q ∈ ⋃((transition A a) ‘ P). even (the (g q))} unfolding
2 using False by auto
      also have ... = {q ∈ ⋃((transition A a) ‘ P). even (fst (the (cs-rep (g, Q)
q)))} using 4 by force
      also have P = (cs-st (refresh-1 (cs-rep (f, P)))) using assms(1) False by
auto
      finally show ?thesis by simp
    qed
  qed

  lemma complement-succ-1-refine: (complement-succ-1, complement-succ) ∈
    Id → Id → cs-rel → ⟨cs-rel⟩ set-rel
  proof (clarsimp simp: br-set-rel-alt in-br-conv)
    fix A :: ('a, 'b) nba
    fix a f
    show complement-succ A a (cs-abs f) = cs-abs ‘ complement-succ-1 A a f
    proof safe
      fix g Q
      assume 1: (g, Q) ∈ complement-succ A a (cs-abs f)
      have 2: Q ⊆ dom g
        using 1 unfolding complement-succ-def lr-succ-def st-succ-def
        by (auto) (metis IntE cs-abs-rep cs-lr-dom cs-rep-st)
      have 3: cs-st f ⊆ dom (cs-lr f) unfolding cs-st-def by auto
      show (g, Q) ∈ cs-abs ‘ complement-succ-1 A a f
      proof
        show (g, Q) = cs-abs (cs-rep (g, Q)) using 2 by auto
        have cs-rep (g, Q) ∈ complement-succ-1 A a (cs-rep (cs-abs f))
          using complement-succ-1-rep 3 1 by this
        also have cs-rep (cs-abs f) = f by simp
        finally show cs-rep (g, Q) ∈ complement-succ-1 A a f by this
      qed
    next
      fix g
      assume 1: g ∈ complement-succ-1 A a f
```

show *cs-abs g* ∈ *complement-succ A a* (*cs-abs f*) **using** *complement-succ-1-abs 1* **by** *this*
qed
qed
lemma *complement-1-refine*: (*complement-1*, *complement*) ∈ ⟨*Id*, *Id*⟩ *nba-rel* → ⟨*Id*, *cs-rel*⟩ *nba-rel*
unfolding *complement-1-def complement-def*
proof *parametricity*
fix *A B* :: ('*a*, '*b*) *nba*
assume *1*: (*A*, *B*) ∈ ⟨*Id*, *Id*⟩ *nba-rel*
have *2*: (*const* (*Some* (*2* ∗ *card* (*nodes B*), *False*)) |‘ *initial B*,
const (*Some* (*2* ∗ *card* (*nodes B*))) |‘ *initial B*, {}) ∈ *cs-rel*
unfolding *cs-lr-def cs-st-def in-br-conv* **by** (*force simp*: *restrict-map-def*)
show (*complement-succ-1 A*, *complement-succ B*) ∈ *Id* → *cs-rel* → ⟨*cs-rel*⟩ *set-rel*
using *complement-succ-1-refine 1* **by** *parametricity auto*
show ({*const* (*Some* (*2* ∗ *card* (*nodes A*), *False*)) |‘ *initial A*},
{*const* (*Some* (*2* ∗ *card* (*nodes B*))) |‘ *initial B*} × {{}}) ∈ ⟨*cs-rel*⟩ *set-rel*
using *1 2* **by** *simp parametricity*
show (λ *f*. *cs-st f* = {}, λ (*f*, *P*). *P* = {}) ∈ *cs-rel* → *bool-rel* **by** (*auto simp*: *in-br-conv*)
qed

5.2 Phase 2

definition *ranks-2* :: ('*label*, '*state*) *nba* ⇒ '*label* ⇒ '*state items* ⇒ '*state items set* **where**
ranks-2 A a f ≡ {*g*.
dom g = ⋃((*transition A a*) ‘ (*dom f*)) ∧
(∀ *q l d*. *g q* = *Some* (*l*, *d*) ⟶
l ≤ ⨅ (*fst* ‘ *Some* −‘ *f* ‘ *pred A a q*) ∧
(*d* ⟷ ⨆ (*snd* ‘ *Some* −‘ *f* ‘ *pred A a q*) ∧ *even l*) ∧
(*accepting A q* ⟶ *even l*))}
definition *complement-succ-2* ::
('*label*, '*state*) *nba* ⇒ '*label* ⇒ '*state items* ⇒ '*state items set* **where**
complement-succ-2 A a ≡ *ranks-2 A a* ∘ *refresh-1*
definition *complement-2* :: ('*label*, '*state*) *nba* ⇒ ('*label*, '*state items*) *nba* **where**
complement-2 A ≡ *nba*
(*alphabet A*)
({*const* (*Some* (*2* ∗ *card* (*nodes A*), *False*)) |‘ *initial A*})
(*complement-succ-2 A*)
(λ *f*. *True* ∉ *snd* ‘ *ran f*)

lemma *ranks-2-refine*: *ranks-2* = *ranks-1*
proof (*intro ext*)
fix *A* :: ('*a*, '*b*) *nba* **and** *a f*
show *ranks-2 A a f* = *ranks-1 A a f*
proof *safe*
fix *g*

```
    assume 1: g ∈ ranks-2 A a f
   have 2: dom g = ⋃((transition A a) ‘ (dom f)) using 1 unfolding ranks-2-def
by auto
    have 3: g q = Some (l, d) ⟹ l ≤ ⨅ (fst ‘ Some −‘ f ‘ pred A a q) for q l d
     using 1 unfolding ranks-2-def by auto
    have 4: g q = Some (l, d) ⟹ d ⟷ ⨆ (snd ‘ Some −‘ f ‘ pred A a q) ∧
even l for q l d
     using 1 unfolding ranks-2-def by auto
    have 5: g q = Some (l, d) ⟹ accepting A q ⟹ even l for q l d
     using 1 unfolding ranks-2-def by auto
    show g ∈ ranks-1 A a f
    unfolding ranks-1-def
    proof (intro CollectI conjI ballI impI)
     show dom g = ⋃((transition A a) ‘ (dom f)) using 2 by this
    next
     fix p q
     assume 10: p ∈ dom f q ∈ transition A a p
     obtain k c where 11: f p = Some (k, c) using 10(1) by auto
     have 12: q ∈ dom g using 10 2 by auto
     obtain l d where 13: g q = Some (l, d) using 12 by auto
     have fst (the (g q)) = l unfolding 13 by simp
     also have ... ≤ ⨅ (fst ‘ Some −‘ f ‘ pred A a q) using 3 13 by this
     also have ... ≤ k
     proof (rule cInf-lower)
      show k ∈ fst ‘ Some −‘ f ‘ pred A a q using 11 10(2) by force
      show bdd-below (fst ‘ Some −‘ f ‘ pred A a q) by simp
     qed
     also have ... = fst (the (f p)) unfolding 11 by simp
     finally show fst (the (g q)) ≤ fst (the (f p)) by this
    next
     fix q
     assume 10: q ∈ dom g accepting A q
     show even (fst (the (g q))) using 10 5 by auto
    next
     show cs-st g = {q ∈ ⋃((transition A a) ‘ (cs-st f)). even (fst (the (g q)))}
     proof
      show cs-st g ⊆ {q ∈ ⋃((transition A a) ‘ (cs-st f)). even (fst (the (g q)))}
       using 4 unfolding cs-st-def image-def vimage-def by auto metis+
      show {q ∈ ⋃((transition A a) ‘ (cs-st f)). even (fst (the (g q)))} ⊆ cs-st g
       proof safe
        fix p q
        assume 10: even (fst (the (g q))) p ∈ cs-st f q ∈ transition A a p
        have 12: q ∈ dom g using 10 2 unfolding cs-st-def by auto
        show q ∈ cs-st g using 10 4 12 unfolding cs-st-def image-def by force
       qed
     qed
    qed
   next
    fix g
```

assume *1*: $g \in$ *ranks-1 A a f*
have *2*: *dom g* = $\bigcup$((*transition A a*) ‘ (*dom f*)) **using** *1* **unfolding** *ranks-1-def* **by** *auto*
have *3*: $\bigwedge$ *p q*. *p* ∈ *dom f* ⟹ *q* ∈ *transition A a p* ⟹ *fst* (*the* (*g q*)) ≤ *fst* (*the* (*f p*))
using *1* **unfolding** *ranks-1-def* **by** *auto*
have *4*: $\bigwedge$ *q*. *q* ∈ *dom g* ⟹ *accepting A q* ⟹ *even* (*fst* (*the* (*g q*)))
using *1* **unfolding** *ranks-1-def* **by** *auto*
have *5*: *cs-st g* = {*q* ∈ $\bigcup$((*transition A a*) ‘ (*cs-st f*)). *even* (*fst* (*the* (*g q*)))}
using *1* **unfolding** *ranks-1-def* **by** *auto*
show *g* ∈ *ranks-2 A a f*
unfolding *ranks-2-def*
proof (*intro CollectI conjI allI impI*)
show *dom g* = $\bigcup$((*transition A a*) ‘ (*dom f*)) **using** *2* **by** *this*
next
fix *q l d*
assume *10*: *g q* = *Some* (*l*, *d*)
have *11*: *q* ∈ *dom g* **using** *10* **by** *auto*
show *l* ≤ $\bigsqcap$ (*fst* ‘ *Some* −‘ *f* ‘ *pred A a q*)
proof (*rule cInf-greatest*)
show *fst* ‘ *Some* −‘ *f* ‘ *pred A a q* ≠ {} **using** *11* **unfolding** *2 image-def vimage-def* **by** *force*
show $\bigwedge$ *x*. *x* ∈ *fst* ‘ *Some* −‘ *f* ‘ *pred A a q* ⟹ *l* ≤ *x*
using *3 10* **by** (*auto*) (*metis domI fst-conv option.sel*)
qed
have *d* ⟷ *q* ∈ *cs-st g* **unfolding** *cs-st-def* **by** (*force simp*: *10*)
also have *cs-st g* = {*q* ∈ $\bigcup$((*transition A a*) ‘ (*cs-st f*)). *even* (*fst* (*the* (*g q*)))} **using** *5* **by** *this*
also have *q* ∈ ... ⟷ (∃ *x* ∈ *cs-st f*. *q* ∈ *transition A a x*) ∧ *even l*
unfolding *mem-Collect-eq 10* **by** *simp*
also have ... ⟷ $\bigsqcup$ (*snd* ‘ *Some* −‘ *f* ‘ *pred A a q*) ∧ *even l*
unfolding *cs-st-def image-def vimage-def* **by** *auto metis+*
finally show *d* ⟷ $\bigsqcup$ (*snd* ‘ *Some* −‘ *f* ‘ *pred A a q*) ∧ *even l* **by** *this*
show *accepting A q* ⟹ *even l* **using** *4 10 11* **by** *force*
qed
qed
qed

lemma *complement-2-refine*: (*complement-2*, *complement-1*) ∈ ⟨*Id*, *Id*⟩ *nba-rel* → ⟨*Id*, *Id*⟩ *nba-rel*
unfolding *complement-2-def complement-1-def complement-succ-2-def complement-succ-1-def*
unfolding *ranks-2-refine cs-st-def image-def vimage-def ran-def* **by** *auto*

5.3 Phase 3

definition *bounds-3* :: (*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state items* ⇒ *'state items* **where**
bounds-3 A a f ≡ λ *q*. *let S* = *Some* −‘ *f* ‘ *pred A a q in*

if $S = \{\}$ *then None else Some* $(\bigsqcap (fst\ `\ S), \bigsqcup (snd\ `\ S))$
definition *items-3* :: $('label, 'state)\ nba \Rightarrow 'state \Rightarrow item \Rightarrow item\ set$ **where**
items-3 $A\ p \equiv \lambda\ (k, c).\ \{(l, c \wedge even\ l)\ |l.\ l \leq k \wedge (accepting\ A\ p \longrightarrow even\ l)\}$
definition *get-3* :: $('label, 'state)\ nba \Rightarrow 'state\ items \Rightarrow ('state \rightharpoonup item\ set)$ **where**
get-3 $A\ f \equiv \lambda\ p.\ map\text{-}option\ (items\text{-}3\ A\ p)\ (f\ p)$
definition *complement-succ-3* ::
$('label, 'state)\ nba \Rightarrow 'label \Rightarrow 'state\ items \Rightarrow 'state\ items\ set$ **where**
complement-succ-3 $A\ a \equiv expand\text{-}map \circ get\text{-}3\ A \circ bounds\text{-}3\ A\ a \circ refresh\text{-}1$
definition *complement-3* :: $('label, 'state)\ nba \Rightarrow ('label, 'state\ items)\ nba$ **where**
complement-3 $A \equiv nba$
$(alphabet\ A)$
$(\{(Some \circ (const\ (2 * card\ (nodes\ A), False))) |`\ initial\ A\})$
$(complement\text{-}succ\text{-}3\ A)$
$(\lambda\ f.\ \forall\ (p, k, c) \in map\text{-}to\text{-}set\ f.\ \neg\ c)$

lemma *bounds-3-dom*[*simp*]: $dom\ (bounds\text{-}3\ A\ a\ f) = \bigcup ((transition\ A\ a)\ `\ (dom\ f))$
unfolding *bounds-3-def Let-def dom-def* **by** (*force split*: *if-splits*)

lemma *items-3-nonempty*[*intro*!, *simp*]: *items-3 A p s* $\neq \{\}$ **unfolding** *items-3-def* **by** *auto*
lemma *items-3-finite*[*intro*!, *simp*]: *finite* $(items\text{-}3\ A\ p\ s)$
unfolding *items-3-def* **by** (*auto split*: *prod.splits*)

lemma *get-3-dom*[*simp*]: $dom\ (get\text{-}3\ A\ f) = dom\ f$ **unfolding** *get-3-def* **by** (*auto split*: *bind-splits*)
lemma *get-3-finite*[*intro*, *simp*]: $S \in ran\ (get\text{-}3\ A\ f) \Longrightarrow finite\ S$
unfolding *get-3-def ran-def* **by** *auto*
lemma *get-3-update*[*simp*]: $get\text{-}3\ A\ (f\ (p \mapsto s)) = (get\text{-}3\ A\ f)\ (p \mapsto items\text{-}3\ A\ p\ s)$
unfolding *get-3-def* **by** *auto*

lemma *expand-map-get-bounds-3*: $expand\text{-}map \circ get\text{-}3\ A \circ bounds\text{-}3\ A\ a = ranks\text{-}2\ A\ a$
proof (*intro ext set-eqI*, *unfold comp-apply*)
fix *f g*
have *1*: $(\forall\ x\ S\ y.\ get\text{-}3\ A\ (bounds\text{-}3\ A\ a\ f)\ x = Some\ S \longrightarrow g\ x = Some\ y \longrightarrow y \in S) \longleftrightarrow$
$(\forall\ q\ S\ l\ d.\ get\text{-}3\ A\ (bounds\text{-}3\ A\ a\ f)\ q = Some\ S \longrightarrow g\ q = Some\ (l, d) \longrightarrow (l, d) \in S)$
by *auto*
have *2*: $(\forall\ S.\ get\text{-}3\ A\ (bounds\text{-}3\ A\ a\ f)\ q = Some\ S \longrightarrow g\ q = Some\ (l, d) \longrightarrow (l, d) \in S) \longleftrightarrow$
$(g\ q = Some\ (l, d) \longrightarrow l \leq \bigsqcap (fst\ `\ (Some -`\ f\ `\ pred\ A\ a\ q)) \wedge$
$(d \longleftrightarrow \bigsqcup (snd\ `\ (Some -`\ f\ `\ pred\ A\ a\ q)) \wedge even\ l) \wedge (accepting\ A\ q \longrightarrow even\ l))$
if *3*: $dom\ g = \bigcup ((transition\ A\ a)\ `\ (dom\ f))$ **for** *q l d*
proof −

have *4*: $q \notin dom\ g$ **if** $Some -` f `` pred\ A\ a\ q = \{\}$ **unfolding** *3* **using** *that*
by *force*
show *?thesis* **unfolding** *get-3-def items-3-def bounds-3-def Let-def* **using** *4*
by *auto*
qed
show $g \in expand\text{-}map\ (get\text{-}3\ A\ (bounds\text{-}3\ A\ a\ f)) \longleftrightarrow g \in ranks\text{-}2\ A\ a\ f$
unfolding *expand-map-alt-def ranks-2-def mem-Collect-eq*
unfolding *get-3-dom bounds-3-dom 1* **using** *2* **by** *blast*
qed

lemma *complement-succ-3-refine*: *complement-succ-3* = *complement-succ-2*
unfolding *complement-succ-3-def complement-succ-2-def expand-map-get-bounds-3*
by *rule*
lemma *complement-initial-3-refine*: $\{const\ (Some\ (2 * card\ (nodes\ A), False)) |` initial\ A\} =$
$\{(Some \circ (const\ (2 * card\ (nodes\ A), False))) |` initial\ A\}$
unfolding *comp-apply* **by** *rule*
lemma *complement-accepting-3-refine*: $True \notin snd\ `` ran\ f \longleftrightarrow (\forall\ (p, k, c) \in map\text{-}to\text{-}set\ f.\ \neg\ c)$
unfolding *map-to-set-def ran-def* **by** *auto*

lemma *complement-3-refine*: $(complement\text{-}3, complement\text{-}2) \in \langle Id, Id\rangle\ nba\text{-}rel \to \langle Id, Id\rangle\ nba\text{-}rel$
unfolding *complement-3-def complement-2-def*
unfolding *complement-succ-3-refine complement-initial-3-refine complement-accepting-3-refine*
by *auto*

5.4 Phase 4

definition *items-4* :: $('label, 'state)\ nba \Rightarrow 'state \Rightarrow item \Rightarrow item\ set$ **where**
$items\text{-}4\ A\ p \equiv \lambda\ (k, c).\ \{(l, c \wedge even\ l) |l.\ k \leq Suc\ l \wedge l \leq k \wedge (accepting\ A\ p \longrightarrow even\ l)\}$
definition *get-4* :: $('label, 'state)\ nba \Rightarrow 'state\ items \Rightarrow ('state \rightharpoonup item\ set)$
where
$get\text{-}4\ A\ f \equiv \lambda\ p.\ map\text{-}option\ (items\text{-}4\ A\ p)\ (f\ p)$
definition *complement-succ-4* ::
$('label, 'state)\ nba \Rightarrow 'label \Rightarrow 'state\ items \Rightarrow 'state\ items\ set$ **where**
$complement\text{-}succ\text{-}4\ A\ a \equiv expand\text{-}map \circ get\text{-}4\ A \circ bounds\text{-}3\ A\ a \circ refresh\text{-}1$
definition *complement-4* :: $('label, 'state)\ nba \Rightarrow ('label, 'state\ items)\ nba$ **where**
$complement\text{-}4\ A \equiv nba$
$(alphabet\ A)$
$(\{(Some \circ (const\ (2 * card\ (nodes\ A), False))) |` initial\ A\})$
$(complement\text{-}succ\text{-}4\ A)$
$(\lambda\ f.\ \forall\ (p, k, c) \in map\text{-}to\text{-}set\ f.\ \neg\ c)$

lemma *get-4-dom*[*simp*]: $dom\ (get\text{-}4\ A\ f) = dom\ f$ **unfolding** *get-4-def* **by** (*auto split*: *bind-splits*)

definition R :: $'state\ items\ rel$ **where**

$R \equiv \{(f,\ g).$
$dom\ f = dom\ g \wedge$
$(\forall\ p \in dom\ f.\ fst\ (the\ (f\ p)) \leq fst\ (the\ (g\ p))) \wedge$
$(\forall\ p \in dom\ f.\ snd\ (the\ (f\ p)) \longleftrightarrow snd\ (the\ (g\ p)))\}$

lemma *bounds-R*:
assumes $(f,\ g) \in R$
assumes *bounds-3 A a* (*refresh-1 f*) $p = Some\ (n,\ e)$
assumes *bounds-3 A a* (*refresh-1 g*) $p = Some\ (k,\ c)$
shows $n \leq k\ e \longleftrightarrow c$
proof −
have *1*:
$dom\ f = dom\ g$
$\forall\ p \in dom\ f.\ fst\ (the\ (f\ p)) \leq fst\ (the\ (g\ p))$
$\forall\ p \in dom\ f.\ snd\ (the\ (f\ p)) \longleftrightarrow snd\ (the\ (g\ p))$
using *assms*(*1*) **unfolding** *R-def* **by** *auto*
have $n = \bigsqcap (fst\ `\ (Some\ -`\ refresh\text{-}1\ f\ `\ pred\ A\ a\ p))$
using *assms*(*2*) **unfolding** *bounds-3-def* **by** (*auto simp*: *Let-def split*: *if-splits*)
also have *fst ` Some −` refresh-1 f ` pred A a p* = *fst ` Some −` f ` pred A a p*
proof
show *fst ` Some −` refresh-1 f ` pred A a p* ⊆ *fst ` Some −` f ` pred A a p*
unfolding *refresh-1-def image-def*
by (*auto simp*: *map-option-case split*: *option.split*) (*force*)
show *fst ` Some −` f ` pred A a p* ⊆ *fst ` Some −` refresh-1 f ` pred A a p*
unfolding *refresh-1-def image-def*
by (*auto simp*: *map-option-case split*: *option.split*) (*metis fst-conv option.sel*)
qed
also have ... = *fst ` Some −` f ` (pred A a p ∩ dom f)*
unfolding *dom-def image-def Int-def* **by** *auto metis*
also have ... = *fst ` the ` f ` (pred A a p ∩ dom f)*
unfolding *dom-def* **by** *force*
also have ... = (*fst* ∘ *the* ∘ *f*) ` (*pred A a p* ∩ *dom f*) **by** *force*
also have $\bigsqcap((fst \circ the \circ f)\ `\ (pred\ A\ a\ p \cap dom\ f)) \leq$
$\bigsqcap((fst \circ the \circ g)\ `\ (pred\ A\ a\ p \cap dom\ g))$
proof (*rule cINF-mono*)
show *pred A a p* ∩ *dom g* ≠ {}
using *assms*(*2*) *1*(*1*) **unfolding** *bounds-3-def refresh-1-def*
by (*auto simp*: *Let-def split*: *if-splits*) (*force*+)
show *bdd-below* ((*fst* ∘ *the* ∘ *f*) ` (*pred A a p* ∩ *dom f*)) **by** *rule*
show ∃ *n* ∈ *pred A a p* ∩ *dom f*. (*fst* ∘ *the* ∘ *f*) *n* ≤ (*fst* ∘ *the* ∘ *g*) *m*
if *m* ∈ *pred A a p* ∩ *dom g* **for** *m* **using** *1 that* **by** *auto*
qed
also have (*fst* ∘ *the* ∘ *g*) ` (*pred A a p* ∩ *dom g*) = *fst ` the ` g ` (pred A a p* ∩ *dom g*) **by** *force*
also have ... = *fst ` Some −` g ` (pred A a p* ∩ *dom g*)
unfolding *dom-def* **by** *force*
also have ... = *fst ` Some −` g ` pred A a p*
unfolding *dom-def image-def Int-def* **by** *auto metis*
also have ... = *fst ` Some −` refresh-1 g ` pred A a p*

proof
show *fst ‘ Some −‘ g ‘ pred A a p ⊆ fst ‘ Some −‘ refresh-1 g ‘ pred A a p*
unfolding *refresh-1-def image-def*
by (*auto simp*: *map-option-case split*: *option.split*) (*metis fst-conv option.sel*)
show *fst ‘ Some −‘ refresh-1 g ‘ pred A a p ⊆ fst ‘ Some −‘ g ‘ pred A a p*
unfolding *refresh-1-def image-def*
by (*auto simp*: *map-option-case split*: *option.split*) (*force*)
qed
also have $\bigsqcap$(*fst ‘ (Some −‘ refresh-1 g ‘ pred A a p*)) = *k*
using *assms*(*3*) **unfolding** *bounds-3-def* **by** (*auto simp*: *Let-def split*: *if-splits*)
finally show *n* ≤ *k* **by** *this*
have *e* ⟷ $\bigsqcup$(*snd ‘ (Some −‘ refresh-1 f ‘ pred A a p*))
using *assms*(*2*) **unfolding** *bounds-3-def* **by** (*auto simp*: *Let-def split*: *if-splits*)
also have *snd ‘ Some −‘ refresh-1 f ‘ pred A a p = snd ‘ Some −‘ refresh-1 f ‘* (*pred A a p ∩ dom* (*refresh-1 f*))
unfolding *dom-def image-def Int-def* **by** *auto metis*
also have ... = *snd ‘ the ‘ refresh-1 f ‘* (*pred A a p ∩ dom* (*refresh-1 f*))
unfolding *dom-def* **by** *force*
also have ... = (*snd ∘ the ∘ refresh-1 f*) ‘ (*pred A a p ∩ dom* (*refresh-1 f*)) **by** *force*
also have ... = (*snd ∘ the ∘ refresh-1 g*) ‘ (*pred A a p ∩ dom* (*refresh-1 g*))
proof (*rule image-cong*)
show *pred A a p ∩ dom* (*refresh-1 f*) = *pred A a p ∩ dom* (*refresh-1 g*)
unfolding *refresh-1-dom 1*(*1*) **by** *rule*
show (*snd ∘ the ∘ refresh-1 f*) *q* ⟷ (*snd ∘ the ∘ refresh-1 g*) *q*
if *2*: *q* ∈ *pred A a p ∩ dom* (*refresh-1 g*) **for** *q*
proof
have *3*: ∀ *x* ∈ *ran f*. ¬ *snd x* ⟹ (*n*, *True*) ∈ *ran g* ⟹ *g q* = *Some* (*k*, *c*) ⟹ *c* **for** *n k c*
using *1*(*1*, *3*) **unfolding** *dom-def ran-def*
by (*auto dest*!: *Collect-inj*) (*metis option.sel snd-conv*)
have *4*: *g q* = *Some* (*n*, *True*) ⟹ *f q* = *Some* (*k*, *c*) ⟹ *c* **for** *n k c*
using *1*(*3*) **unfolding** *dom-def* **by** *force*
have *5*: ∀ *x* ∈ *ran g*. ¬ *snd x* ⟹ (*k*, *True*) ∈ *ran f* ⟹ *False* **for** *k*
using *1*(*1*, *3*) **unfolding** *dom-def ran-def*
by (*auto dest*!: *Collect-inj*) (*metis option.sel snd-conv*)
show (*snd ∘ the ∘ refresh-1 f*) *q* ⟹ (*snd ∘ the ∘ refresh-1 g*) *q*
using *1*(*1*, *3*) *2 3* **unfolding** *refresh-1-def* **by** (*force split*: *if-splits*)
show (*snd ∘ the ∘ refresh-1 g*) *q* ⟹ (*snd ∘ the ∘ refresh-1 f*) *q*
using *1*(*1*, *3*) *2 4 5* **unfolding** *refresh-1-def*
by (*auto simp*: *map-option-case split*: *option.splits if-splits*) (*force*+)
qed
qed
also have ... = *snd ‘ the ‘ refresh-1 g ‘* (*pred A a p ∩ dom* (*refresh-1 g*)) **by** *force*
also have ... = *snd ‘ Some −‘ refresh-1 g ‘* (*pred A a p ∩ dom* (*refresh-1 g*))
unfolding *dom-def* **by** *force*
also have ... = *snd ‘ Some −‘ refresh-1 g ‘ pred A a p*
unfolding *dom-def image-def Int-def* **by** *auto metis*

```
  also have ⨆(snd ` (Some −` refresh-1 g ` pred A a p)) ⟷ c
   using assms(3) unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
  finally show e ⟷ c by this
 qed

 lemma complement-4-language-1: language (complement-3 A) ⊆ language (complement-4
A)
 proof (rule simulation-language)
  show alphabet (complement-3 A) ⊆ alphabet (complement-4 A)
   unfolding complement-3-def complement-4-def by simp
  show ∃ q ∈ initial (complement-4 A). (p, q) ∈ R if p ∈ initial (complement-3
A) for p
   using that unfolding complement-3-def complement-4-def R-def by simp
  show ∃ g′ ∈ transition (complement-4 A) a g. (f′, g′) ∈ R
   if f′ ∈ transition (complement-3 A) a f (f, g) ∈ R
   for a f f′ g
  proof −
   have 1: f′ ∈ expand-map (get-3 A (bounds-3 A a (refresh-1 f)))
    using that(1) unfolding complement-3-def complement-succ-3-def by auto
   have 2:
    dom f = dom g
    ∀ p ∈ dom f. fst (the (f p)) ≤ fst (the (g p))
    ∀ p ∈ dom f. snd (the (f p)) ⟷ snd (the (g p))
    using that(2) unfolding R-def by auto
    have dom f′ = dom (get-3 A (bounds-3 A a (refresh-1 f))) using ex-
pand-map-dom 1 by this
   also have ... = dom (bounds-3 A a (refresh-1 f)) by simp
   finally have 3: dom f′ = dom (bounds-3 A a (refresh-1 f)) by this
   define g′ where g′ p ≡ do
   {
    (k, c) ← bounds-3 A a (refresh-1 g) p;
    (l, d) ← f′ p;
    Some (if even k = even l then k else k − 1, d)
   } for p
   have 4: g′ p = do
   {
    kc ← bounds-3 A a (refresh-1 g) p;
    ld ← f′ p;
    Some (if even (fst kc) = even (fst ld) then fst kc else fst kc − 1, snd ld)
   } for p unfolding g′-def case-prod-beta by rule
  have dom g′ = dom (bounds-3 A a (refresh-1 g)) ∩ dom f′ using 4 bind-eq-Some-conv
by fastforce
   also have ... = dom f′ using 2 3 by simp
   finally have 5: dom g′ = dom f′ by this
   have 6: (l, d) ∈ items-3 A p (k, c)
    if bounds-3 A a (refresh-1 f) p = Some (k, c) f′ p = Some (l, d) for p k c l
d
    using 1 that unfolding expand-map-alt-def get-3-def by blast
   show ?thesis
```

unfolding *complement-4-def nba.sel complement-succ-4-def comp-apply*
proof
show $(f', g') \in R$
unfolding *R-def mem-Collect-eq prod.case*
proof (*intro conjI ballI*)
show $dom\ f' = dom\ g'$ **using** *5* **by** *rule*
next
fix *p*
assume *10*: $p \in dom\ f'$
have *11*: $p \in dom\ (bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g))$ **using** *2(1) 3 10* **by** *simp*
obtain *k c* **where** *12*: $bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g)\ p = Some\ (k, c)$ **using** *11* **by** *fast*
obtain *l d* **where** *13*: $f'\ p = Some\ (l, d)$ **using** *10* **by** *auto*
obtain *n e* **where** *14*: $bounds\text{-}3\ A\ a\ (refresh\text{-}1\ f)\ p = Some\ (n, e)$ **using** *10 3* **by** *fast*
have *15*: $(l, d) \in items\text{-}3\ A\ p\ (n, e)$ **using** *6 14 13* **by** *this*
have *16*: $n \leq k$ **using** *bounds-R(1) that(2) 14 12* **by** *this*
have *17*: $l \leq k$ **using** *15 16* **unfolding** *items-3-def* **by** *simp*
have *18*: $even\ k \longleftrightarrow odd\ l \Longrightarrow l \leq k \Longrightarrow l \leq k - 1$ **by** *presburger*
have *19*: $e \longleftrightarrow c$ **using** *bounds-R(2) that(2) 14 12* **by** *this*
show $fst\ (the\ (f'\ p)) \leq fst\ (the\ (g'\ p))$ **using** *17 18* **unfolding** *4 12 13* **by** *simp*
show $snd\ (the\ (f'\ p)) \longleftrightarrow snd\ (the\ (g'\ p))$ **using** *19* **unfolding** *4 12 13* **by** *simp*
qed
show $g' \in expand\text{-}map\ (get\text{-}4\ A\ (bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g)))$
unfolding *expand-map-alt-def mem-Collect-eq*
proof (*intro conjI allI impI*)
show $dom\ g' = dom\ (get\text{-}4\ A\ (bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g)))$ **using** *2(1) 3 5* **by** *simp*
fix *p S xy*
assume *10*: $get\text{-}4\ A\ (bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g))\ p = Some\ S$
assume *11*: $g'\ p = Some\ xy$
obtain *k c* **where** *12*: $bounds\text{-}3\ A\ a\ (refresh\text{-}1\ g)\ p = Some\ (k, c)\ S = items\text{-}4\ A\ p\ (k, c)$
using *10* **unfolding** *get-4-def* **by** *auto*
obtain *l d* **where** *13*: $f'\ p = Some\ (l, d)\ xy = (if\ even\ k \longleftrightarrow even\ l\ then\ k\ else\ k - 1, d)$
using *11 12* **unfolding** *g'-def* **by** (*auto split*: *bind-splits*)
obtain *n e* **where** *14*: $bounds\text{-}3\ A\ a\ (refresh\text{-}1\ f)\ p = Some\ (n, e)$ **using** *13(1) 3* **by** *fast*
have *15*: $(l, d) \in items\text{-}3\ A\ p\ (n, e)$ **using** *6 14 13(1)* **by** *this*
have *16*: $n \leq k$ **using** *bounds-R(1) that(2) 14 12(1)* **by** *this*
have *17*: $e \longleftrightarrow c$ **using** *bounds-R(2) that(2) 14 12(1)* **by** *this*
show $xy \in S$ **using** *15 16 17* **unfolding** *12(2) 13(2) items-3-def items-4-def* **by** *auto*
qed
qed
qed

show $\bigwedge p\ q.\ (p,\ q) \in R \Longrightarrow$ *accepting* (*complement-3 A*) $p \Longrightarrow$ *accepting* (*complement-4 A*) *q*
unfolding *complement-3-def complement-4-def R-def map-to-set-def*
by (*auto*) (*metis domIff eq-snd-iff option.exhaust-sel option.sel*)
qed
lemma *complement-4-less*: *complement-4 A* $\leq$ *complement-3 A*
unfolding *less-eq-nba-def*
unfolding *complement-4-def complement-3-def nba.sel*
unfolding *complement-succ-4-def complement-succ-3-def*
proof (*safe intro*!: *le-funI*, *unfold comp-apply*)
fix *a f g*
assume *g* $\in$ *expand-map* (*get-4 A* (*bounds-3 A a* (*refresh-1 f*)))
then show *g* $\in$ *expand-map* (*get-3 A* (*bounds-3 A a* (*refresh-1 f*)))
unfolding *get-4-def get-3-def items-4-def items-3-def expand-map-alt-def* **by** *blast*
qed
lemma *complement-4-language-2*: *language* (*complement-4 A*) $\subseteq$ *language* (*complement-3 A*)
using *language-mono complement-4-less* **by** (*auto dest*: *monoD*)
lemma *complement-4-language*: *language* (*complement-3 A*) = *language* (*complement-4 A*)
using *complement-4-language-1 complement-4-language-2* **by** *blast*

lemma *complement-4-finite*[*simp*]:
assumes *finite* (*nodes A*)
shows *finite* (*nodes* (*complement-4 A*))
proof −
have (*nodes* (*complement-3 A*), *nodes* (*complement-2 A*)) $\in \langle Id \rangle$ *set-rel*
using *complement-3-refine* **by** *parametricity auto*
also have (*nodes* (*complement-2 A*), *nodes* (*complement-1 A*)) $\in \langle Id \rangle$ *set-rel*
using *complement-2-refine* **by** *parametricity auto*
also have (*nodes* (*complement-1 A*), *nodes* (*complement A*)) $\in \langle$*cs-rel*$\rangle$ *set-rel*
using *complement-1-refine* **by** *parametricity auto*
finally have *1*: (*nodes* (*complement-3 A*), *nodes* (*complement A*)) $\in \langle$*cs-rel*$\rangle$ *set-rel* **by** *simp*
have *2*: *finite* (*nodes* (*complement A*)) **using** *complement-finite assms*(*1*) **by** *this*
have *3*: *finite* (*nodes* (*complement-3 A*))
using *finite-set-rel-transfer-back 1 cs-rel-inv-single-valued 2* **by** *this*
have *4*: *nodes* (*complement-4 A*) $\subseteq$ *nodes* (*complement-3 A*)
using *nodes-mono complement-4-less* **by** (*auto dest*: *monoD*)
show *finite* (*nodes* (*complement-4 A*)) **using** *finite-subset 4 3* **by** *this*
qed
lemma *complement-4-correct*:
assumes *finite* (*nodes A*)
shows *language* (*complement-4 A*) = *streams* (*alphabet A*) − *language A*
proof −
have *language* (*complement-4 A*) = *language* (*complement-3 A*)
using *complement-4-language* **by** *rule*

also have (*language* (*complement-3 A*), *language* (*complement-2 A*)) ∈ ⟨⟨*Id*⟩ *stream-rel*⟩ *set-rel*
using *complement-3-refine* **by** *parametricity auto*
also have (*language* (*complement-2 A*), *language* (*complement-1 A*)) ∈ ⟨⟨*Id*⟩ *stream-rel*⟩ *set-rel*
using *complement-2-refine* **by** *parametricity auto*
also have (*language* (*complement-1 A*), *language* (*complement A*)) ∈ ⟨⟨*Id*⟩ *stream-rel*⟩ *set-rel*
using *complement-1-refine* **by** *parametricity auto*
also have *language* (*complement A*) = *streams* (*alphabet A*) − *language A*
using *complement-language assms*(*1*) **by** *this*
finally show *language* (*complement-4 A*) = *streams* (*alphabet A*) − *language A* **by** *simp*
qed

5.5 Phase 5

definition *refresh-5* :: *'state items* ⇒ *'state items nres* **where**
refresh-5 f ≡ *if* ∃ (*p*, *k*, *c*) ∈ *map-to-set f*. *c*
then RETURN f
else do
{
ASSUME (*finite* (*dom f*));
FOREACH (*map-to-set f*) (λ (*p*, *k*, *c*) *m*. *do*
{
ASSERT (*p* ∉ *dom m*);
RETURN (*m* (*p* ↦ (*k*, *True*)))
}
) *Map.empty*
}
definition *merge-5* :: *item* ⇒ *item option* ⇒ *item* **where**
merge-5 ≡ λ (*k*, *c*). λ *None* ⇒ (*k*, *c*) | *Some* (*l*, *d*) ⇒ (*k* ⊓ *l*, *c* ⊔ *d*)
definition *bounds-5* :: (*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state items* ⇒ *'state items nres* **where**
bounds-5 A a f ≡ *do*
{
ASSUME (*finite* (*dom f*));
ASSUME (∀ *p*. *finite* (*transition A a p*));
FOREACH (*map-to-set f*) (λ (*p*, *s*) *m*.
FOREACH (*transition A a p*) (λ *q f*.
RETURN (*f* (*q* ↦ *merge-5 s* (*f q*))))
m)
Map.empty
}
definition *items-5* :: (*'label*, *'state*) *nba* ⇒ *'state* ⇒ *item* ⇒ *item set* **where**
items-5 A p ≡ λ (*k*, *c*). *do*
{
let values = *if accepting A p then Set.filter even* {*k* − *1* .. *k*} *else* {*k* − *1* .. *k*};

```
    let item = λ l. (l, c ∧ even l);
    item ‘ values
  }
 definition get-5 :: ('label, 'state) nba ⇒ 'state items ⇒ ('state ⇀ item set)
where
  get-5 A f ≡ λ p. map-option (items-5 A p) (f p)
 definition expand-5 :: ('a ⇀ 'b set) ⇒ ('a ⇀ 'b) set nres where
  expand-5 f ≡ FOREACH (map-to-set f) (λ (x, S) X. do {
    ASSERT (∀ g ∈ X. x ∉ dom g);
    ASSERT (∀ a ∈ S. ∀ b ∈ S. a ≠ b ⟶ (λ y. (λ g. g (x ↦ y)) ‘ X) a ∩ (λ
y. (λ g. g (x ↦ y)) ‘ X) b = {});
    RETURN (⋃ y ∈ S. (λ g. g (x ↦ y)) ‘ X)
  }) {Map.empty}
 definition complement-succ-5 ::
  ('label, 'state) nba ⇒ 'label ⇒ 'state items ⇒ 'state items set nres where
  complement-succ-5 A a f ≡ do
  {
   f ← refresh-5 f;
   f ← bounds-5 A a f;
   ASSUME (finite (dom f));
   expand-5 (get-5 A f)
  }

 lemma bounds-3-empty: bounds-3 A a Map.empty = Map.empty
  unfolding bounds-3-def Let-def by auto
 lemma bounds-3-update: bounds-3 A a (f (p ↦ s)) =
   override-on (bounds-3 A a f) (Some ∘ merge-5 s ∘ bounds-3 A a (f (p :=
None))) (transition A a p)
 proof
  note fun-upd-image[simp]
  fix q
  show bounds-3 A a (f (p ↦ s)) q =
    override-on (bounds-3 A a f) (Some ∘ merge-5 s ∘ bounds-3 A a (f (p :=
None))) (transition A a p) q
  proof (cases q ∈ transition A a p)
   case True
   define S where S ≡ Some −‘ f ‘ (pred A a q − {p})
    have 1: Some −‘ f (p := Some s) ‘ pred A a q = insert s S using True
unfolding S-def by auto
   have 2: Some −‘ f (p := None) ‘ pred A a q = S unfolding S-def by auto
    have bounds-3 A a (f (p ↦ s)) q = Some (⨅(fst ‘ (insert s S)), ⨆(snd ‘
(insert s S)))
     unfolding bounds-3-def 1 by simp
   also have ... = Some (merge-5 s (bounds-3 A a (f (p := None)) q))
   unfolding 2 bounds-3-def merge-5-def by (cases s) (simp-all add: cInf-insert)
   also have ... = override-on (bounds-3 A a f) (Some ∘ merge-5 s ∘ bounds-3
A a (f (p := None)))
     (transition A a p) q using True by simp
   finally show ?thesis by this
```

```
    next
      case False
      then have pred A a q ∩ {x. x ≠ p} = pred A a q
        by auto
      with False show ?thesis by (simp add: bounds-3-def)
    qed
  qed

  lemma refresh-5-refine: (refresh-5, λ f. RETURN (refresh-1 f)) ∈ Id → ⟨Id⟩
nres-rel
  proof safe
    fix f :: 'a ⇒ item option
    have 1: (∃ (p, k, c) ∈ map-to-set f. c) ⟷ True ∈ snd ‘ ran f
      unfolding image-def map-to-set-def ran-def by force
    show (refresh-5 f, RETURN (refresh-1 f)) ∈ ⟨Id⟩ nres-rel
      unfolding refresh-5-def refresh-1-def 1
      by (refine-vcg FOREACH-rule-map-eq[where X = λ m. map-option (apsnd
⊤) ∘ m]) (auto)
  qed
  lemma bounds-5-refine: (bounds-5 A a, λ f. RETURN (bounds-3 A a f)) ∈ Id
→ ⟨Id⟩ nres-rel
    unfolding bounds-5-def by
    (refine-vcg FOREACH-rule-map-eq[where X = bounds-3 A a] FOREACH-rule-insert-eq)
      (auto simp: override-on-insert bounds-3-empty bounds-3-update)
  lemma items-5-refine: items-5 = items-4
    unfolding items-5-def items-4-def by (intro ext) (auto split: if-splits)
  lemma get-5-refine: get-5 = get-4
    unfolding get-5-def get-4-def items-5-refine by rule
  lemma expand-5-refine: (expand-5 f, ASSERT (finite (dom f)) ≫ RETURN
(expand-map f)) ∈ ⟨Id⟩ nres-rel
    unfolding expand-5-def
    by (refine-vcg FOREACH-rule-map-eq[where X = expand-map]) (auto dest!:
expand-map-dom map-upd-eqD1)

  lemma complement-succ-5-refine: (complement-succ-5, RETURN ∘∘∘ comple-
ment-succ-4) ∈
    Id → Id → Id → ⟨Id⟩ nres-rel
  unfolding complement-succ-5-def complement-succ-4-def get-5-refine comp-apply
    by (refine-vcg vcg1[OF refresh-5-refine] vcg1[OF bounds-5-refine] vcg0[OF ex-
pand-5-refine]) (auto)
```

5.6 Phase 6

```
  definition expand-map-get-6 :: ('label, 'state) nba ⇒ 'state items ⇒ 'state items
set nres where
    expand-map-get-6 A f ≡ FOREACH (map-to-set f) (λ (k, v) X. do {
      ASSERT (∀ g ∈ X. k ∉ dom g);
      ASSERT (∀ a ∈ (items-5 A k v). ∀ b ∈ (items-5 A k v). a ≠ b ⟶ (λ y. (λ
g. g (k ↦ y)) ‘ X) a ∩ (λ y. (λ g. g (k ↦ y)) ‘ X) b = {});
```

RETURN ($\bigcup$ *y* ∈ *items-5 A k v*. (λ *g*. *g* (*k* ↦ *y*)) ‘ *X*)
}) {*Map.empty*}

lemma *expand-map-get-6-refine*: (*expand-map-get-6*, *expand-5* ∘∘ *get-5*) ∈ *Id* → *Id* → ⟨*Id*⟩ *nres-rel*
unfolding *expand-map-get-6-def expand-5-def get-5-def* **by** (*auto intro*: *FOREACH-rule-map-map*[*param-fo*])

definition *complement-succ-6* ::
(*'label*, *'state*) *nba* ⇒ *'label* ⇒ *'state items* ⇒ *'state items set nres* **where**
complement-succ-6 A a f ≡ *do*
{
f ← *refresh-5 f*;
f ← *bounds-5 A a f*;
ASSUME (*finite* (*dom f*));
expand-map-get-6 A f
}

lemma *complement-succ-6-refine*:
(*complement-succ-6*, *complement-succ-5*) ∈ *Id* → *Id* → *Id* → ⟨*Id*⟩ *nres-rel*
unfolding *complement-succ-6-def complement-succ-5-def*
by (*refine-vcg vcg2*[*OF expand-map-get-6-refine*]) (*auto intro*: *refine-IdI*)

5.7 Phase 7

interpretation *autoref-syn* **by** *this*

context
fixes *fi f*
assumes *fi*[*autoref-rules*]: (*fi*, *f*) ∈ *state-rel*
begin

private lemma [*simp*]: *finite* (*dom f*)
using *list-map-rel-finite fi* **unfolding** *finite-map-rel-def* **by** *force*

schematic-goal *refresh-7*: (*?f* :: *?'a*, *refresh-5 f*) ∈ *?R*
unfolding *refresh-5-def* **by** (*autoref-monadic* (*plain*))

end

concrete-definition *refresh-7* **uses** *refresh-7*

lemma *refresh-7-refine*: (λ *f*. *RETURN* (*refresh-7 f*), *refresh-5*) ∈ *state-rel* → ⟨*state-rel*⟩ *nres-rel*
using *refresh-7.refine* **by** *fast*

context
fixes *A* :: (*'label*, *nat*) *nba*
fixes *succi a fi f*

assumes *succi*[*autoref-rules*]: (*succi*, *transition A a*) ∈ *nat-rel* → ⟨*nat-rel*⟩ *list-set-rel*
assumes *fi*[*autoref-rules*]: (*fi*, *f*) ∈ *state-rel*
begin

private lemma [*simp*]: *finite* (*transition A a p*)
using *list-set-rel-finite succi*[*param-fo*] **unfolding** *finite-set-rel-def* **by** *blast*
private lemma [*simp*]: *finite* (*dom f*) **using** *fi* **by** *force*

private lemma [*autoref-op-pat*]: *transition A a* ≡ *OP* (*transition A a*) **by** *simp*

private lemma [*autoref-rules*]: (*min*, *min*) ∈ *nat-rel* → *nat-rel* → *nat-rel* **by** *simp*

schematic-goal *bounds-7*:
notes *ty-REL*[**where** *R* = ⟨*nat-rel*, *item-rel*⟩ *dflt-ahm-rel*, *autoref-tyrel*]
shows (*?f* :: *?'a*, *bounds-5 A a f*) ∈ *?R*
unfolding *bounds-5-def merge-5-def sup-bool-def inf-nat-def* **by** (*autoref-monadic* (*plain*))

end

concrete-definition *bounds-7* **uses** *bounds-7*

lemma *bounds-7-refine*: (*si*, *transition A a*) ∈ *nat-rel* → ⟨*nat-rel*⟩ *list-set-rel* ⟹
(λ *p*. *RETURN* (*bounds-7 si p*), *bounds-5 A a*) ∈
state-rel → ⟨⟨*nat-rel*, *item-rel*⟩ *dflt-ahm-rel*⟩ *nres-rel*
using *bounds-7.refine* **by** *auto*

context
fixes *A* :: (*'label*, *nat*) *nba*
fixes *acci*
assumes [*autoref-rules*]: (*acci*, *accepting A*) ∈ *nat-rel* → *bool-rel*
begin

private lemma [*autoref-op-pat*]: *accepting A* ≡ *OP* (*accepting A*) **by** *simp*

private lemma [*autoref-rules*]: ((*dvd*), (*dvd*)) ∈ *nat-rel* → *nat-rel* → *bool-rel* **by** *simp*
private lemma [*autoref-rules*]: (λ *k l*. *upt k* (*Suc l*), *atLeastAtMost*) ∈
nat-rel → *nat-rel* → ⟨*nat-rel*⟩ *list-set-rel*
by (*auto simp*: *list-set-rel-def in-br-conv*)

schematic-goal *items-7*: (*?f* :: *?'a*, *items-5 A*) ∈ *?R*
unfolding *items-5-def Let-def Set.filter-eq* **by** *autoref*

end

concrete-definition *items-7* **uses** *items-7*

context
fixes $A :: ('label, nat)\ nba$
fixes ai
fixes $fi\ f$
assumes ai: $(ai, accepting\ A) \in nat\text{-}rel \to bool\text{-}rel$
assumes $fi[autoref\text{-}rules]$: $(fi, f) \in \langle nat\text{-}rel, item\text{-}rel\rangle\ dflt\text{-}ahm\text{-}rel$
begin

private lemma [*simp*]: $finite\ (dom\ f)$
using *dflt-ahm-rel-finite-nat fi* **unfolding** *finite-map-rel-def* **by** *force*
private lemma [*simp*]:
assumes $\bigwedge\ m.\ m \in S \Longrightarrow x \notin dom\ m$
shows $inj\text{-}on\ (\lambda\ m.\ m\ (x \mapsto y))\ S$
using *assms* **unfolding** *dom-def inj-on-def* **by** (*auto*) (*metis fun-upd-triv fun-upd-upd*)
private lemmas [*simp*] = *op-map-update-def*[*abs-def*]

private lemma [*autoref-op-pat*]: $items\text{-}5\ A \equiv OP\ (items\text{-}5\ A)$ **by** *simp*

private lemmas [*autoref-rules*] = *items-7.refine*[*OF ai*]

schematic-goal *expand-map-get-7*: $(?f, expand\text{-}map\text{-}get\text{-}6\ A\ f) \in$
$\langle\langle state\text{-}rel\rangle\ list\text{-}set\text{-}rel\rangle\ nres\text{-}rel$
unfolding *expand-map-get-6-def* **by** (*autoref-monadic* (*plain*))

end

concrete-definition *expand-map-get-7* **uses** *expand-map-get-7*

lemma *expand-map-get-7-refine*:
assumes $(ai, accepting\ A) \in nat\text{-}rel \to bool\text{-}rel$
shows $(\lambda\ fi.\ RETURN\ (expand\text{-}map\text{-}get\text{-}7\ ai\ fi),$
$\lambda\ f.\ ASSUME\ (finite\ (dom\ f)) \gg expand\text{-}map\text{-}get\text{-}6\ A\ f) \in$
$\langle nat\text{-}rel, item\text{-}rel\rangle\ dflt\text{-}ahm\text{-}rel \to \langle\langle state\text{-}rel\rangle\ list\text{-}set\text{-}rel\rangle\ nres\text{-}rel$
using *expand-map-get-7.refine*[*OF assms*] **by** *auto*

context
fixes $A :: ('label, nat)\ nba$
fixes $a :: 'label$
fixes $p :: nat\ items$
fixes Ai
fixes ai
fixes pi
assumes Ai: $(Ai, A) \in \langle Id, Id\rangle\ nbai\text{-}nba\text{-}rel$
assumes ai: $(ai, a) \in Id$
assumes $pi[autoref\text{-}rules]$: $(pi, p) \in state\text{-}rel$
begin

private lemmas *succi* = *nbai-nba-param*(*4*)[*THEN fun-relD*, *OF Ai*, *THEN fun-relD*, *OF ai*]
private lemmas *acceptingi* = *nbai-nba-param*(*5*)[*THEN fun-relD*, *OF Ai*]

private lemma [*autoref-op-pat*]: (λ *g*. *ASSUME* (*finite* (*dom g*)) ≫ *expand-map-get-6 A g*) ≡
OP (λ *g*. *ASSUME* (*finite* (*dom g*)) ≫ *expand-map-get-6 A g*) **by** *simp*
private lemma [*autoref-op-pat*]: *bounds-5 A a* ≡ *OP* (*bounds-5 A a*) **by** *simp*

private lemmas [*autoref-rules*] =
refresh-7-refine
bounds-7-refine[*OF succi*]
expand-map-get-7-refine[*OF acceptingi*]

schematic-goal *complement-succ-7*: (*?f* :: *?'a*, *complement-succ-6 A a p*) ∈ *?R*
unfolding *complement-succ-6-def* **by** (*autoref-monadic* (*plain*))

end

concrete-definition *complement-succ-7* **uses** *complement-succ-7*

lemma *complement-succ-7-refine*:
(*RETURN* ooo *complement-succ-7*, *complement-succ-6*) ∈
⟨*Id*, *Id*⟩ *nbai-nba-rel* → *Id* → *state-rel* →
⟨⟨*state-rel*⟩ *list-set-rel*⟩ *nres-rel*
using *complement-succ-7.refine* **unfolding** *comp-apply* **by** *parametricity*

context
fixes *A* :: (*'label*, *nat*) *nba*
fixes *Ai*
fixes *n ni* :: *nat*
assumes *Ai*: (*Ai*, *A*) ∈ ⟨*Id*, *Id*⟩ *nbai-nba-rel*
assumes *ni*[*autoref-rules*]: (*ni*, *n*) ∈ *Id*
begin

private lemma [*autoref-op-pat*]: *initial A* ≡ *OP* (*initial A*) **by** *simp*

private lemmas [*autoref-rules*] = *nbai-nba-param*(*3*)[*THEN fun-relD*, *OF Ai*]

schematic-goal *complement-initial-7*:
(*?f*, {(*Some* ∘ (*const* (*2* ∗ *n*, *False*))) |` *initial A*}) ∈ ⟨*state-rel*⟩ *list-set-rel*
by *autoref*

end

concrete-definition *complement-initial-7* **uses** *complement-initial-7*

schematic-goal *complement-accepting-7*: (*?f*, λ *f*. ∀ (*p*, *k*, *c*) ∈ *map-to-set f*. ¬ *c*) ∈
state-rel → *bool-rel*
by *autoref*

concrete-definition *complement-accepting-7* **uses** *complement-accepting-7*

definition *complement-7* :: ('*label*, *nat*) *nbai* ⇒ *nat* ⇒ ('*label*, *state*) *nbai* **where**
complement-7 Ai ni ≡ *nbai*
(*alphabeti Ai*)
(*complement-initial-7 Ai ni*)
(*complement-succ-7 Ai*)
(*complement-accepting-7*)

lemma *complement-7-refine*[*autoref-rules*]:
assumes (*Ai*, *A*) ∈ ⟨*Id*, *Id*⟩ *nbai-nba-rel*
assumes (*ni*,
(*OP card* ::: ⟨*Id*⟩ *ahs-rel bhc* → *nat-rel*) $
((*OP nodes* ::: ⟨*Id*, *Id*⟩ *nbai-nba-rel* → ⟨*Id*⟩ *ahs-rel bhc*) $ *A*)) ∈ *nat-rel*
shows (*complement-7 Ai ni*, (*OP complement-4* :::
⟨*Id*, *Id*⟩ *nbai-nba-rel* → ⟨*Id*, *state-rel*⟩ *nbai-nba-rel*) $ *A*) ∈ ⟨*Id*, *state-rel*⟩ *nbai-nba-rel*
proof −
note *complement-succ-7-refine*
also note *complement-succ-6-refine*
also note *complement-succ-5-refine*
finally have *1*: (*complement-succ-7*, *complement-succ-4*) ∈
⟨*Id*, *Id*⟩ *nbai-nba-rel* → *Id* → *state-rel* → ⟨*state-rel*⟩ *list-set-rel*
unfolding *nres-rel-comp* **unfolding** *nres-rel-def* **unfolding** *fun-rel-def* **by** *auto*
show *?thesis*
unfolding *complement-7-def complement-4-def*
using *1 complement-initial-7.refine complement-accepting-7.refine assms*
unfolding *autoref-tag-defs*
by *parametricity*
qed

end

6 Boolean Formulae

theory *Formula*
imports *Main*
begin

datatype '*a formula* =
False |
True |
Variable '*a* |

Negation 'a formula |
Conjunction 'a formula 'a formula |
Disjunction 'a formula 'a formula

primrec *satisfies* :: *'a set* ⇒ *'a formula* ⇒ *bool* **where**
satisfies A False ⟷ *HOL.False* |
satisfies A True ⟷ *HOL.True* |
satisfies A (*Variable a*) ⟷ *a* ∈ *A* |
satisfies A (*Negation x*) ⟷ ¬ *satisfies A x* |
satisfies A (*Conjunction x y*) ⟷ *satisfies A x* ∧ *satisfies A y* |
satisfies A (*Disjunction x y*) ⟷ *satisfies A x* ∨ *satisfies A y*

end

7 Final Instantiation of Algorithms Related to Complementation

theory *Complementation-Final*
imports
Complementation-Implement
Formula
Transition-Systems-and-Automata.NBA-Translate
Transition-Systems-and-Automata.NGBA-Algorithms
HOL−Library.Multiset
begin

7.1 Syntax

no-syntax *-do-let* :: [*pttrn*, *'a*] ⇒ *do-bind* (‹(‹*indent=2 notation=‹infix do let››let - =/ -*)› [*1000*, *13*] *13*)
syntax *-do-let* :: [*pttrn*, *'a*] ⇒ *do-bind* (‹(‹*indent=2 notation=‹infix do let››let - =/ -*)› *13*)

7.2 Hashcodes on Complement States

definition *hci k* ≡ *uint32-of-nat k* ∗ *1103515245* + *12345*
definition *hc* ≡ λ (*p*, *q*, *b*). *hci p* + *hci q* ∗ *31* + (*if b then 1 else 0*)
definition *list-hash xs* ≡ *fold* (*xor* ∘ *hc*) *xs 0*

lemma *list-hash-eq*:
assumes *distinct xs distinct ys set xs* = *set ys*
shows *list-hash xs* = *list-hash ys*
proof −
have *mset* (*remdups xs*) = *mset* (*remdups ys*) **using** *assms*(*3*)
using *set-eq-iff-mset-remdups-eq* **by** *blast*
then have *mset xs* = *mset ys* **using** *assms*(*1*, *2*) **by** (*simp add*: *distinct-remdups-id*)
have *fold* (*xor* ∘ *hc*) *xs* = *fold* (*xor* ∘ *hc*) *ys*
apply (*rule fold-multiset-equiv*)
apply (*simp-all add*: *fun-eq-iff ac-simps*)

using ‹*mset xs = mset ys*› **.**
then show *?thesis* **unfolding** *list-hash-def* **by** *simp*
qed

definition *state-hash* :: *nat* ⇒ *Complementation-Implement.state* ⇒ *nat* **where**
state-hash n p ≡ *nat-of-hashcode* (*list-hash p*) *mod n*

lemma *state-hash-bounded-hashcode*[*autoref-ga-rules*]: *is-bounded-hashcode state-rel*
(*gen-equals* (*Gen-Map.gen-ball* (*foldli* ∘ *list-map-to-list*)) (*list-map-lookup* (=))
(*prod-eq* (=) (⟷))) *state-hash*
proof
show [*param*]: (*gen-equals* (*Gen-Map.gen-ball* (*foldli* ∘ *list-map-to-list*)) (*list-map-lookup* (=))
(*prod-eq* (=) (⟷)), (=)) ∈ *state-rel* → *state-rel* → *bool-rel* **by** *autoref*
show *state-hash n xs = state-hash n ys* **if** *xs* ∈ *Domain state-rel ys* ∈ *Domain state-rel*
gen-equals (*Gen-Map.gen-ball* (*foldli* ∘ *list-map-to-list*))
(*list-map-lookup* (=)) (*prod-eq* (=) (=)) *xs ys* **for** *xs ys n*
proof −
have *1*: *distinct* (*map fst xs*) *distinct* (*map fst ys*)
using *that*(*1*, *2*) **unfolding** *list-map-rel-def list-map-invar-def* **by** (*auto simp*: *in-br-conv*)
have *2*: *distinct xs distinct ys* **using** *1* **by** (*auto intro*: *distinct-mapI*)
have *3*: (*xs*, *map-of xs*) ∈ *state-rel* (*ys*, *map-of ys*) ∈ *state-rel*
using *1* **unfolding** *list-map-rel-def list-map-invar-def* **by** (*auto simp*: *in-br-conv*)
have *4*: (*gen-equals* (*Gen-Map.gen-ball* (*foldli* ∘ *list-map-to-list*)) (*list-map-lookup* (=))
(*prod-eq* (=) (⟷)) *xs ys*, *map-of xs = map-of ys*) ∈ *bool-rel* **using** *3* **by** *parametricity*
have *5*: *map-to-set* (*map-of xs*) = *map-to-set* (*map-of ys*) **using** *that*(*3*) *4* **by** *simp*
have *6*: *set xs = set ys* **using** *map-to-set-map-of 1 5* **by** *blast*
show *state-hash n xs = state-hash n ys* **unfolding** *state-hash-def* **using** *list-hash-eq 2 6* **by** *metis*
qed
show *state-hash n x < n* **if** *1 < n* **for** *n x* **using** *that* **unfolding** *state-hash-def* **by** *simp*
qed

7.3 Complementation

schematic-goal *complement-impl*:
assumes [*simp*]: *finite* (*NBA.nodes A*)
assumes [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*?f* :: *?'c*, *op-translate* (*complement-4 A*)) ∈ *?R*
by (*autoref-monadic* (*plain*))
concrete-definition *complement-impl* **uses** *complement-impl*

theorem *complement-impl-correct*:
assumes *finite* (*NBA.nodes A*)
assumes (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows *NBA.language* (*nbae-nba* (*nbaei-nbae* (*complement-impl Ai*))) =
streams (*nba.alphabet A*) − *NBA.language A*
using *op-translate-language*[*OF complement-impl.refine*[*OF assms*]]
using *complement-4-correct*[*OF assms*(*1*)]
by *simp*

7.4 Language Subset

definition [*simp*]: *op-language-subset A B* ≡ *NBA.language A* ⊆ *NBA.language B*

lemmas [*autoref-op-pat*] = *op-language-subset-def*[*symmetric*]

schematic-goal *language-subset-impl*:
assumes [*simp*]: *finite* (*NBA.nodes B*)
assumes [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
assumes [*autoref-rules*]: (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*?f* :: *?'c*, *do* {
let AB' = *intersect' A* (*complement-4 B*);
ASSERT (*finite* (*NGBA.nodes AB'*));
RETURN (*NGBA.language AB'* = {})
}) ∈ *?R*
by (*autoref-monadic* (*plain*))
concrete-definition *language-subset-impl* **uses** *language-subset-impl*
lemma *language-subset-impl-refine*[*autoref-rules*]:
assumes *SIDE-PRECOND* (*finite* (*NBA.nodes A*))
assumes *SIDE-PRECOND* (*finite* (*NBA.nodes B*))
assumes *SIDE-PRECOND* (*nba.alphabet A* ⊆ *nba.alphabet B*)
assumes (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
assumes (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*language-subset-impl Ai Bi*, (*OP op-language-subset* :::
⟨*Id*, *nat-rel*⟩ *nbai-nba-rel* → ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel* → *bool-rel*) $ *A* $ *B*) ∈ *bool-rel*
proof −
have (*RETURN* (*language-subset-impl Ai Bi*), *do* {
let AB' = *intersect' A* (*complement-4 B*);
ASSERT (*finite* (*NGBA.nodes AB'*));
RETURN (*NGBA.language AB'* = {})
}) ∈ ⟨*bool-rel*⟩ *nres-rel*
using *language-subset-impl.refine assms*(*2*, *4*, *5*) **unfolding** *autoref-tag-defs* **by** *this*
also have (*do* {
let AB' = *intersect' A* (*complement-4 B*);
ASSERT (*finite* (*NGBA.nodes AB'*));
RETURN (*NGBA.language AB'* = {})
}, *RETURN* (*NBA.language A* ⊆ *NBA.language B*)) ∈ ⟨*bool-rel*⟩ *nres-rel*

proof *refine-vcg*
show *finite* (*NGBA.nodes* (*intersect′ A* (*complement-4 B*))) **using** *assms*(*1*, *2*) **by** *auto*
have *1*: *NBA.language A* ⊆ *streams* (*nba.alphabet B*)
using *nba.language-alphabet streams-mono2 assms*(*3*) **unfolding** *autoref-tag-defs* **by** *blast*
have *2*: *NBA.language* (*complement-4 B*) = *streams* (*nba.alphabet B*) − *NBA.language B*
using *complement-4-correct assms*(*2*) **by** *auto*
show (*NGBA.language* (*intersect′ A* (*complement-4 B*)) = {},
NBA.language A ⊆ *NBA.language B*) ∈ *bool-rel* **using** *1 2* **by** *auto*
qed
finally show *?thesis* **using** *RETURN-nres-relD* **unfolding** *nres-rel-comp* **by** *force*
qed

7.5 Language Equality

definition [*simp*]: *op-language-equal A B* ≡ *NBA.language A* = *NBA.language B*

lemmas [*autoref-op-pat*] = *op-language-equal-def*[*symmetric*]

schematic-goal *language-equal-impl*:
assumes [*simp*]: *finite* (*NBA.nodes A*)
assumes [*simp*]: *finite* (*NBA.nodes B*)
assumes [*simp*]: *nba.alphabet A* = *nba.alphabet B*
assumes [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
assumes [*autoref-rules*]: (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*?f* :: *?′c*, *NBA.language A* ⊆ *NBA.language B* ∧ *NBA.language B* ⊆ *NBA.language A*) ∈ *?R*
by *autoref*
concrete-definition *language-equal-impl* **uses** *language-equal-impl*
lemma *language-equal-impl-refine*[*autoref-rules*]:
assumes *SIDE-PRECOND* (*finite* (*NBA.nodes A*))
assumes *SIDE-PRECOND* (*finite* (*NBA.nodes B*))
assumes *SIDE-PRECOND* (*nba.alphabet A* = *nba.alphabet B*)
assumes (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
assumes (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*language-equal-impl Ai Bi*, (*OP op-language-equal* :::
⟨*Id*, *nat-rel*⟩ *nbai-nba-rel* → ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel* → *bool-rel*) $ *A* $ *B*) ∈ *bool-rel*
using *language-equal-impl.refine*[*OF assms*[*unfolded autoref-tag-defs*]] **by** *auto*

schematic-goal *product-impl*:
assumes [*simp*]: *finite* (*NBA.nodes B*)
assumes [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
assumes [*autoref-rules*]: (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
shows (*?f* :: *?′c*, *do* {

```
    let AB' = intersect A (complement-4 B);
    ASSERT (finite (NBA.nodes AB'));
    op-translate AB'
  }) ∈ ?R
 by (autoref-monadic (plain))
 concrete-definition product-impl uses product-impl


 export-code
   Set.empty Set.insert Set.member
   Inf :: 'a set set ⇒ 'a set Sup :: 'a set set ⇒ 'a set image Pow set
   nat-of-integer integer-of-nat
   Variable Negation Conjunction Disjunction satisfies map-formula
   nbaei alphabetei initialei transitionei acceptingei
   nbae-nba-impl complement-impl language-equal-impl product-impl
   in SML module-name Complementation file-prefix Complementation

end
```

8 Build and test exported program with MLton

```
theory Complementation-Build
imports Complementation-Final
begin

external-file ‹code/Autool.mlb›
external-file ‹code/Prelude.sml›
external-file ‹code/Autool.sml›

compile-generated-files
 ‹code/Complementation.ML› (in Complementation-Final)
 external-files
   ‹code/Autool.mlb›
   ‹code/Prelude.sml›
   ‹code/Autool.sml›
 export-files ‹code/Complementation.sml› and ‹code/Autool› (exe)
 where ‹fn dir =>
   let
     val exec = Generated-Files.execute (dir + Path.basic code);
     val - = exec ‹Prepare› mv Complementation.ML Complementation.sml;
   val - = exec ‹Compilation› (verbatim ‹$ISABELLE-MLTON $ISABELLE-MLTON-OPTIONS
› ^
           −profile time −default−type intinf Autool.mlb);
     val - = exec ‹Test› ./Autool help;
   in () end›

end
```

References

[1] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, July 2001.