

# Formalisation of an Adaptive State Counting Algorithm

Robert Sachtleben

February 6, 2026

## Abstract

This entry provides a formalisation of a refinement of an adaptive state counting algorithm, used to test for reduction between finite state machines. The algorithm has been originally presented by Hierons in [2] and was slightly refined by Sachtleben et al. in [3]. Definitions for finite state machines and adaptive test cases are given and many useful theorems are derived from these. The algorithm is formalised using mutually recursive functions, for which it is proven that the generated test suite is sufficient to test for reduction against finite state machines of a certain fault domain. Additionally, the algorithm is specified in a simple WHILE-language and its correctness is shown using Hoare-logic.

## Contents

<b>1</b>	<b>Finite state machines</b>	<b>2</b>
1.1	FSMs as transition systems	2
1.2	Language	2
1.3	Product machine for language intersection	4
1.4	Required properties	5
1.5	States reached by a given IO-sequence	11
1.6	D-reachability	17
1.7	Deterministic state cover	18
1.8	IO reduction	19
1.9	Language subsets for input sequences	20
1.10	Sequences to failures	23
1.11	Minimal sequence to failure extending	29
1.12	Complete test suite derived from the product machine	30
<b>2</b>	<b>Product machines with an additional fail state</b>	<b>31</b>
2.1	Sequences to failure in the product machine	46
<b>3</b>	<b>Adaptive test cases</b>	<b>56</b>
3.1	Properties of ATC-reactions	56
3.2	Applicability	57
3.3	Application function IO	58
3.4	R-distinguishability	58
3.5	Response sets	59
3.6	Characterizing sets	61
3.7	Reduction over ATCs	61
3.8	Reduction over ATCs applied after input sequences	62
<b>4</b>	<b>The lower bound function</b>	<b>67</b>
4.1	Permutation function Perm	68
4.2	Helper predicates	69
4.3	Function R	72
4.4	Function RP	82
4.5	Conditions for the result of LB to be a valid lower bound	92
4.6	Function LB	99
4.7	Validity of the result of LB constituting a lower bound	108

<b>5</b>	<b>Test suite generated by the Adaptive State Counting Algorithm</b>	<b>111</b>
5.1	Maximum length contained prefix . . . . .	111
5.2	Function N . . . . .	113
5.3	Functions TS, C, RM . . . . .	116
5.4	Basic properties of the test suite calculation functions . . . . .	117
5.5	Final iteration . . . . .	133
<b>6</b>	<b>Sufficiency of the test suite to test for reduction</b>	<b>139</b>
6.1	Properties of minimal sequences to failures extending the deterministic state cover . . . . .	139
6.2	Sufficiency of the test suite to test for reduction . . . . .	145
6.3	Main result . . . . .	152
<b>7</b>	<b>Correctness of the Adaptive State Counting Algorithm in Hoare-Logic</b>	<b>152</b>
<b>8</b>	<b>Example product machines and properties</b>	<b>167</b>
8.1	Constructing FSMs from transition relations . . . . .	167
8.2	Example FSMs and properties . . . . .	169

```

theory FSM
imports
  Transition-Systems-and-Automata.Sequence-Zip
  Transition-Systems-and-Automata.Transition-System
  Transition-Systems-and-Automata.Transition-System-Extra
  Transition-Systems-and-Automata.Transition-System-Construction
begin

```

## 1 Finite state machines

We formalise finite state machines as a 4-tuples, omitting the explicit formulation of the state set, as it can easily be calculated from the successor function. This definition does not require the successor function to be restricted to the input or output alphabet, which is later expressed by the property `well_formed`, together with the finiteness of the state set.

```

record ('in, 'out, 'state) FSM =
  succ    :: ('in × 'out) ⇒ 'state ⇒ 'state set
  inputs  :: 'in set
  outputs :: 'out set
  initial :: 'state

```

### 1.1 FSMs as transition systems

We interpret FSMs as transition systems with a singleton initial state set, based on [1].

```

global-interpretation FSM : transition-system-initial
λ a p. snd a           — execute
λ a p. snd a ∈ succ A (fst a) p — enabled
λ p. p = initial A    — initial
for A
defines path = FSM.path
  and run = FSM.run
  and reachable = FSM.reachable
  and nodes = FSM.nodes
by this

```

```

abbreviation size-FSM M ≡ card (nodes M)

```

```

notation
size-FSM (⟨(|-|)⟩)

```

### 1.2 Language

The following definitions establish basic notions for FSMs similarly to those of nondeterministic finite automata as defined in [1].

In particular, the language of an FSM state are the IO-parts of the paths in the FSM enabled from that state.

```

abbreviation target ≡ FSM.target
abbreviation states ≡ FSM.states

```

**abbreviation**  $trace \equiv FSM.trace$

**abbreviation**  $successors :: ('in, 'out, 'state, 'more) FSM-scheme \Rightarrow 'state \Rightarrow 'state\ set$  **where**  
 $successors \equiv FSM.successors\ TYPE('in)\ TYPE('out)\ TYPE('more)$

**lemma**  $states-alt-def: states\ r\ p = map\ snd\ r$   
**by**  $(induct\ r\ arbitrary: p)\ (auto)$

**lemma**  $trace-alt-def: trace\ r\ p = smap\ snd\ r$   
**by**  $(coinduction\ arbitrary: r\ p)\ (auto)$

**definition**  $language-state :: ('in, 'out, 'state) FSM \Rightarrow 'state$   
 $\Rightarrow ('in \times 'out)\ list\ set\ (\langle LS \rangle)$

**where**  
 $language-state\ M\ q \equiv \{map\ fst\ r \mid r . path\ M\ r\ q\}$

The language of an FSM is the language of its initial state.

**abbreviation**  $L\ M \equiv LS\ M\ (initial\ M)$

**lemma**  $language-state-alt-def : LS\ M\ q = \{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\}$

**proof** –

**have**  $LS\ M\ q \subseteq \{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\}$

**proof**

**fix**  $xr$  **assume**  $xr-assm : xr \in LS\ M\ q$   
**then obtain**  $r$  **where**  $r-def : map\ fst\ r = xr\ path\ M\ r\ q$   
**unfolding**  $language-state-def$  **by**  $auto$   
**then obtain**  $xs\ ys$  **where**  $xr-split : xr = xs \parallel ys$   
 $length\ xs = length\ ys$   
 $length\ xs = length\ xr$

**by**  $(metis\ length-map\ zip-map-fst-snd)$

**then have**  $(xs \parallel ys) \in \{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\}$

**proof** –

**have**  $f1 : xs \parallel ys = map\ fst\ r$   
**by**  $(simp\ add: r-def(1)\ xr-split(1))$   
**then have**  $f2 : path\ M\ ((xs \parallel ys) \parallel take\ (min\ (length\ (xs \parallel ys))\ (length\ (map\ snd\ r)))\ (map\ snd\ r))\ q$

**by**  $(simp\ add: r-def(2))$

**have**  $length\ (xs \parallel ys) = length\ (take\ (min\ (length\ (xs \parallel ys))\ (length\ (map\ snd\ r)))\ (map\ snd\ r))$

**using**  $f1$  **by**  $force$

**then show**  $?thesis$

**using**  $f2$  **by**  $blast$

**qed**

**then show**  $xr \in \{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\}$

**using**  $xr-split$  **by**  $metis$

**qed**

**moreover have**  $\{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\} \subseteq LS\ M\ q$

**proof**

**fix**  $xs$  **assume**  $xs-assm : xs \in \{io \mid io\ tr . path\ M\ (io \parallel tr)\ q \wedge length\ io = length\ tr\}$

**then obtain**  $ys$  **where**  $ys-def : path\ M\ (xs \parallel ys)\ q\ length\ xs = length\ ys$

**by**  $auto$

**then have**  $xs = map\ fst\ (xs \parallel ys)$

**by**  $auto$

**then show**  $xs \in LS\ M\ q$

**using**  $ys-def$  **unfolding**  $language-state-def$  **by**  $blast$

**qed**

**ultimately show**  $?thesis$

**by**  $auto$

**qed**

**lemma**  $language-state[intro]:$

**assumes**  $path\ M\ (w \parallel r)\ q\ length\ w = length\ r$

**shows**  $w \in LS\ M\ q$

**using**  $assms$  **unfolding**  $language-state-def$  **by**  $force$

```

lemma language-state-elim[elim]:
  assumes  $w \in LS\ M\ q$ 
  obtains  $r$ 
  where  $path\ M\ (w\ ||\ r)\ q\ length\ w = length\ r$ 
  using assms unfolding language-state-def by (force iff: split-zip-ex)

lemma language-state-split:
  assumes  $w1\ @\ w2 \in LS\ M\ q$ 
  obtains  $tr1\ tr2$ 
  where  $path\ M\ (w1\ ||\ tr1)\ q\ length\ w1 = length\ tr1$ 
   $path\ M\ (w2\ ||\ tr2)\ (target\ (w1\ ||\ tr1)\ q)\ length\ w2 = length\ tr2$ 
proof –
  obtain  $tr$  where  $tr-def : path\ M\ ((w1\ @\ w2)\ ||\ tr)\ q\ length\ (w1\ @\ w2) = length\ tr$ 
  using assms by blast
  let  $?tr1 = take\ (length\ w1)\ tr$ 
  let  $?tr2 = drop\ (length\ w1)\ tr$ 
  have  $tr-split : ?tr1\ @\ ?tr2 = tr$ 
  by auto
  then show  $?thesis$ 
proof –
  have  $f1 : length\ w1 + length\ w2 = length\ tr$ 
  using  $tr-def(2)$  by auto
  then have  $f2 : length\ w2 = length\ tr - length\ w1$ 
  by presburger
  then have  $length\ w1 = length\ (take\ (length\ w1)\ tr)$ 
  using  $f1$  by (metis (no-types) tr-split diff-add-inverse2 length-append length-drop)
  then show  $?thesis$ 
  using  $f2$  by (metis (no-types) FSM.path-append-elim length-drop that tr-def(1) zip-append1)
qed
qed

lemma language-state-prefix :
  assumes  $w1\ @\ w2 \in LS\ M\ q$ 
shows  $w1 \in LS\ M\ q$ 
  using assms by (meson language-state language-state-split)

lemma succ-nodes :
  fixes  $A :: ('a,'b,'c)\ FSM$ 
  and  $w :: ('a \times 'b)$ 
  assumes  $q2 \in succ\ A\ w\ q1$ 
  and  $q1 \in nodes\ A$ 
shows  $q2 \in nodes\ A$ 
proof –
  obtain  $x\ y$  where  $w = (x,y)$ 
  by (meson surj-pair)
  then have  $q2 \in successors\ A\ q1$ 
  using assms by auto
  then have  $q2 \in reachable\ A\ q1$ 
  by blast
  then have  $q2 \in reachable\ A\ (initial\ A)$ 
  using assms by blast
  then show  $q2 \in nodes\ A$ 
  by blast
qed

lemma states-target-index :
  assumes  $i < length\ p$ 
  shows  $(states\ p\ q1) ! i = target\ (take\ (Suc\ i)\ p)\ q1$ 
  using assms by auto

```

### 1.3 Product machine for language intersection

The following describes the construction of a product machine from two FSMs  $M1$  and  $M2$  such that the language of the product machine is the intersection of the language of  $M1$  and the language of  $M2$ .

**definition** *product* :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  $\Rightarrow$   
('in, 'out, 'state1  $\times$  'state2) FSM **where**  
*product* A B  $\equiv$   
(|  
succ =  $\lambda$  a (p<sub>1</sub>, p<sub>2</sub>). succ A a p<sub>1</sub>  $\times$  succ B a p<sub>2</sub>,  
inputs = inputs A  $\cup$  inputs B,  
outputs = outputs A  $\cup$  outputs B,  
initial = (initial A, initial B)  
|)

**lemma** *product-simps*[simp]:  
succ (product A B) a (p<sub>1</sub>, p<sub>2</sub>) = succ A a p<sub>1</sub>  $\times$  succ B a p<sub>2</sub>  
inputs (product A B) = inputs A  $\cup$  inputs B  
outputs (product A B) = outputs A  $\cup$  outputs B  
initial (product A B) = (initial A, initial B)  
**unfolding** *product-def* **by** *simp+*

**lemma** *product-target*[simp]:  
**assumes** length w = length r<sub>1</sub> length r<sub>1</sub> = length r<sub>2</sub>  
**shows** target (w || r<sub>1</sub> || r<sub>2</sub>) (p<sub>1</sub>, p<sub>2</sub>) = (target (w || r<sub>1</sub>) p<sub>1</sub>, target (w || r<sub>2</sub>) p<sub>2</sub>)  
**using** *assms* **by** (induct arbitrary: p<sub>1</sub> p<sub>2</sub> rule: list-induct3) (auto)

**lemma** *product-path*[iff]:  
**assumes** length w = length r<sub>1</sub> length r<sub>1</sub> = length r<sub>2</sub>  
**shows** path (product A B) (w || r<sub>1</sub> || r<sub>2</sub>) (p<sub>1</sub>, p<sub>2</sub>)  $\longleftrightarrow$  path A (w || r<sub>1</sub>) p<sub>1</sub>  $\wedge$  path B (w || r<sub>2</sub>) p<sub>2</sub>  
**using** *assms* **by** (induct arbitrary: p<sub>1</sub> p<sub>2</sub> rule: list-induct3) (auto)

**lemma** *product-language-state*[simp]: LS (product A B) (q<sub>1</sub>, q<sub>2</sub>) = LS A q<sub>1</sub>  $\cap$  LS B q<sub>2</sub>  
**by** (*fastforce* *iff*: *split-zip*)

**lemma** *product-nodes* :  
nodes (product A B)  $\subseteq$  nodes A  $\times$  nodes B  
**proof**  
**fix** q **assume** q  $\in$  nodes (product A B)  
**then show** q  $\in$  nodes A  $\times$  nodes B  
**proof** (*induction* rule: FSM.nodes.induct)  
**case** (initial p)  
**then show** ?*case* **by** *auto*  
**next**  
**case** (execute p a)  
**then have** fst p  $\in$  nodes A *and* snd p  $\in$  nodes B  
**by** *auto*

**have** snd a  $\in$  (succ A (fst a) (fst p))  $\times$  (succ B (fst a) (snd p))  
**using** *execute* **by** *auto*  
**then have** fst (snd a)  $\in$  succ A (fst a) (fst p)  
snd (snd a)  $\in$  succ B (fst a) (snd p)  
**by** *auto*

**have** fst (snd a)  $\in$  nodes A  
**using**  $\langle$ fst p  $\in$  nodes A $\rangle$   $\langle$ fst (snd a)  $\in$  succ A (fst a) (fst p) $\rangle$   
**by** (*metis* FSM.nodes.simps fst-conv snd-conv)  
**moreover have** snd (snd a)  $\in$  nodes B  
**using**  $\langle$ snd p  $\in$  nodes B $\rangle$   $\langle$ snd (snd a)  $\in$  succ B (fst a) (snd p) $\rangle$   
**by** (*metis* FSM.nodes.simps fst-conv snd-conv)  
**ultimately show** ?*case*  
**by** (*simp* *add*: *mem-Times-iff*)

**qed**  
**qed**

## 1.4 Required properties

FSMs used by the adaptive state counting algorithm are required to satisfy certain properties which are introduced in here. Most notably, the observability property (see function **observable**) implies the uniqueness of certain paths and hence allows for several stronger variations of previous results.

```

fun finite-FSM :: ('in, 'out, 'state) FSM  $\Rightarrow$  bool where
  finite-FSM M = (finite (nodes M)
     $\wedge$  finite (inputs M)
     $\wedge$  finite (outputs M))

fun observable :: ('in, 'out, 'state) FSM  $\Rightarrow$  bool where
  observable M = ( $\forall$  t .  $\forall$  s1 . ((succ M) t s1 = {})
     $\vee$  ( $\exists$  s2 . (succ M) t s1 = {s2}))

fun completely-specified :: ('in, 'out, 'state) FSM  $\Rightarrow$  bool where
  completely-specified M = ( $\forall$  s1  $\in$  nodes M .  $\forall$  x  $\in$  inputs M .
     $\exists$  y  $\in$  outputs M .
     $\exists$  s2 . s2  $\in$  (succ M) (x,y) s1)

fun well-formed :: ('in, 'out, 'state) FSM  $\Rightarrow$  bool where
  well-formed M = (finite-FSM M
     $\wedge$  ( $\forall$  s1 x y . (x  $\notin$  inputs M  $\vee$  y  $\notin$  outputs M)
       $\longrightarrow$  succ M (x,y) s1 = {})
     $\wedge$  inputs M  $\neq$  {}
     $\wedge$  outputs M  $\neq$  {})

abbreviation OFSM M  $\equiv$  well-formed M
   $\wedge$  observable M
   $\wedge$  completely-specified M

lemma OFSM-props[elim!] :
  assumes OFSM M
shows well-formed M
  observable M
  completely-specified M using assms by auto

lemma set-of-succs-finite :
  assumes well-formed M
  and q  $\in$  nodes M
shows finite (succ M io q)
proof (rule ccontr)
  assume infinite (succ M io q)
  moreover have succ M io q  $\subseteq$  nodes M
    using assms by (simp add: subsetI succ-nodes)
  ultimately have infinite (nodes M)
    using infinite-super by blast
  then show False
    using assms by auto
qed

lemma well-formed-path-io-containment :
  assumes well-formed M
  and path M p q
shows set (map fst p)  $\subseteq$  (inputs M  $\times$  outputs M)
using assms proof (induction p arbitrary: q)
case Nil
  then show ?case by auto
next
  case (Cons a p)
  have fst a  $\in$  (inputs M  $\times$  outputs M)
  proof (rule ccontr)
  assume fst a  $\notin$  inputs M  $\times$  outputs M
  then have fst (fst a)  $\notin$  inputs M  $\vee$  snd (fst a)  $\notin$  outputs M
    by (metis SigmaI prod.collapse)
  then have succ M (fst a) q = {}
    using Cons by (metis prod.collapse well-formed.elims(2))
  moreover have (snd a)  $\in$  succ M (fst a) q
    using Cons by auto
  ultimately show False
    by auto
qed

```

```

moreover have set (map fst p)  $\subseteq$  (inputs M  $\times$  outputs M)
using Cons by blast
ultimately show ?case
by auto
qed

```

```

lemma path-input-containment :
  assumes well-formed M
  and path M p q
shows set (map fst (map fst p))  $\subseteq$  inputs M
using assms proof (induction p arbitrary: q rule: rev-induct)
  case Nil
  then show ?case by auto
next
  case (snoc a p)
  have set (map fst (p @ [a]))  $\subseteq$  (inputs M  $\times$  outputs M)
  using well-formed-path-io-containment[OF snoc.prem] by assumption
  then have (fst a)  $\in$  (inputs M  $\times$  outputs M)
  by auto
  then have fst (fst a)  $\in$  inputs M
  by auto
  moreover have set (map fst (map fst p))  $\subseteq$  inputs M
  using snoc.IH[OF snoc.prem(1)]
  using snoc.prem(2) by blast
  ultimately show ?case
  by simp
qed

```

```

lemma path-state-containment :
  assumes path M p q
  and q  $\in$  nodes M
shows set (map snd p)  $\subseteq$  nodes M
using assms by (metis FSM.nodes-states states-alt-def)

```

```

lemma language-state-inputs :
  assumes well-formed M
  and io  $\in$  language-state M q
shows set (map fst io)  $\subseteq$  inputs M
proof -
  obtain tr where path M (io || tr) q length tr = length io
  using assms(2) by auto
  show ?thesis
  by (metis (no-types)
     $\langle \bigwedge thesis. (\bigwedge tr. \llbracket path M (io || tr) q; length tr = length io \rrbracket \implies thesis) \implies thesis \rangle$ 
    assms(1) map-fst-zip path-input-containment)
qed

```

```

lemma set-of-paths-finite :
  assumes well-formed M
  and q1  $\in$  nodes M
shows finite { p . path M p q1  $\wedge$  target p q1 = q2  $\wedge$  length p  $\leq$  k }
proof -
  let ?trs = { tr . set tr  $\subseteq$  nodes M  $\wedge$  length tr  $\leq$  k }
  let ?ios = { io . set io  $\subseteq$  inputs M  $\times$  outputs M  $\wedge$  length io  $\leq$  k }
  let ?iotrs = image ( $\lambda (io, tr) . io || tr$ ) (?ios  $\times$  ?trs)

  let ?paths = { p . path M p q1  $\wedge$  target p q1 = q2  $\wedge$  length p  $\leq$  k }

  have finite (inputs M  $\times$  outputs M)
  using assms by auto
  then have finite ?ios
  using assms by (simp add: finite-lists-length-le)

```

**moreover have**  $\text{finite } ?\text{trs}$   
**using**  $\text{assms}$  **by** ( $\text{simp add: finite-lists-length-le}$ )  
**ultimately have**  $\text{finite } ?\text{iotrs}$   
**by**  $\text{auto}$

**moreover have**  $?paths \subseteq ?\text{iotrs}$

**proof**

**fix**  $p$  **assume**  $p\text{-assm} : p \in \{ p . \text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \text{length } p \leq k \}$   
**then obtain**  $io \ tr$  **where**  $p\text{-split} : p = io \parallel tr \wedge \text{length } io = \text{length } tr$   
**using**  $\text{that}$  **by** ( $\text{metis (no-types) length-map zip-map-fst-snd}$ )  
**then have**  $io \in ?ios$   
**using**  $\text{well-formed-path-io-containment}$   
**proof** –  
**have**  $f1 : \text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \text{length } p \leq k$   
**using**  $p\text{-assm}$  **by**  $\text{force}$   
**then have**  $\text{set } io \subseteq \text{inputs } M \times \text{outputs } M$   
**by** ( $\text{metis (no-types) assms(1) map-fst-zip p-split well-formed-path-io-containment}$ )  
**then show**  $?thesis$   
**using**  $f1$  **by** ( $\text{simp add: p-split}$ )  
**qed**

**moreover have**  $tr \in ?\text{trs}$  **using**  $p\text{-split}$

**proof** –

**have**  $f1 : \text{path } M (io \parallel tr) q1 \wedge \text{target } (io \parallel tr) q1 = q2$   
 $\wedge \text{length } (io \parallel tr) \leq k$  **using**  $\langle p \in \{ p . \text{path } M p q1$   
 $\wedge \text{target } p q1 = q2 \wedge \text{length } p \leq k \} \rangle$   $p\text{-split}$  **by**  $\text{force}$   
**then have**  $f2 : \text{length } tr \leq k$  **by** ( $\text{simp add: p-split}$ )  
**have**  $\text{set } tr \subseteq \text{nodes } M$   
**using**  $f1$  **by** ( $\text{metis (no-types) assms(2) length-map p-split path-state-containment}$   
 $\text{zip-eq zip-map-fst-snd}$ )  
**then show**  $?thesis$   
**using**  $f2$  **by**  $\text{blast}$

**qed**

**ultimately show**  $p \in ?\text{iotrs}$

**using**  $p\text{-split}$  **by**  $\text{auto}$

**qed**

**ultimately show**  $?thesis$

**using**  $\text{Finite-Set.finite-subset}$  **by**  $\text{blast}$

**qed**

**lemma**  $\text{non-distinct-duplicate-indices} :$

**assumes**  $\neg \text{distinct } xs$

**shows**  $\exists i1 \ i2 . i1 \neq i2 \wedge xs ! i1 = xs ! i2 \wedge i1 \leq \text{length } xs \wedge i2 \leq \text{length } xs$

**using**  $\text{assms}$  **by** ( $\text{meson distinct-conv-nth less-imp-le}$ )

**lemma**  $\text{reaching-path-without-repetition} :$

**assumes**  $\text{well-formed } M$

**and**  $q2 \in \text{reachable } M q1$

**and**  $q1 \in \text{nodes } M$

**shows**  $\exists p . \text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \text{distinct } (q1 \# \text{states } p q1)$

**proof** –

**have**  $\text{shorten-nondistinct} : \forall p . (\text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \neg \text{distinct } (q1 \# \text{states } p q1))$   
 $\longrightarrow (\exists p' . \text{path } M p' q1 \wedge \text{target } p' q1 = q2 \wedge \text{length } p' < \text{length } p)$

**proof**

**fix**  $p$

**show**  $(\text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \neg \text{distinct } (q1 \# \text{states } p q1))$

$\longrightarrow (\exists p' . \text{path } M p' q1 \wedge \text{target } p' q1 = q2 \wedge \text{length } p' < \text{length } p)$

**proof**

**assume**  $\text{assm} : \text{path } M p q1 \wedge \text{target } p q1 = q2 \wedge \neg \text{distinct } (q1 \# \text{states } p q1)$

**then show**  $(\exists p' . \text{path } M p' q1 \wedge \text{target } p' q1 = q2 \wedge \text{length } p' < \text{length } p)$

**proof** ( $\text{cases } q1 \in \text{set } (\text{states } p q1)$ )

**case**  $\text{True}$

**have**  $\exists i1 . \text{target } (\text{take } i1 p) q1 = q1 \wedge i1 \leq \text{length } p \wedge i1 > 0$

**proof** ( $\text{rule ccontr}$ )

```

assume  $\neg (\exists i1. \text{target } (\text{take } i1 \text{ } p) \text{ } q1 = q1 \wedge i1 \leq \text{length } p \wedge i1 > 0)$ 
then have  $\neg (\exists i1. (\text{states } p \text{ } q1) ! i1 = q1 \wedge i1 \leq \text{length } (\text{states } p \text{ } q1))$ 
  by (metis True in-set-conv-nth less-eq-Suc-le scan-length scan-nth zero-less-Suc)
then have  $q1 \notin \text{set } (\text{states } p \text{ } q1)$ 
  by (meson in-set-conv-nth less-imp-le)
then show False
  using True by auto
qed
then obtain  $i1$  where  $i1\text{-def} : \text{target } (\text{take } i1 \text{ } p) \text{ } q1 = q1 \wedge i1 \leq \text{length } p \wedge i1 > 0$ 
  by auto

then have  $\text{path } M \text{ } (\text{take } i1 \text{ } p) \text{ } q1$ 
  using assm by (metis FSM.path-append-elim append-take-drop-id)
moreover have  $\text{path } M \text{ } (\text{drop } i1 \text{ } p) \text{ } q1$ 
  using  $i1\text{-def}$  by (metis FSM.path-append-elim append-take-drop-id assm)
ultimately have  $\text{path } M \text{ } (\text{drop } i1 \text{ } p) \text{ } q1 \wedge (\text{target } (\text{drop } i1 \text{ } p) \text{ } q1 = q2)$ 
  using  $i1\text{-def}$  by (metis (no-types) append-take-drop-id assm fold-append o-apply)

moreover have  $\text{length } (\text{drop } i1 \text{ } p) < \text{length } p$ 
  using  $i1\text{-def}$  by auto
ultimately show ?thesis
  using assms by blast

next
case False
then have  $\text{assm}' : \text{path } M \text{ } p \text{ } q1 \wedge \text{target } p \text{ } q1 = q2 \wedge \neg \text{distinct } (\text{states } p \text{ } q1)$ 
  using assm by auto

have  $\exists i1 \ i2. i1 \neq i2 \wedge \text{target } (\text{take } i1 \text{ } p) \text{ } q1 = \text{target } (\text{take } i2 \text{ } p) \text{ } q1$ 
   $\wedge i1 \leq \text{length } p \wedge i2 \leq \text{length } p$ 
proof (rule ccontr)
  assume  $\neg (\exists i1 \ i2. i1 \neq i2 \wedge \text{target } (\text{take } i1 \text{ } p) \text{ } q1 = \text{target } (\text{take } i2 \text{ } p) \text{ } q1$ 
   $\wedge i1 \leq \text{length } p \wedge i2 \leq \text{length } p)$ 
  then have  $\neg (\exists i1 \ i2. i1 \neq i2 \wedge (\text{states } p \text{ } q1) ! i1 = (\text{states } p \text{ } q1) ! i2$ 
   $\wedge i1 \leq \text{length } (\text{states } p \text{ } q1) \wedge i2 \leq \text{length } (\text{states } p \text{ } q1))$ 
  by (metis (no-types, lifting) Suc-leI assm' distinct-conv-nth nat.inject scan-length scan-nth)

  then have  $\text{distinct } (\text{states } p \text{ } q1)$ 
  using non-distinct-duplicate-indices by blast
then show False
  using  $\text{assm}'$  by auto
qed
then obtain  $i1 \ i2$  where  $i\text{-def} : i1 < i2 \wedge \text{target } (\text{take } i1 \text{ } p) \text{ } q1 = \text{target } (\text{take } i2 \text{ } p) \text{ } q1$ 
   $\wedge i1 \leq \text{length } p \wedge i2 \leq \text{length } p$ 
  by (metis nat-neq-iff)

then have  $\text{path } M \text{ } (\text{take } i1 \text{ } p) \text{ } q1$ 
  using assm by (metis FSM.path-append-elim append-take-drop-id)
moreover have  $\text{path } M \text{ } (\text{drop } i2 \text{ } p) \text{ } (\text{target } (\text{take } i2 \text{ } p) \text{ } q1)$ 
  by (metis FSM.path-append-elim append-take-drop-id assm)
ultimately have  $\text{path } M \text{ } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) \text{ } q1$ 
   $\wedge (\text{target } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) \text{ } q1 = q2)$ 
  using  $i\text{-def}$  assm
  by (metis FSM.path-append append-take-drop-id fold-append o-apply)

moreover have  $\text{length } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) < \text{length } p$ 
  using  $i\text{-def}$  by auto

ultimately have  $\text{path } M \text{ } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) \text{ } q1$ 
   $\wedge \text{target } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) \text{ } q1 = q2$ 
   $\wedge \text{length } ((\text{take } i1 \text{ } p) \text{ } @ \text{ } (\text{drop } i2 \text{ } p)) < \text{length } p$ 
  by simp

then show ?thesis
  using assms by blast

```

```

    qed
  qed
qed

obtain p where p-def : path M p q1 ∧ target p q1 = q2
  using assms by auto

let ?paths = {p' . (path M p' q1 ∧ target p' q1 = q2 ∧ length p' ≤ length p)}
let ?minPath = arg-min length (λ io . io ∈ ?paths)

have ?paths ≠ empty
  using p-def by auto
moreover have finite ?paths
  using assms by (simp add: set-of-paths-finite)
ultimately have minPath-def : ?minPath ∈ ?paths ∧ (∀ p' ∈ ?paths . length ?minPath ≤ length p')
  by (meson arg-min-nat-lemma equals0I)

moreover have distinct (q1 # states ?minPath q1)
proof (rule ccontr)
  assume ¬ distinct (q1 # states ?minPath q1)
  then have ∃ p' . path M p' q1 ∧ target p' q1 = q2 ∧ length p' < length ?minPath
    using shorten-nondistinct minPath-def by blast
  then show False
    using minPath-def using arg-min-nat-le dual-order.strict-trans1 by auto
qed

ultimately show ?thesis by auto
qed

```

```

lemma observable-path-unique[simp] :
  assumes io ∈ LS M q
  and observable M
  and path M (io || tr1) q length io = length tr1
  and path M (io || tr2) q length io = length tr2
shows tr1 = tr2
proof (rule ccontr)
  assume tr-assm : tr1 ≠ tr2
  then have state-diff : (states (io || tr1) q) ≠ (states (io || tr2) q)
    by (metis assms(4) assms(6) map-snd-zip states-alt-def)
  show False
  using assms tr-assm proof (induction io arbitrary: q tr1 tr2)
    case Nil
    then show ?case using Nil
      by simp
  next
    case (Cons io-hd io-tl)
    then obtain tr1-hd tr1-tl tr2-hd tr2-tl where tr-split : tr1 = tr1-hd # tr1-tl
      ∧ tr2 = tr2-hd # tr2-tl
      by (metis length-0-conv neq-Nil-conv)

    have p1: path M ([io-hd] || [tr1-hd]) q
      using Cons.prem1 tr-split by auto
    have p2: path M ([io-hd] || [tr2-hd]) q
      using Cons.prem1 tr-split by auto
    have tr-hd-eq : tr1-hd = tr2-hd
      using Cons.prem1 unfolding observable.simps
    proof -
      assume ∀ t s1. succ M t s1 = {} ∨ (∃ s2. succ M t s1 = {s2})
      then show ?thesis
        by (metis (no-types) p1 p2 FSM.path-cons-elim empty-iff prod.sel(1) prod.sel(2) singletonD
            zip-Cons-Cons)
    qed
  qed
qed

```

```

then show ?thesis
  using Cons.IH Cons.prem(3) Cons.prem(4) Cons.prem(5) Cons.prem(6) Cons.prem(7) assms(2)
    tr-split by auto
qed
qed

```

```

lemma observable-path-unique-ex[elim] :
  assumes observable M
  and io ∈ LS M q
obtains tr
where { t . path M (io || t) q ∧ length io = length t } = { tr }
proof -
  obtain tr where tr-def : path M (io || tr) q ∧ length io = length tr
    using assms by auto
  then have { t . path M (io || t) q ∧ length io = length t } ≠ {}
    by blast
  moreover have ∀ t ∈ { t . path M (io || t) q ∧ length io = length t } . t = tr
    using assms tr-def by auto
  ultimately show ?thesis
    using that by auto
qed

```

```

lemma well-formed-product[simp] :
  assumes well-formed M1
  and well-formed M2
shows well-formed (product M2 M1) (is well-formed ?PM)
unfolding well-formed.simps proof
  have finite (nodes M1) finite (nodes M2)
    using assms by auto
  then have finite (nodes M2 × nodes M1)
    by simp

  moreover have nodes ?PM ⊆ nodes M2 × nodes M1
    using product-nodes assms by blast
  ultimately show finite-FSM ?PM
    using infinite-subset assms by auto
next
  have inputs ?PM = inputs M2 ∪ inputs M1
    outputs ?PM = outputs M2 ∪ outputs M1
    by auto
  then show (∀ s1 x y. x ∉ inputs ?PM ∨ y ∉ outputs ?PM ⟶ succ ?PM (x, y) s1 = {})
    ∧ inputs ?PM ≠ {} ∧ outputs ?PM ≠ {}
    using assms by auto
qed

```

## 1.5 States reached by a given IO-sequence

Function `io_targets` collects all states of an FSM reached from a given state by a given IO-sequence. Notably, for any observable FSM, this set contains at most one state.

```

fun io-targets :: ('in, 'out, 'state) FSM ⇒ 'state ⇒ ('in × 'out) list ⇒ 'state set where
  io-targets M q io = { target (io || tr) q | tr . path M (io || tr) q ∧ length io = length tr }

```

```

lemma io-target-implies-L :
  assumes q ∈ io-targets M (initial M) io
  shows io ∈ L M
proof -
  obtain tr where path M (io || tr) (initial M)
    length tr = length io
    target (io || tr) (initial M) = q
    using assms by auto
  then show ?thesis by auto
qed

```

**lemma** *io-target-from-path* :  
**assumes** *path*  $M (w \parallel tr) q$   
**and**  $length\ w = length\ tr$   
**shows**  $target\ (w \parallel tr) q \in io\text{-}targets\ M\ q\ w$   
**using** *assms* **by** *auto*

**lemma** *io-targets-observable-singleton-ex* :  
**assumes** *observable*  $M$   
**and**  $io \in LS\ M\ q1$   
**shows**  $\exists\ q2 . io\text{-}targets\ M\ q1\ io = \{ q2 \}$   
**proof** –  
**obtain**  $tr$  **where**  $tr\text{-}def : \{ t . path\ M\ (io \parallel t) q1 \wedge length\ io = length\ t \} = \{ tr \}$   
**using** *assms* *observable-path-unique-ex* **by** (*metis* (*mono-tags*, *lifting*))  
**then** **have**  $io\text{-}targets\ M\ q1\ io = \{ target\ (io \parallel tr) q1 \}$   
**by** *fastforce*  
**then** **show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *io-targets-observable-singleton-ob* :  
**assumes** *observable*  $M$   
**and**  $io \in LS\ M\ q1$   
**obtains**  $q2$   
**where**  $io\text{-}targets\ M\ q1\ io = \{ q2 \}$   
**proof** –  
**obtain**  $tr$  **where**  $tr\text{-}def : \{ t . path\ M\ (io \parallel t) q1 \wedge length\ io = length\ t \} = \{ tr \}$   
**using** *assms* *observable-path-unique-ex* **by** (*metis* (*mono-tags*, *lifting*))  
**then** **have**  $io\text{-}targets\ M\ q1\ io = \{ target\ (io \parallel tr) q1 \}$   
**by** *fastforce*  
**then** **show** *?thesis* **using** *that* **by** *blast*  
**qed**

**lemma** *io-targets-elim*[*elim*] :  
**assumes**  $p \in io\text{-}targets\ M\ q\ io$   
**obtains**  $tr$   
**where**  $target\ (io \parallel tr) q = p \wedge path\ M\ (io \parallel tr) q \wedge length\ io = length\ tr$   
**using** *assms* **unfolding** *io-targets.simps* **by** *force*

**lemma** *io-targets-reachable* :  
**assumes**  $q2 \in io\text{-}targets\ M\ q1\ io$   
**shows**  $q2 \in reachable\ M\ q1$   
**using** *assms* **unfolding** *io-targets.simps* **by** *blast*

**lemma** *io-targets-nodes* :  
**assumes**  $q2 \in io\text{-}targets\ M\ q1\ io$   
**and**  $q1 \in nodes\ M$   
**shows**  $q2 \in nodes\ M$   
**using** *assms* **by** *auto*

**lemma** *observable-io-targets-split* :  
**assumes** *observable*  $M$   
**and**  $io\text{-}targets\ M\ q1\ (vs @ xs) = \{ q3 \}$   
**and**  $io\text{-}targets\ M\ q1\ vs = \{ q2 \}$   
**shows**  $io\text{-}targets\ M\ q2\ xs = \{ q3 \}$   
**proof** –  
**have**  $vs @ xs \in LS\ M\ q1$   
**using** *assms*(2) **by** *force*  
**then** **obtain**  $trV\ trX$  **where**  $tr\text{-}def :$   
 $path\ M\ (vs \parallel trV) q1\ length\ vs = length\ trV$   
 $path\ M\ (xs \parallel trX) (target\ (vs \parallel trV) q1)\ length\ xs = length\ trX$   
**using** *language-state-split*[*of*  $vs\ xs\ M\ q1$ ] **by** *auto*  
**then** **have**  $tgt\text{-}V : target\ (vs \parallel trV) q1 = q2$   
**using** *assms*(3) **by** *auto*  
**then** **have**  $path\text{-}X : path\ M\ (xs \parallel trX) q2 \wedge length\ xs = length\ trX$

```

using tr-def by auto

have tgt-all : target (vs @ xs || trV @ trX) q1 = q3
proof -
  have f1:  $\exists cs. q3 = \text{target } (vs @ xs || cs) q1$ 
     $\wedge \text{path } M (vs @ xs || cs) q1 \wedge \text{length } (vs @ xs) = \text{length } cs$ 
    using assms(2) by auto
  have length (vs @ xs) = length trV + length trX
    by (simp add: tr-def(2) tr-def(4))
  then have length (vs @ xs) = length (trV @ trX)
    by simp
  then show ?thesis
    using f1 by (metis FSM.path-append  $\langle vs @ xs \in LS M q1 \rangle$  assms(1) observable-path-unique
      tr-def(1) tr-def(2) tr-def(3) zip-append)
qed

then have target ((vs || trV) @ (xs || trX)) q1 = q3
  using tr-def by simp
then have target (xs || trX) q2 = q3
  using tgt-V by auto
then have  $q3 \in \text{io-targets } M q2 xs$ 
  using path-X by auto
then show ?thesis
  by (metis (no-types)  $\langle \text{observable } M \rangle$  path-X insert-absorb io-targets-observable-singleton-ex
    language-state singleton-insert-inj-eq)
qed

```

lemma observable-io-target-unique-target :

```

assumes observable M
and io-targets M q1 io = {q2}
and path M (io || tr) q1
and length io = length tr
shows target (io || tr) q1 = q2
using assms by auto

```

lemma target-in-states :

```

assumes length io = length tr
and length io > 0
shows last (states (io || tr) q) = target (io || tr) q
proof -
  have 0 < length tr
    using assms(1) assms(2) by presburger
  then show ?thesis
    by (simp add: FSM.target-alt-def assms(1) states-alt-def)
qed

```

lemma target-alt-def :

```

assumes length io = length tr
shows length io = 0  $\implies$  target (io || tr) q = q
      length io > 0  $\implies$  target (io || tr) q = last tr
proof -
  show length io = 0  $\implies$  target (io || tr) q = q by simp
  show length io > 0  $\implies$  target (io || tr) q = last tr
    by (metis assms last-ConsR length-greater-0-conv map-snd-zip scan-last states-alt-def)
qed

```

lemma obs-target-is-io-targets :

```

assumes observable M
and path M (io || tr) q
and length io = length tr
shows io-targets M q io = {target (io || tr) q}
  by (metis assms(1) assms(2) assms(3) io-targets-observable-singleton-ex language-state
    observable-io-target-unique-target)

```

```

lemma io-target-target :
  assumes io-targets M q1 io = {q2}
  and path M (io || tr) q1
  and length io = length tr
shows target (io || tr) q1 = q2
proof -
  have target (io || tr) q1 ∈ io-targets M q1 io using assms(2) assms(3) by auto
  then show ?thesis using assms(1) by blast
qed

```

```

lemma index-last-take :
  assumes i < length xs
  shows xs ! i = last (take (Suc i) xs)
  by (simp add: assms take-Suc-conv-app-nth)

```

```

lemma path-last-io-target :
  assumes path M (xs || tr) q
  and length xs = length tr
  and length xs > 0
shows last tr ∈ io-targets M q xs
proof -
  have last tr = target (xs || tr) q
  by (metis assms(2) assms(3) map-snd-zip states-alt-def target-in-states)
  then show ?thesis using assms(1) assms(2) by auto
qed

```

```

lemma path-prefix-io-targets :
  assumes path M (xs || tr) q
  and length xs = length tr
  and length xs > 0
shows last (take (Suc i) tr) ∈ io-targets M q (take (Suc i) xs)
proof -
  have path M (take (Suc i) xs || take (Suc i) tr) q
  by (metis (no-types) FSM.path-append-elim append-take-drop-id assms(1) take-zip)
  then show ?thesis
  using assms(2) assms(3) path-last-io-target by fastforce
qed

```

```

lemma states-index-io-target :
  assumes i < length xs
  and path M (xs || tr) q
  and length xs = length tr
  and length xs > 0
shows (states (xs || tr) q) ! i ∈ io-targets M q (take (Suc i) xs)
proof -
  have (states (xs || tr) q) ! i = last (take (Suc i) (states (xs || tr) q))
  by (metis assms(1) assms(3) map-snd-zip states-alt-def index-last-take)
  then have (states (xs || tr) q) ! i = last (states (take (Suc i) xs || take (Suc i) tr) q)
  by (simp add: take-zip)
  then have (states (xs || tr) q) ! i = last (take (Suc i) tr)
  by (simp add: assms(3) states-alt-def)
  moreover have last (take (Suc i) tr) ∈ io-targets M q (take (Suc i) xs)
  by (meson assms(2) assms(3) assms(4) path-prefix-io-targets)
  ultimately show ?thesis
  by simp
qed

```

```

lemma observable-io-targets-append :
  assumes observable M
  and io-targets M q1 vs = {q2}
  and io-targets M q2 xs = {q3}

```

**shows**  $io\text{-targets } M \ q1 \ (vs@xs) = \{q3\}$   
**proof** –  
**obtain**  $trV$  **where**  $path \ M \ (vs \ || \ trV) \ q1 \ \wedge \ length \ trV = length \ vs \ \wedge \ target \ (vs \ || \ trV) \ q1 = q2$   
**by**  $(metis \ assms(2) \ io\text{-targets-elim} \ singletonI)$   
**moreover obtain**  $trX$  **where**  $path \ M \ (xs \ || \ trX) \ q2 \ \wedge \ length \ trX = length \ xs$   
 $\quad \wedge \ target \ (xs \ || \ trX) \ q2 = q3$   
**by**  $(metis \ assms(3) \ io\text{-targets-elim} \ singletonI)$   
**ultimately have**  $path \ M \ (vs \ @ \ xs \ || \ trV \ @ \ trX) \ q1 \ \wedge \ length \ (trV \ @ \ trX) = length \ (vs \ @ \ xs)$   
 $\quad \wedge \ target \ (vs \ @ \ xs \ || \ trV \ @ \ trX) \ q1 = q3$   
**by**  $auto$   
**then show**  $?thesis$   
**by**  $(metis \ assms(1) \ obs\text{-target-is-io-targets})$   
**qed**

**lemma**  $io\text{-path-states-prefix} :$   
**assumes**  $observable \ M$   
**and**  $path \ M \ (io1 \ || \ tr1) \ q$   
**and**  $length \ tr1 = length \ io1$   
**and**  $path \ M \ (io2 \ || \ tr2) \ q$   
**and**  $length \ tr2 = length \ io2$   
**and**  $prefix \ io1 \ io2$   
**shows**  $tr1 = take \ (length \ tr1) \ tr2$   
**proof** –  
**let**  $?tr1' = take \ (length \ tr1) \ tr2$   
**let**  $?io1' = take \ (length \ tr1) \ io2$   
**have**  $path \ M \ (?io1' \ || \ ?tr1') \ q$   
**by**  $(metis \ FSM.path\text{-append-elim} \ append\text{-take-drop-id} \ assms(4) \ take\text{-zip})$   
**have**  $length \ ?tr1' = length \ ?io1'$   
**using**  $assms \ (5) \ by \ auto$   
  
**have**  $?io1' = io1$   
**proof** –  
**have**  $\forall \ ps \ psa. \ \neg \ prefix \ (ps::('a \ \times \ 'b) \ list) \ psa \ \vee \ length \ ps \leq \ length \ psa$   
**using**  $prefix\text{-length-le} \ by \ blast$   
**then have**  $length \ (take \ (length \ tr1) \ io2) = length \ io1$   
**using**  $assms(3) \ assms(6) \ min.\text{absorb2} \ by \ auto$   
**then show**  $?thesis$   
**by**  $(metis \ assms(6) \ min.\text{cobounded2} \ min\text{-def-raw} \ prefix\text{-length-prefix} \ prefix\text{-order.dual-order}.\text{antisym} \ take\text{-is-prefix})$   
**qed**

**show**  $tr1 = ?tr1'$   
**by**  $(metis \ \langle length \ (take \ (length \ tr1) \ tr2) = length \ (take \ (length \ tr1) \ io2) \rangle$   
 $\ \langle path \ M \ (take \ (length \ tr1) \ io2 \ || \ take \ (length \ tr1) \ tr2) \ q \rangle \ \langle take \ (length \ tr1) \ io2 = io1 \rangle$   
 $assms(1) \ assms(2) \ assms(3) \ language\text{-state} \ observable\text{-path-unique})$   
**qed**

**lemma**  $observable\text{-io-targets-suffix} :$   
**assumes**  $observable \ M$   
**and**  $io\text{-targets } M \ q1 \ vs = \{q2\}$   
**and**  $io\text{-targets } M \ q1 \ (vs@xs) = \{q3\}$   
**shows**  $io\text{-targets } M \ q2 \ xs = \{q3\}$   
**proof** –  
**have**  $prefix \ vs \ (vs@xs)$   
**by**  $auto$   
  
**obtain**  $trV$  **where**  $path \ M \ (vs \ || \ trV) \ q1 \ \wedge \ length \ trV = length \ vs \ \wedge \ target \ (vs \ || \ trV) \ q1 = q2$   
**by**  $(metis \ assms(2) \ io\text{-targets-elim} \ singletonI)$   
**moreover obtain**  $trVX$  **where**  $path \ M \ (vs@xs \ || \ trVX) \ q1$   
 $\quad \wedge \ length \ trVX = length \ (vs@xs) \ \wedge \ target \ (vs@xs \ || \ trVX) \ q1 = q3$   
**by**  $(metis \ assms(3) \ io\text{-targets-elim} \ singletonI)$

ultimately have  $trV = take (length trV) trVX$   
 using  $io\text{-path}\text{-states}\text{-prefix}[OF\ assms(1) \text{ - - - } \langle prefix\ vs\ (vs@xs) \rangle, \text{ of } trV\ q1\ trVX]$  by auto  
 show  $?thesis$   
 by (meson  $assms(1)$   $assms(2)$   $assms(3)$   $observable\text{-}io\text{-}targets\text{-}split$ )  
 qed

lemma  $observable\text{-}io\text{-}target\text{-}is\text{-}singleton[simp]$  :  
 assumes  $observable\ M$   
 and  $p \in io\text{-}targets\ M\ q\ io$   
 shows  $io\text{-}targets\ M\ q\ io = \{p\}$   
 proof -  
 have  $io \in LS\ M\ q$   
 using  $assms(2)$  by auto  
 then obtain  $p'$  where  $io\text{-}targets\ M\ q\ io = \{p'\}$   
 using  $assms(1)$  by (meson  $io\text{-}targets\text{-}observable\text{-}singleton\text{-}ex$ )  
 then show  $?thesis$   
 using  $assms(2)$  by simp  
 qed

lemma  $observable\text{-}path\text{-}prefix$  :  
 assumes  $observable\ M$   
 and  $path\ M\ (io \parallel tr)\ q$   
 and  $length\ io = length\ tr$   
 and  $path\ M\ (ioP \parallel trP)\ q$   
 and  $length\ ioP = length\ trP$   
 and  $prefix\ ioP\ io$   
 shows  $trP = take (length\ ioP)\ tr$   
 proof -  
 have  $ioP\text{-}def : ioP = take (length\ ioP)\ io$   
 using  $assms(6)$  by (metis  $append\text{-}eq\text{-}conv\text{-}conj\ prefixE$ )  
 then have  $take (length\ ioP)\ (io \parallel tr) = take (length\ ioP)\ io \parallel take (length\ ioP)\ tr$   
 using  $take\text{-}zip$  by blast  
 moreover have  $path\ M\ (take (length\ ioP)\ (io \parallel tr))\ q$   
 using  $assms$  by (metis  $FSM.path\text{-}append\text{-}elim\ append\text{-}take\text{-}drop\text{-}id$ )  
 ultimately have  $path\ M\ (take (length\ ioP)\ io \parallel take (length\ ioP)\ tr)\ q$   
 $\wedge length (take (length\ ioP)\ io) = length (take (length\ ioP)\ tr)$   
 using  $assms(3)$  by auto  
 then have  $path\ M\ (ioP \parallel take (length\ ioP)\ tr)\ q \wedge length\ ioP = length (take (length\ ioP)\ tr)$   
 using  $assms(3)$  using  $ioP\text{-}def$  by auto  
 then show  $?thesis$   
 by (meson  $assms(1)$   $assms(4)$   $assms(5)$   $language\text{-}state\ observable\text{-}path\text{-}unique$ )  
 qed

lemma  $io\text{-}targets\text{-}succ$  :  
 assumes  $q2 \in io\text{-}targets\ M\ q1\ [xy]$   
 shows  $q2 \in succ\ M\ xy\ q1$   
 proof -  
 obtain  $tr$  where  $tr\text{-}def : target ([xy] \parallel tr)\ q1 = q2$   
 $path\ M\ ([xy] \parallel tr)\ q1$   
 $length\ [xy] = length\ tr$   
 using  $assms$  by auto  
 have  $length\ tr = Suc\ 0$   
 using  $\langle length\ [xy] = length\ tr \rangle$  by auto  
 then obtain  $q2'$  where  $tr = [q2']$   
 by (metis  $Suc\text{-}length\text{-}conv\ length\text{-}0\text{-}conv$ )  
 then have  $target ([xy] \parallel tr)\ q1 = q2'$   
 by auto  
 then have  $q2' = q2$   
 using  $\langle target ([xy] \parallel tr)\ q1 = q2 \rangle$  by simp  
 then have  $path\ M\ ([xy] \parallel [q2])\ q1$   
 using  $tr\text{-}def(2)$   $\langle tr = [q2] \rangle$  by auto  
 then have  $path\ M\ [(xy, q2)]\ q1$

```

by auto

show ?thesis
proof (cases rule: FSM.path.cases[of M [(xy,q2)] q1])
  case nil
  show ?case
  using ⟨path M [(xy,q2)] q1⟩ by simp
next
  case cons
  show snd (xy, q2) ∈ succ M (fst (xy, q2)) q1 ⟹ path M [] (snd (xy, q2))
  ⟹ q2 ∈ succ M xy q1
  by auto
qed
qed

```

## 1.6 D-reachability

A state of some FSM is d-reached (deterministically reached) by some input sequence if any sequence in the language of the FSM with this input sequence reaches that state. That state is then called d-reachable.

**abbreviation**  $d\text{-reached-by } M p xs q tr ys \equiv$   
 $((length\ xs = length\ ys \wedge length\ xs = length\ tr$   
 $\wedge (path\ M\ ((xs\ ||\ ys)\ ||\ tr)\ p) \wedge target\ ((xs\ ||\ ys)\ ||\ tr)\ p = q)$   
 $\wedge (\forall\ ys2\ tr2 . (length\ xs = length\ ys2 \wedge length\ xs = length\ tr2$   
 $\wedge path\ M\ ((xs\ ||\ ys2)\ ||\ tr2)\ p \longrightarrow target\ ((xs\ ||\ ys2)\ ||\ tr2)\ p = q))$

**fun**  $d\text{-reaches} :: ('in, 'out, 'state)\ FSM \Rightarrow 'state \Rightarrow 'in\ list \Rightarrow 'state \Rightarrow bool$  **where**  
 $d\text{-reaches } M p xs q = (\exists\ tr\ ys . d\text{-reached-by } M p xs q tr ys)$

**fun**  $d\text{-reachable} :: ('in, 'out, 'state)\ FSM \Rightarrow 'state \Rightarrow 'state\ set$  **where**  
 $d\text{-reachable } M p = \{ q . (\exists\ xs . d\text{-reaches } M p xs q) \}$

**lemma**  $d\text{-reaches-unique}[elim]$  :  
**assumes**  $d\text{-reaches } M p xs q1$   
**and**  $d\text{-reaches } M p xs q2$   
**shows**  $q1 = q2$   
**using**  $assms$  **unfolding**  $d\text{-reaches.simps}$  **by**  $blast$

**lemma**  $d\text{-reaches-unique-cases}[simp]$  :  $\{ q . d\text{-reaches } M (initial\ M)\ xs\ q \} = \{ \}$   
 $\vee (\exists\ q2 . \{ q . d\text{-reaches } M (initial\ M)\ xs\ q \} = \{ q2 \})$   
**unfolding**  $d\text{-reaches.simps}$  **by**  $blast$

**lemma**  $d\text{-reaches-unique-obtain}[simp]$  :  
**assumes**  $d\text{-reaches } M (initial\ M)\ xs\ q$   
**shows**  $\{ p . d\text{-reaches } M (initial\ M)\ xs\ p \} = \{ q \}$   
**using**  $assms$  **unfolding**  $d\text{-reaches.simps}$  **by**  $blast$

**lemma**  $d\text{-reaches-io-target}$  :  
**assumes**  $d\text{-reaches } M p xs\ q$   
**and**  $length\ ys = length\ xs$   
**shows**  $io\text{-targets } M p (xs\ ||\ ys) \subseteq \{ q \}$   
**proof**

**fix**  $q'$  **assume**  $q' \in io\text{-targets } M p (xs\ ||\ ys)$   
**then obtain**  $trQ$  **where**  $path\ M\ ((xs\ ||\ ys)\ ||\ trQ)\ p \wedge length\ (xs\ ||\ ys) = length\ trQ$   
**by**  $auto$   
**moreover obtain**  $trD\ ysD$  **where**  $d\text{-reached-by } M p xs\ q\ trD\ ysD$  **using**  $assms(1)$   
**by**  $auto$   
**ultimately have**  $target\ ((xs\ ||\ ys)\ ||\ trQ)\ p = q$   
**by**  $(simp\ add:\ assms(2))$   
**then show**  $q' \in \{ q \}$   
**using**  $\langle d\text{-reached-by } M p xs\ q\ trD\ ysD \rangle \langle q' \in io\text{-targets } M p (xs\ ||\ ys) \rangle assms(2)$  **by**  $auto$   
**qed**

**lemma**  $d\text{-reachable-reachable} : d\text{-reachable } M p \subseteq reachable\ M p$   
**unfolding**  $d\text{-reaches.simps}$   $d\text{-reachable.simps}$  **by**  $blast$

## 1.7 Deterministic state cover

The deterministic state cover of some FSM is a minimal set of input sequences such that every d-reachable state of the FSM is d-reached by a sequence in the set and the set contains the empty sequence (which d-reaches the initial state).

```
fun is-det-state-cover-ass :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('state  $\Rightarrow$  'in list)  $\Rightarrow$  bool where
  is-det-state-cover-ass M f = []  $\wedge$  ( $\forall$  s  $\in$  d-reachable M (initial M) .
    d-reaches M (initial M) (f s) s)
```

```
lemma det-state-cover-ass-dist :
  assumes is-det-state-cover-ass M f
  and s1  $\in$  d-reachable M (initial M)
  and s2  $\in$  d-reachable M (initial M)
  and s1  $\neq$  s2
shows  $\neg$ (d-reaches M (initial M) (f s2) s1)
by (meson assms(1) assms(3) assms(4) d-reaches-unique is-det-state-cover-ass.simps)
```

```
lemma det-state-cover-ass-diff :
  assumes is-det-state-cover-ass M f
  and s1  $\in$  d-reachable M (initial M)
  and s2  $\in$  d-reachable M (initial M)
  and s1  $\neq$  s2
shows f s1  $\neq$  f s2
by (metis assms det-state-cover-ass-dist is-det-state-cover-ass.simps)
```

```
fun is-det-state-cover :: ('in, 'out, 'state) FSM  $\Rightarrow$  'in list set  $\Rightarrow$  bool where
  is-det-state-cover M V = ( $\exists$  f . is-det-state-cover-ass M f
     $\wedge$  V = image f (d-reachable M (initial M)))
```

```
lemma det-state-cover-d-reachable[elim] :
  assumes is-det-state-cover M V
  and v  $\in$  V
obtains q
where d-reaches M (initial M) v q
by (metis (no-types, opaque-lifting) assms(1) assms(2) image-iff is-det-state-cover.simps
  is-det-state-cover-ass.elims(2))
```

```
lemma det-state-cover-card[simp] :
  assumes is-det-state-cover M V
  and finite (nodes M)
shows card (d-reachable M (initial M)) = card V
proof -
  obtain f where f-def : is-det-state-cover-ass M f  $\wedge$  V = image f (d-reachable M (initial M))
  using assms unfolding is-det-state-cover.simps by blast
  then have card-f : card V = card (image f (d-reachable M (initial M)))
  by simp

  have d-reachable M (initial M)  $\subseteq$  nodes M
  unfolding d-reachable.simps d-reaches.simps using d-reachable-reachable by blast
  then have dr-finite : finite (d-reachable M (initial M))
  using assms infinite-super by blast

  then have card-le : card (image f (d-reachable M (initial M)))  $\leq$  card (d-reachable M (initial M))
  using card-image-le by blast

  have card (image f (d-reachable M (initial M))) = card (d-reachable M (initial M))
  by (meson card-image det-state-cover-ass-diff f-def inj-onI)

  then show ?thesis using card-f by auto
qed
```

```
lemma det-state-cover-finite :
```

```

assumes is-det-state-cover  $M V$ 
and finite (nodes  $M$ )
shows finite  $V$ 
proof –
  have d-reachable  $M$  (initial  $M$ )  $\subseteq$  nodes  $M$ 
    by auto
  show finite  $V$  using det-state-cover-card[OF assms]
    by (metis  $\langle$ d-reachable  $M$  (initial  $M$ )  $\subseteq$  nodes  $M$  $\rangle$  assms(1) assms(2) finite-imageI infinite-super
      is-det-state-cover.simps)
qed

```

```

lemma det-state-cover-initial :
  assumes is-det-state-cover  $M V$ 
  shows  $\square \in V$ 
proof –
  have d-reached-by  $M$  (initial  $M$ )  $\square$  (initial  $M$ )  $\square$   $\square$ 
    by (simp add: FSM.nil)
  then have d-reaches  $M$  (initial  $M$ )  $\square$  (initial  $M$ )
    by auto

  have initial  $M \in$  d-reachable  $M$  (initial  $M$ )
    by (metis (no-types)  $\langle$ d-reaches  $M$  (initial  $M$ )  $\square$  (initial  $M$ ) $\rangle$  d-reachable.simps mem-Collect-eq)
  then show ?thesis
    by (metis (no-types, lifting) assms image-iff is-det-state-cover.elims(2)
      is-det-state-cover-ass.simps)
qed

```

```

lemma det-state-cover-empty :
  assumes is-det-state-cover  $M V$ 
  shows  $\square \in V$ 
proof –
  obtain  $f$  where f-def : is-det-state-cover-ass  $M f \wedge V = f'$  d-reachable  $M$  (initial  $M$ )
    using assms by auto
  then have  $f$  (initial  $M$ ) =  $\square$ 
    by auto
  moreover have initial  $M \in$  d-reachable  $M$  (initial  $M$ )
proof –
  have d-reaches  $M$  (initial  $M$ )  $\square$  (initial  $M$ )
    by auto
  then show ?thesis
    by (metis d-reachable.simps mem-Collect-eq)
qed
  moreover have  $f$  (initial  $M$ )  $\in V$ 
    using f-def calculation by blast
  ultimately show ?thesis
    by auto
qed

```

## 1.8 IO reduction

An FSM is a reduction of another, if its language is a subset of the language of the latter FSM.

```

fun io-reduction :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('in, 'out, 'state) FSM
   $\Rightarrow$  bool (infix  $\langle$ \preceq $\rangle$  200)

where
   $M1 \preceq M2 = (LS\ M1\ (initial\ M1) \subseteq LS\ M2\ (initial\ M2))$ 

```

```

lemma language-state-inclusion-of-state-reached-by-same-sequence :
  assumes  $LS\ M1\ q1 \subseteq LS\ M2\ q2$ 
  and observable  $M1$ 
  and observable  $M2$ 
  and io-targets  $M1\ q1\ io = \{ q1t \}$ 
  and io-targets  $M2\ q2\ io = \{ q2t \}$ 
shows  $LS\ M1\ q1t \subseteq LS\ M2\ q2t$ 
proof

```

```

fix  $x$  assume  $x \in LS\ M1\ q1t$ 
obtain  $q1x$  where  $io\text{-targets}\ M1\ q1t\ x = \{q1x\}$ 
  by ( $meson\ \langle x \in LS\ M1\ q1t \rangle\ assms(2)\ io\text{-targets-observable-singleton-ex}$ )
have  $io \in LS\ M1\ q1$ 
  using  $assms(4)$  by  $auto$ 
have  $io@x \in LS\ M1\ q1$ 
  using  $observable\text{-}io\text{-targets-}append[OF\ assms(2)\ \langle io\text{-targets}\ M1\ q1\ io = \{q1t\} \rangle$ 
     $\langle io\text{-targets}\ M1\ q1t\ x = \{q1x\} \rangle$ 
  by ( $metis\ io\text{-targets-elim}\ language\text{-}state\ singletonI$ )
then have  $io@x \in LS\ M2\ q2$ 
  using  $assms(1)$  by  $blast$ 
then obtain  $q2x$  where  $io\text{-targets}\ M2\ q2\ (io@x) = \{q2x\}$ 
  by ( $meson\ assms(3)\ io\text{-targets-observable-singleton-ex}$ )
show  $x \in LS\ M2\ q2t$ 
  using  $observable\text{-}io\text{-targets-split}[OF\ assms(3)\ \langle io\text{-targets}\ M2\ q2\ (io\ @\ x) = \{q2x\} \rangle\ assms(5)]$ 
  by  $auto$ 
qed

```

## 1.9 Language subsets for input sequences

The following definitions describe restrictions of languages to only those IO-sequences that exhibit a certain input sequence or whose input sequence is contained in a given set of input sequences. This allows to define the notion that some FSM is a reduction of another over a given set of input sequences, but not necessarily over the entire language of the latter FSM.

```

fun  $language\text{-}state\text{-}for\text{-}input ::$ 
  ( $'in, 'out, 'state$ )  $FSM \Rightarrow 'state \Rightarrow 'in\ list \Rightarrow ('in \times 'out)\ list\ set$  where
   $language\text{-}state\text{-}for\text{-}input\ M\ q\ xs = \{(xs \parallel ys) \mid ys . (length\ xs = length\ ys \wedge (xs \parallel ys) \in LS\ M\ q)\}$ 

```

```

fun  $language\text{-}state\text{-}for\text{-}inputs ::$ 
  ( $'in, 'out, 'state$ )  $FSM \Rightarrow 'state \Rightarrow 'in\ list\ set \Rightarrow ('in \times 'out)\ list\ set$ 
  ( $\langle (LS_{in} - - -) \rangle [1000,1000,1000]$ ) where
   $language\text{-}state\text{-}for\text{-}inputs\ M\ q\ ISeqs = \{(xs \parallel ys) \mid xs\ ys . (xs \in ISeqs$ 
     $\wedge length\ xs = length\ ys$ 
     $\wedge (xs \parallel ys) \in LS\ M\ q)\}$ 

```

**abbreviation**  $L_{in}\ M\ TS \equiv LS_{in}\ M\ (initial\ M)\ TS$

**abbreviation**  $io\text{-reduction-on}\ M1\ TS\ M2 \equiv (L_{in}\ M1\ TS \subseteq L_{in}\ M2\ TS)$

**notation**

$io\text{-reduction-on}\ (\langle (- \preceq [-] -) \rangle [1000,0,0]\ 61)$

**notation** (*latex output*)

$io\text{-reduction-on}\ (\langle (- \preceq -) \rangle [1000,0,0]\ 61)$

**lemma**  $language\text{-}state\text{-}for\text{-}input\text{-}alt\text{-}def :$

$language\text{-}state\text{-}for\text{-}input\ M\ q\ xs = LS_{in}\ M\ q\ \{xs\}$

**unfolding**  $language\text{-}state\text{-}for\text{-}input.simps\ language\text{-}state\text{-}for\text{-}inputs.simps$  **by**  $blast$

**lemma**  $language\text{-}state\text{-}for\text{-}inputs\text{-}alt\text{-}def :$

$LS_{in}\ M\ q\ ISeqs = \bigcup (image\ (language\text{-}state\text{-}for\text{-}input\ M\ q)\ ISeqs)$

**by**  $auto$

**lemma**  $language\text{-}state\text{-}for\text{-}inputs\text{-}in\text{-}language\text{-}state :$

$LS_{in}\ M\ q\ T \subseteq language\text{-}state\ M\ q$

**unfolding**  $language\text{-}state\text{-}for\text{-}inputs.simps\ language\text{-}state\text{-}def$

**by**  $blast$

**lemma**  $language\text{-}state\text{-}for\text{-}inputs\text{-}map\text{-}fst :$

**assumes**  $io \in language\text{-}state\ M\ q$

**and**  $map\ fst\ io \in T$

**shows**  $io \in LS_{in}\ M\ q\ T$

**proof** –

**let**  $?xs = map\ fst\ io$

**let**  $?ys = map\ snd\ io$

**have**  $?xs \in T \wedge length\ ?xs = length\ ?ys \wedge ?xs \parallel ?ys \in language\text{-}state\ M\ q$

```

  using assms(2,1) by auto
then have  $?xs \parallel ?ys \in LS_{in} M q T$ 
  unfolding language-state-for-inputs.simps by blast
then show ?thesis
  by simp
qed

```

```

lemma language-state-for-inputs-nonempty :
  assumes set xs  $\subseteq$  inputs M
  and completely-specified M
  and  $q \in$  nodes M
shows  $LS_{in} M q \{xs\} \neq \{\}$ 
using assms proof (induction xs arbitrary: q)
  case Nil
  then show ?case by auto
next
  case (Cons x xs)
  then have  $x \in$  inputs M
    by simp
  then obtain  $y q'$  where x-step :  $q' \in succ M (x,y) q$ 
    using Cons(3,4) unfolding completely-specified.simps by blast
  then have path M ( $[(x,y)] \parallel [q']$ )  $q \wedge length [q] = length [(x,y)]$ 
    target ( $[(x,y)] \parallel [q']$ )  $q = q'$ 
    by auto
  then have  $q' \in$  nodes M
    using Cons(4) by (metis FSM.nodes-target)
  then have  $LS_{in} M q' \{xs\} \neq \{\}$ 
    using Cons.premis Cons.IH by auto
  then obtain ys where  $length xs = length ys \wedge (xs \parallel ys) \in LS M q'$ 
    by auto
  then obtain tr where path M ( $(xs \parallel ys) \parallel tr$ )  $q' \wedge length tr = length (xs \parallel ys)$ 
    by auto
  then have path M ( $[(x,y)] @ (xs \parallel ys) \parallel [q'] @ tr$ )  $q$ 
     $\wedge length ([q'] @ tr) = length ([x,y] @ (xs \parallel ys))$ 
    by (simp add: FSM.path.intros(2) x-step)
  then have path M ( $(x\#xs \parallel y\#ys) \parallel [q'] @ tr$ )  $q \wedge length ([q'] @ tr) = length (x\#xs \parallel y\#ys)$ 
    by auto
  then have  $(x\#xs \parallel y\#ys) \in LS M q$ 
    by (metis language-state)
  moreover have  $length (x\#xs) = length (y\#ys)$ 
    by (simp add: <length xs = length ys  $\wedge xs \parallel ys \in LS M q'$ )
  ultimately have  $(x\#xs \parallel y\#ys) \in LS_{in} M q \{x \# xs\}$ 
    unfolding language-state-for-inputs.simps by blast
  then show ?case by blast
qed

```

```

lemma language-state-for-inputs-map-fst-contained :
  assumes  $vs \in LS_{in} M q V$ 
shows map fst vs  $\in V$ 
proof -
  have  $(map fst vs) \parallel (map snd vs) = vs$ 
    by auto
  then have  $(map fst vs) \parallel (map snd vs) \in LS_{in} M q V$ 
    using assms by auto
  then show ?thesis by auto
qed

```

```

lemma language-state-for-inputs-empty :
  assumes  $\square \in V$ 
  shows  $\square \in LS_{in} M q V$ 
proof -
  have  $\square \in$  language-state-for-input M q  $\square$  by auto
  then show ?thesis using language-state-for-inputs-alt-def by (metis UN-I assms)
qed

```

```

lemma language-state-for-input-empty[simp] :

```

*language-state-for-input*  $M q [] = \{\}\}$

by *auto*

lemma *language-state-for-input-take* :

assumes  $io \in \text{language-state-for-input } M q xs$

shows  $\text{take } n \text{ io} \in \text{language-state-for-input } M q (\text{take } n xs)$

proof –

obtain  $ys$  where  $io = xs \parallel ys$  length  $xs = \text{length } ys$   $xs \parallel ys \in \text{language-state } M q$

using *assms* by *auto*

then obtain  $p$  where  $\text{length } p = \text{length } xs$   $\text{path } M ((xs \parallel ys) \parallel p) q$

by *auto*

then have  $\text{path } M (\text{take } n ((xs \parallel ys) \parallel p)) q$

by (*metis* *FSM.path-append-elim* *append-take-drop-id*)

then have  $\text{take } n (xs \parallel ys) \in \text{language-state } M q$

by (*simp* *add: <length p = length xs>* *<length xs = length ys>* *language-state take-zip*)

then have  $(\text{take } n xs) \parallel (\text{take } n ys) \in \text{language-state } M q$

by (*simp* *add: take-zip*)

have  $\text{take } n \text{ io} = (\text{take } n xs) \parallel (\text{take } n ys)$

using  $\langle io = xs \parallel ys \rangle$  *take-zip* by *blast*

moreover have  $\text{length } (\text{take } n xs) = \text{length } (\text{take } n ys)$

by (*simp* *add: <length xs = length ys>*)

ultimately show *?thesis*

using  $\langle (\text{take } n xs) \parallel (\text{take } n ys) \in \text{language-state } M q \rangle$

unfolding *language-state-for-input.simps* by *blast*

qed

lemma *language-state-for-inputs-prefix* :

assumes  $vs@xs \in L_{in} M1 \{vs'@xs'\}$

and  $\text{length } vs = \text{length } vs'$

shows  $vs \in L_{in} M1 \{vs'\}$

proof –

have  $vs@xs \in L M1$

using *assms(1)* by *auto*

then have  $vs \in L M1$

by (*meson* *language-state-prefix*)

then have  $vs \in L_{in} M1 \{\text{map } fst \text{ vs}\}$

by (*meson* *insertI1* *language-state-for-inputs-map-fst*)

moreover have  $vs' = \text{map } fst \text{ vs}$

by (*metis* *append-eq-append-conv* *assms(1)* *assms(2)* *language-state-for-inputs-map-fst-contained* *length-map* *map-append* *singletonD*)

ultimately show *?thesis*

by *blast*

qed

lemma *language-state-for-inputs-union* :

shows  $LS_{in} M q T1 \cup LS_{in} M q T2 = LS_{in} M q (T1 \cup T2)$

unfolding *language-state-for-inputs.simps* by *blast*

lemma *io-reduction-on-subset* :

assumes *io-reduction-on*  $M1 T M2$

and  $T' \subseteq T$

shows *io-reduction-on*  $M1 T' M2$

proof (*rule ccontr*)

assume  $\neg \text{io-reduction-on } M1 T' M2$

then obtain  $xs'$  where  $xs' \in T' \neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\}$

proof –

have  $f1: \forall ps P Pa. (ps::('a \times 'b) \text{ list}) \notin P \vee \neg P \subseteq Pa \vee ps \in Pa$

by *blast*

obtain  $pps :: ('a \times 'b) \text{ list set} \Rightarrow ('a \times 'b) \text{ list set} \Rightarrow ('a \times 'b) \text{ list}$  where

$\forall x0 x1. (\exists v2. v2 \in x1 \wedge v2 \notin x0) = (pps x0 x1 \in x1 \wedge pps x0 x1 \notin x0)$

by *moura*

then have  $f2: \forall P Pa. pps Pa P \in P \wedge pps Pa P \notin Pa \vee P \subseteq Pa$

by (*meson* *subsetI*)

have  $f3: \forall ps f c A. (ps::('a \times 'b) \text{ list}) \notin LS_{in} f (c::'c) A \vee \text{map } fst \text{ ps} \in A$

```

  by (meson language-state-for-inputs-map-fst-contained)
then have  $L_{in} M1 T' \subseteq L_{in} M1 T$ 
  using f2 by (meson assms(2) language-state-for-inputs-in-language-state
    language-state-for-inputs-map-fst set-rev-mp)
then show ?thesis
  using f3 f2 f1 by (meson < $\neg$  io-reduction-on M1 T' M2> assms(1)
    language-state-for-inputs-in-language-state
    language-state-for-inputs-map-fst)
qed
then have  $xs' \in T$ 
  using assms(2) by blast

have  $\neg$  io-reduction-on M1 T M2
proof -
  have f1:  $\forall as. as \notin T' \vee as \in T$ 
    using assms(2) by auto
  obtain pps :: ('a × 'b) list set  $\Rightarrow$  ('a × 'b) list set where
     $\forall x0 x1. (\exists v2. v2 \in x1 \wedge v2 \notin x0) = (pps x0 x1 \in x1 \wedge pps x0 x1 \notin x0)$ 
    by moura
  then have  $\forall P Pa. (\neg P \subseteq Pa \vee (\forall ps. ps \notin P \vee ps \in Pa))$ 
     $\wedge (P \subseteq Pa \vee pps Pa P \in P \wedge pps Pa P \notin Pa)$ 
    by blast
  then show ?thesis
    using f1 by (meson < $\neg$  io-reduction-on M1 T' M2> language-state-for-inputs-in-language-state
      language-state-for-inputs-map-fst language-state-for-inputs-map-fst-contained)
qed

then show False
  using assms(1) by auto
qed

```

## 1.10 Sequences to failures

A sequence to a failure for FSMs  $M1$  and  $M2$  is a sequence such that any proper prefix of it is contained in the languages of both  $M1$  and  $M2$ , while the sequence itself is contained only in the language of  $A$ .

That is, if a sequence to a failure for  $M1$  and  $M2$  exists, then  $M1$  is not a reduction of  $M2$ .

```

fun sequence-to-failure ::
  ('in,'out,'state) FSM  $\Rightarrow$  ('in,'out,'state) FSM  $\Rightarrow$  ('in × 'out) list  $\Rightarrow$  bool where
  sequence-to-failure M1 M2 xs = (
    (butlast xs)  $\in$  (language-state M2 (initial M2)  $\cap$  language-state M1 (initial M1))
     $\wedge$  xs  $\in$  (language-state M1 (initial M1) - language-state M2 (initial M2)))

```

**lemma** *sequence-to-failure-ob* :

```

  assumes  $\neg M1 \preceq M2$ 
  and well-formed M1
  and well-formed M2

```

**obtains** *io*

**where** *sequence-to-failure M1 M2 io*

**proof** -

```

  let ?diff = { io . io  $\in$  language-state M1 (initial M1)  $\wedge$  io  $\notin$  language-state M2 (initial M2) }

```

```

  have ?diff  $\neq$  empty

```

```

  using assms by auto

```

```

  moreover obtain io where io-def[simp] : io = arg-min length ( $\lambda$  io . io  $\in$  ?diff)

```

```

  using assms by auto

```

```

  ultimately have io-diff : io  $\in$  ?diff

```

```

  using assms by (meson all-not-in-conv arg-min-natI)

```

```

  then have io  $\neq$  []

```

```

  using assms io-def language-state by auto

```

```

  then obtain io-init io-last where io-split[simp] : io = io-init @ [io-last]

```

```

  by (metis append-butlast-last-id)

```

```

  have io-init-inclusion : io-init  $\in$  language-state M1 (initial M1)

```

```

     $\wedge$  io-init  $\in$  language-state M2 (initial M2)

```

**proof** (rule ccontr)  
**assume**  $assm : \neg (io-init \in \text{language-state } M1 \text{ (initial } M1))$   
 $\wedge io-init \in \text{language-state } M2 \text{ (initial } M2))$

**have**  $io-init @ [io-last] \in \text{language-state } M1 \text{ (initial } M1)$   
**using**  $io-diff \ io-split$  **by**  $auto$

**then have**  $io-init \in \text{language-state } M1 \text{ (initial } M1)$   
**by** ( $meson \ \text{language-state} \ \text{language-state-split}$ )

**moreover have**  $io-init \notin \text{language-state } M2 \text{ (initial } M2)$   
**using**  $assm \ \text{calculation}$  **by**  $auto$

**ultimately have**  $io-init \in ?diff$   
**by**  $auto$

**moreover have**  $\text{length } io-init < \text{length } io$   
**using**  $io-split$  **by**  $auto$

**ultimately have**  $io \neq \text{arg-min length } (\lambda \ io . \ io \in ?diff)$

**proof** –  
**have**  $\exists ps. ps \in \{\text{language-state } M1 \text{ (initial } M1),$   
 $ps \notin \text{language-state } M2 \text{ (initial } M2)\} \wedge \neg \text{length } io \leq \text{length } ps$

**using**  $\langle io-init \in \{\text{language-state } M1 \text{ (initial } M1), io \notin \text{language-state } M2 \text{ (initial } M2)\} \rangle$   
 $\langle \text{length } io-init < \text{length } io \rangle \ \text{linorder-not-less}$

**by**  $blast$

**then show**  $?thesis$   
**by** ( $meson \ \text{arg-min-nat-le}$ )

**qed**

**then show**  $False$  **using**  $io-def$  **by**  $simp$

**qed**

**then have**  $\text{sequence-to-failure } M1 \ M2 \ io$   
**using**  $io-split \ io-diff$  **by**  $auto$

**then show**  $?thesis$   
**using**  $that$  **by**  $auto$

**qed**

**lemma**  $\text{sequence-to-failure-succ} :$   
**assumes**  $\text{sequence-to-failure } M1 \ M2 \ io$   
**shows**  $\forall q \in \text{io-targets } M2 \text{ (initial } M2) \ (\text{butlast } io) . \ \text{succ } M2 \ (\text{last } io) \ q = \{\}$

**proof**  
**have**  $io \neq []$   
**using**  $assms$  **by**  $auto$

**fix**  $q$  **assume**  $q \in \text{io-targets } M2 \text{ (initial } M2) \ (\text{butlast } io)$

**then obtain**  $tr$  **where**  $q = \text{target } (\text{butlast } io \ || \ tr) \text{ (initial } M2)$   
**and**  $\text{path } M2 \ (\text{butlast } io \ || \ tr) \text{ (initial } M2)$   
**and**  $\text{length } (\text{butlast } io) = \text{length } tr$

**unfolding**  $\text{io-targets.simps}$  **by**  $auto$

**show**  $\text{succ } M2 \ (\text{last } io) \ q = \{\}$

**proof** (rule ccontr)  
**assume**  $\text{succ } M2 \ (\text{last } io) \ q \neq \{\}$

**then obtain**  $q'$  **where**  $q' \in \text{succ } M2 \ (\text{last } io) \ q$   
**by**  $blast$

**then have**  $\text{path } M2 \ [(last \ io, \ q')] \ (\text{target } (\text{butlast } io \ || \ tr) \text{ (initial } M2))$   
**using**  $\langle q = \text{target } (\text{butlast } io \ || \ tr) \text{ (initial } M2) \rangle$  **by**  $auto$

**have**  $\text{path } M2 \ ((\text{butlast } io \ || \ tr) @ [(last \ io, \ q')]) \text{ (initial } M2)$   
**using**  $\langle \text{path } M2 \ (\text{butlast } io \ || \ tr) \text{ (initial } M2) \rangle$   
 $\langle \text{path } M2 \ [(last \ io, \ q')] \ (\text{target } (\text{butlast } io \ || \ tr) \text{ (initial } M2)) \rangle$  **by**  $auto$

**have**  $\text{butlast } io @ [last \ io] = io$   
**by** ( $meson \ \langle io \neq [] \rangle \ \text{append-butlast-last-id}$ )

**have**  $\text{path } M2 \ (io \ || \ (tr @ [q'])) \text{ (initial } M2)$

**proof** –  
**have**  $\text{path } M2 \ ((\text{butlast } io \ || \ tr) @ ([last \ io] \ || \ [q'])) \text{ (initial } M2)$   
**by** ( $simp \ add: \ \text{FSM.path-append} \ \langle \text{path } M2 \ (\text{butlast } io \ || \ tr) \text{ (initial } M2) \rangle$   
 $\langle \text{path } M2 \ [(last \ io, \ q')] \ (\text{target } (\text{butlast } io \ || \ tr) \text{ (initial } M2)) \rangle$ )

**then show**  $?thesis$

```

    by (metis (no-types) ‹butlast io @ [last io] = io›
        ‹length (butlast io) = length tr› zip-append)
qed

have io ∈ L M2
proof -
  have length tr + (0 + Suc 0) = length io
    by (metis ‹butlast io @ [last io] = io› ‹length (butlast io) = length tr›
        length-append list.size(3) list.size(4))
  then show ?thesis
    using ‹path M2 (io || tr @ [q']) (initial M2)› by fastforce
qed
then show False
  using assms by auto
qed
qed

```

```

lemma sequence-to-failure-non-nil :
  assumes sequence-to-failure M1 M2 xs
  shows xs ≠ []
proof
  assume xs = []
  then have xs ∈ L M1 ∩ L M2
    by auto
  then show False using assms by auto
qed

```

```

lemma sequence-to-failure-from-arbitrary-failure :
  assumes vs@xs ∈ L M1 - L M2
  and vs ∈ L M2 ∩ L M1
shows ∃ xs'. prefix xs' xs ∧ sequence-to-failure M1 M2 (vs@xs')
using assms proof (induction xs rule: rev-induct)
  case Nil
  then show ?case by auto
next
  case (snoc x xs)

  have vs @ xs ∈ L M1
    using snoc.premis(1) by (metis Diff-iff append.assoc language-state-prefix)

  show ?case
proof (cases vs@xs ∈ L M2)
  case True
  have butlast (vs@xs@[x]) ∈ L M2 ∩ L M1
    using True ‹vs @ xs ∈ L M1› by (simp add: butlast-append)
  then show ?thesis
    using sequence-to-failure.simps snoc.premis by blast
next
  case False
  then have vs@xs ∈ L M1 - L M2
    using ‹vs @ xs ∈ L M1› by blast
  then obtain xs' where prefix xs' xs sequence-to-failure M1 M2 (vs@xs')
    using snoc.premis(2) snoc.IH by blast
  then show ?thesis
    using prefix-snoc by auto
qed
qed

```

The following lemma shows that if  $M1$  is not a reduction of  $M2$ , then a minimal sequence to a failure exists that is of length at most the number of states in  $M1$  times the number of states in  $M2$ .

```

lemma sequence-to-failure-length :
  assumes well-formed M1
  and well-formed M2
  and observable M1
  and observable M2
  and ¬ M1 ≼ M2

```

shows  $\exists xs . \text{sequence-to-failure } M1\ M2\ xs \wedge \text{length } xs \leq |M2| * |M1|$

**proof** –

**obtain** *seq* **where** *sequence-to-failure* *M1* *M2* *seq*  
**using** *assms* *sequence-to-failure-ob* **by** *blast*  
**then have** *seq*  $\neq []$   
**by** *auto*

**let** *?bls* = *butlast* *seq*  
**have** *?bls*  $\in L\ M1$  *?bls*  $\in L\ M2$   
**using**  $\langle \text{sequence-to-failure } M1\ M2\ \text{seq} \rangle$  **by** *auto*

**then obtain** *tr1b* *tr2b* **where**  
*path* *M1* (*?bls* || *tr1b*) (*initial* *M1*)  
*length* *tr1b* = *length* *?bls*  
*path* *M2* (*?bls* || *tr2b*) (*initial* *M2*)  
*length* *?bls* = *length* *tr2b*  
**by** *fastforce*  
**then have** *length* *tr2b* = *length* *tr1b*  
**by** *auto*

**let** *?PM* = *product* *M2* *M1*  
**have** *well-formed* *?PM*  
**using** *well-formed-product*[*OF* *assms*(1,2)] **by** *assumption*

**have** *path* *?PM* (*?bls* || *tr2b* || *tr1b*) (*initial* *M2*, *initial* *M1*)  
**using** *product-path*[*OF*  $\langle \text{length } ?bls = \text{length } tr2b \rangle$   $\langle \text{length } tr2b = \text{length } tr1b \rangle$ ,  
*of* *M2* *M1* *initial* *M2* *initial* *M1*]  
**using**  $\langle \text{path } M1\ (\text{butlast } seq\ ||\ tr1b)\ (\text{initial } M1) \rangle$   
 $\langle \text{path } M2\ (\text{butlast } seq\ ||\ tr2b)\ (\text{initial } M2) \rangle$   
**by** *blast*

**let** *?q1b* = *target* (*?bls* || *tr1b*) (*initial* *M1*)  
**let** *?q2b* = *target* (*?bls* || *tr2b*) (*initial* *M2*)

**have** *io-targets* *M2* (*initial* *M2*) *?bls* = {*?q2b*}  
**by** (*metis*  $\langle \text{length } (\text{butlast } seq) = \text{length } tr2b \rangle$   $\langle \text{path } M2\ (\text{butlast } seq\ ||\ tr2b)\ (\text{initial } M2) \rangle$ ,  
*assms*(4) *obs-target-is-io-targets*)  
**have** *io-targets* *M1* (*initial* *M1*) *?bls* = {*?q1b*}  
**by** (*metis*  $\langle \text{length } tr1b = \text{length } (\text{butlast } seq) \rangle$   $\langle \text{path } M1\ (\text{butlast } seq\ ||\ tr1b)\ (\text{initial } M1) \rangle$ ,  
*assms*(3) *obs-target-is-io-targets*)

**have** (*?q2b*, *?q1b*)  $\in \text{reachable } (\text{product } M2\ M1)\ (\text{initial } M2, \text{initial } M1)$

**proof** –

**have** *target* (*butlast* *seq* || *tr2b* || *tr1b*) (*initial* *M2*, *initial* *M1*)  
 $\in \text{reachable } (\text{product } M2\ M1)\ (\text{initial } M2, \text{initial } M1)$   
**using**  $\langle \text{path } (\text{product } M2\ M1)\ (\text{butlast } seq\ ||\ tr2b\ ||\ tr1b)\ (\text{initial } M2, \text{initial } M1) \rangle$  **by** *blast*  
**then show** *?thesis*  
**using**  $\langle \text{length } (\text{butlast } seq) = \text{length } tr2b \rangle$   $\langle \text{length } tr2b = \text{length } tr1b \rangle$  **by** *auto*  
**qed**

**have** (*initial* *M2*, *initial* *M1*)  $\in \text{nodes } (\text{product } M2\ M1)$   
**by** (*simp* *add*: *FSM.nodes.initial*)

**obtain** *p* **where** *repFreePath* : *path* (*product* *M2* *M1*) *p* (*initial* *M2*, *initial* *M1*)  $\wedge$   
*target* *p* (*initial* *M2*, *initial* *M1*) =  
(*?q2b*, *?q1b*)  
*distinct* ((*initial* *M2*, *initial* *M1*)  $\#$  *states* *p* (*initial* *M2*, *initial* *M1*))  
**using** *reaching-path-without-repetition*[*OF*  $\langle \text{well-formed } ?PM \rangle$   
 $\langle (\text{?q2b}, \text{?q1b}) \in \text{reachable } (\text{product } M2\ M1)\ (\text{initial } M2, \text{initial } M1) \rangle$ ]

```

    ⟨(initial M2, initial M1) ∈ nodes (product M2 M1)⟩]
  by blast

then have set (states p (initial M2, initial M1)) ⊆ nodes ?PM
  by (simp add: FSM.nodes-states ⟨(initial M2, initial M1) ∈ nodes (product M2 M1)⟩)
moreover have (initial M2, initial M1) ∉ set (states p (initial M2, initial M1))
  using ⟨distinct ((initial M2, initial M1) # states p (initial M2, initial M1))⟩ by auto
ultimately have set (states p (initial M2, initial M1)) ⊆ nodes ?PM - {(initial M2, initial M1)}
  by blast
moreover have finite (nodes ?PM)
  using ⟨well-formed ?PM⟩ by auto
ultimately have card (set (states p (initial M2, initial M1))) < card (nodes ?PM)
  by (metis ⟨(initial M2, initial M1) ∈ nodes (product M2 M1)⟩
    ⟨(initial M2, initial M1) ∉ set (states p (initial M2, initial M1))⟩
    ⟨set (states p (initial M2, initial M1)) ⊆ nodes (product M2 M1)⟩
    psubsetI psubset-card-mono)

moreover have card (set (states p (initial M2, initial M1)))
  = length (states p (initial M2, initial M1))
  using distinct-card repFreePath(2) by fastforce
ultimately have length (states p (initial M2, initial M1)) < |?PM|
  by linarith
then have length p < |?PM|
  by auto

let ?p1 = map (snd ∘ snd) p
let ?p2 = map (fst ∘ snd) p
let ?pIO = map fst p

have p = ?pIO || ?p2 || ?p1
  by (metis map-map zip-map-fst-snd)

have path M2 (?pIO || ?p2) (initial M2)
  path M1 (?pIO || ?p1) (initial M1)
  using product-path[of ?pIO ?p2 ?p1 M2 M1]
  using ⟨p = ?pIO || ?p2 || ?p1⟩ repFreePath(1) by auto

have (?q2b, ?q1b) = (target (?pIO || ?p2 || ?p1) (initial M2, initial M1))
  using ⟨p = ?pIO || ?p2 || ?p1⟩ repFreePath(1) by auto

then have ?q2b = target (?pIO || ?p2) (initial M2)
  ?q1b = target (?pIO || ?p1) (initial M1)
  by auto

have io-targets M2 (initial M2) ?pIO = {?q2b}
  by (metis ⟨path M2 (map fst p || map (fst ∘ snd) p) (initial M2)⟩
    ⟨target (?b1s || tr2b) (initial M2) = target (map fst p || map (fst ∘ snd) p) (initial M2)⟩
    assms(4) length-map obs-target-is-io-targets)

have io-targets M1 (initial M1) ?pIO = {?q1b}
  by (metis ⟨path M1 (map fst p || map (snd ∘ snd) p) (initial M1)⟩
    ⟨target (?b1s || tr1b) (initial M1) = target (map fst p || map (snd ∘ snd) p) (initial M1)⟩
    assms(3) length-map obs-target-is-io-targets)

have seq ∈ L M1 seq ∉ L M2
  using ⟨sequence-to-failure M1 M2 seq⟩ by auto

have io-targets M1 (initial M1) ?b1s = {?q1b}
  by (metis ⟨length tr1b = length (butlast seq)⟩ ⟨path M1 (butlast seq || tr1b) (initial M1)⟩
    assms(3) obs-target-is-io-targets)

```

**obtain**  $q1s$  **where**  $io\text{-targets } M1$  ( $initial\ M1$ )  $seq = \{q1s\}$   
**by** ( $meson \langle seq \in L\ M1 \rangle\ assms(3)\ io\text{-targets-observable-singleton-ob}$ )

**moreover have**  $seq = (butlast\ seq)@[last\ seq]$   
**using**  $\langle seq \neq [] \rangle$  **by**  $auto$

**ultimately have**  $io\text{-targets } M1$  ( $initial\ M1$ )  $((butlast\ seq)@[last\ seq]) = \{q1s\}$   
**by**  $auto$

**have**  $io\text{-targets } M1\ ?q1b\ [last\ seq] = \{q1s\}$   
**using**  $observable\text{-io-targets-suffix}[OF\ assms(3)\ \langle io\text{-targets } M1\ (initial\ M1)\ ?b1s = \{?q1b\} \rangle$   
 $\langle io\text{-targets } M1\ (initial\ M1)\ ((butlast\ seq)@[last\ seq]) = \{q1s\} \rangle$  **by**  $assumption$

**then obtain**  $tr1s$  **where**  $q1s = target\ ([last\ seq] \parallel tr1s)\ ?q1b$   
 $path\ M1\ ([last\ seq] \parallel tr1s)\ ?q1b$   
 $length\ [last\ seq] = length\ tr1s$

**by**  $auto$

**have**  $path\ M1\ ([last\ seq] \parallel [q1s])\ ?q1b$   
**by** ( $metis\ (no\text{-types})\ \langle length\ [last\ seq] = length\ tr1s \rangle$   
 $\langle path\ M1\ ([last\ seq] \parallel tr1s)\ (target\ (butlast\ seq \parallel tr1b)\ (initial\ M1)) \rangle$   
 $\langle q1s = target\ ([last\ seq] \parallel tr1s)\ (target\ (butlast\ seq \parallel tr1b)\ (initial\ M1)) \rangle$   
 $append\ Nil\ append\ butlast\ last\ id\ butlast.simps(2)\ length\ butlast\ length\ greater\ 0\ conv$   
 $not\ Cons\ self2\ target\ alt\ def(2)$ )

**then have**  $q1s \in succ\ M1\ (last\ seq)\ ?q1b$   
**by**  $auto$

**have**  $succ\ M2\ (last\ seq)\ ?q2b = \{\}$

**proof** ( $rule\ ccontr$ )

**assume**  $succ\ M2\ (last\ seq)\ (target\ (butlast\ seq \parallel tr2b)\ (initial\ M2)) \neq \{\}$

**then obtain**  $q2f$  **where**  $q2f \in succ\ M2\ (last\ seq)\ ?q2b$   
**by**  $blast$

**then have**  $target\ ([last\ seq] \parallel [q2f])\ ?q2b = q2f$   
 $path\ M2\ ([last\ seq] \parallel [q2f])\ ?q2b$   
 $length\ [q2f] = length\ [last\ seq]$

**by**  $auto$

**then have**  $q2f \in io\text{-targets } M2\ ?q2b\ [last\ seq]$   
**by** ( $metis\ io\text{-target-from-path}$ )

**then have**  $io\text{-targets } M2\ ?q2b\ [last\ seq] = \{q2f\}$   
**using**  $assms(4)$  **by** ( $meson\ observable\text{-io-target-is-singleton}$ )

**have**  $io\text{-targets } M2\ (initial\ M2)\ (butlast\ seq\ @\ [last\ seq]) = \{q2f\}$   
**using**  $observable\text{-io-targets-append}[OF\ assms(4)\ \langle io\text{-targets } M2\ (initial\ M2)\ ?b1s = \{?q2b\} \rangle$   
 $\langle io\text{-targets } M2\ ?q2b\ [last\ seq] = \{q2f\} \rangle$  **by**  $assumption$

**then have**  $seq \in L\ M2$   
**using**  $\langle seq = butlast\ seq\ @\ [last\ seq] \rangle$  **by**  $auto$

**then show**  $False$   
**using**  $\langle seq \notin L\ M2 \rangle$  **by**  $blast$

**qed**

**have**  $?pIO \in L\ M1\ ?pIO \in L\ M2$   
**using**  $\langle path\ M1\ (?pIO \parallel ?p1)\ (initial\ M1) \rangle\ \langle path\ M2\ (?pIO \parallel ?p2)\ (initial\ M2) \rangle$  **by**  $auto$

**then have**  $butlast\ (?pIO@[last\ seq]) \in L\ M1 \cap L\ M2$   
**by**  $auto$

**have**  $?pIO@[last\ seq] \in L\ M1$   
**using**  $observable\text{-io-targets-append}[OF\ assms(3)\ \langle io\text{-targets } M1\ (initial\ M1)\ ?pIO = \{?q1b\} \rangle$   
 $\langle io\text{-targets } M1\ ?q1b\ [last\ seq] = \{q1s\} \rangle$   
**by** ( $metis\ all\ not\ in\ conv\ insert\ not\ empty\ io\text{-targets-elim\ language-state}$ )

**moreover have**  $?pIO@[last\ seq] \notin L\ M2$

**proof**

**assume**  $?pIO@[last\ seq] \in L\ M2$

**then obtain**  $q2f$  **where**  $io\text{-targets } M2\ (initial\ M2)\ (?pIO@[last\ seq]) = \{q2f\}$   
**by** ( $meson\ assms(4)\ io\text{-targets-observable-singleton-ob}$ )

```

have io-targets  $M2$   $?q2b$  [last seq] = { $q2f$ }
  using observable-io-targets-split [OF assms ( $4$ )
     $\langle$ io-targets  $M2$  (initial  $M2$ ) ( $?pIO@[last\ seq]$ ) = { $q2f$ } $\rangle$ 
     $\langle$ io-targets  $M2$  (initial  $M2$ ) (map fst p) = { $?q2b$ } $\rangle$ ] by assumption

  then have  $q2f \in succ\ M2\ (last\ seq)\ ?q2b$ 
    by (simp add: io-targets-succ)
  then show False
    using  $\langle succ\ M2\ (last\ seq)\ ?q2b = \{\} \rangle$  by auto
qed

ultimately have  $?pIO@[last\ seq] \in L\ M1 - L\ M2$ 
  by auto

have sequence-to-failure  $M1\ M2$  ( $?pIO@[last\ seq]$ )
  using  $\langle butlast\ (?pIO@[last\ seq]) \in L\ M1 \cap L\ M2 \rangle$   $\langle ?pIO@[last\ seq] \in L\ M1 - L\ M2 \rangle$  by auto

have length ( $?pIO@[last\ seq]$ ) = Suc (length  $?pIO$ )
  by auto
then have length ( $?pIO@[last\ seq]$ )  $\leq$   $|?PM|$ 
  using  $\langle length\ p < |?PM| \rangle$  by auto

have card (nodes  $M2 \times$  nodes  $M1$ )  $\leq$   $|M2| * |M1|$ 
  by (simp add: card-cartesian-product)

have finite (nodes  $M2 \times$  nodes  $M1$ )
proof
  show finite (nodes  $M2$ )
    using assms by auto
  show finite (nodes  $M1$ )
    using assms by auto
qed

have  $|?PM| \leq |M2| * |M1|$ 
  by (meson  $\langle card\ (nodes\ M2 \times nodes\ M1) \leq |M2| * |M1| \rangle$   $\langle finite\ (nodes\ M2 \times nodes\ M1) \rangle$ 
    card-mono dual-order.trans product-nodes)

then have length ( $?pIO@[last\ seq]$ )  $\leq$   $|M2| * |M1|$ 
  using  $\langle length\ (?pIO@[last\ seq]) \leq |?PM| \rangle$  by auto

then have sequence-to-failure  $M1\ M2$  ( $?pIO@[last\ seq]$ )  $\wedge$  length ( $?pIO@[last\ seq]$ )  $\leq$   $|M2| * |M1|$ 
  using  $\langle sequence-to-failure\ M1\ M2\ (?pIO@[last\ seq]) \rangle$  by auto
then show ?thesis
  by blast
qed

```

## 1.11 Minimal sequence to failure extending

A minimal sequence to a failure extending some set of IO-sequences is a sequence to a failure of minimal length such that a prefix of that sequence is contained in the set.

```

fun minimal-sequence-to-failure-extending ::
  'in list set  $\Rightarrow$  ('in, 'out, 'state) FSM  $\Rightarrow$  ('in, 'out, 'state) FSM  $\Rightarrow$  ('in  $\times$  'out) list
   $\Rightarrow$  ('in  $\times$  'out) list  $\Rightarrow$  bool where
  minimal-sequence-to-failure-extending  $V\ M1\ M2\ v'\ io =$  (
     $v' \in L_{in}\ M1\ V \wedge sequence-to-failure\ M1\ M2\ (v' @ io)$ 
     $\wedge \neg (\exists io'. \exists w' \in L_{in}\ M1\ V . sequence-to-failure\ M1\ M2\ (w' @ io')$ 
       $\wedge length\ io' < length\ io)$ 
  )

```

```

lemma minimal-sequence-to-failure-extending-det-state-cover-ob :
  assumes well-formed  $M1$ 
  and well-formed  $M2$ 
  and observable  $M2$ 

```

```

and    is-det-state-cover  $M2\ V$ 
and     $\neg M1 \preceq M2$ 
obtains  $vs\ xs$ 
where minimal-sequence-to-failure-extending  $V\ M1\ M2\ vs\ xs$ 
proof –
  — set of all IO-sequences that extend some reaction of M1 to V to a failure
  let  $?exts = \{xs. \exists vs' \in L_{in}\ M1\ V. \text{sequence-to-failure}\ M1\ M2\ (vs'@xs)\}$ 

  — arbitrary sequence to failure
  — must be contained in ?exts as V contains the empty sequence
  obtain  $stf$  where sequence-to-failure  $M1\ M2\ stf$ 
    using assms sequence-to-failure-ob by blast
  then have sequence-to-failure  $M1\ M2\ (\ []\ @\ stf)$ 
    by simp
  moreover have  $\ [] \in L_{in}\ M1\ V$ 
    by (meson assms(4) det-state-cover-initial language-state-for-inputs-empty)
  ultimately have  $stf \in ?exts$ 
    by blast

  — the minimal length sequence of ?exts
  — is a minimal sequence to a failure extending V by construction
  let  $?xsMin = \text{arg-min length } (\lambda xs. xs \in ?exts)$ 
  have  $xsMin\text{-def} : ?xsMin \in ?exts$ 
     $\wedge (\forall xs \in ?exts. \text{length } ?xsMin \leq \text{length } xs)$ 
    by (metis (no-types, lifting) <stf \in ?exts> arg-min-nat-lemma)
  then obtain  $vs$  where  $vs \in L_{in}\ M1\ V$ 
     $\wedge \text{sequence-to-failure}\ M1\ M2\ (vs\ @\ ?xsMin)$ 
    by blast
  moreover have  $\neg(\exists xs. \exists ws \in L_{in}\ M1\ V. \text{sequence-to-failure}\ M1\ M2\ (ws@xs)$ 
     $\wedge \text{length } xs < \text{length } ?xsMin)$ 
    using leD xsMin-def by blast
  ultimately have minimal-sequence-to-failure-extending  $V\ M1\ M2\ vs\ ?xsMin$ 
    by auto
  then show ?thesis
    using that by auto
qed

```

```

lemma mstfe-prefix-input-in-V :
  assumes minimal-sequence-to-failure-extending  $V\ M1\ M2\ vs\ xs$ 
  shows  $(\text{map fst } vs) \in V$ 
proof –
  have  $vs \in L_{in}\ M1\ V$ 
    using assms by auto
  then show ?thesis
    using language-state-for-inputs-map-fst-contained by auto
qed

```

## 1.12 Complete test suite derived from the product machine

The classical result of testing FSMs for language inclusion : Any failure can be observed by a sequence of length at most  $n*m$  where  $n$  is the number of states of the reference model (here FSM  $M2$ ) and  $m$  is an upper bound on the number of states of the SUT (here FSM  $M1$ ).

```

lemma product-suite-soundness :
  assumes well-formed  $M1$ 
  and    well-formed  $M2$ 
  and    observable  $M1$ 
  and    observable  $M2$ 
  and    inputs  $M2 = \text{inputs } M1$ 
  and     $|M1| \leq m$ 
shows  $\neg M1 \preceq M2 \longrightarrow \neg M1 \preceq [\{xs. \text{set } xs \subseteq \text{inputs } M2 \wedge \text{length } xs \leq |M2| * m\}] M2$ 
  (is  $\neg M1 \preceq M2 \longrightarrow \neg M1 \preceq [?TS] M2$ )
proof
  assume  $\neg M1 \preceq M2$ 
  obtain  $stf$  where sequence-to-failure  $M1\ M2\ stf \wedge \text{length } stf \leq |M2| * |M1|$ 
    using sequence-to-failure-length[OF assms(1-4) <\neg M1 \preceq M2>] by blast

```

```

then have sequence-to-failure M1 M2 stf length stf ≤ |M2| * |M1|
  by auto

then have stf ∈ L M1
  by auto
let ?xs = map fst stf
have set ?xs ⊆ inputs M1
  by (meson ⟨stf ∈ L M1⟩ assms(1) language-state-inputs)
then have set ?xs ⊆ inputs M2
  using assms(5) by auto

have length ?xs ≤ |M2| * |M1|
  using ⟨length stf ≤ |M2| * |M1|⟩ by auto
have length ?xs ≤ |M2| * m
proof -
  show ?thesis
  by (metis (no-types) ⟨length (map fst stf) ≤ |M2| * |M1|⟩ ⟨|M1| ≤ m⟩
    dual-order.trans mult.commute mult-le-mono1)
qed

have stf ∈ Lin M1 {?xs}
  by (meson ⟨stf ∈ L M1⟩ insertI1 language-state-for-inputs-map-fst)
have ?xs ∈ ?TS
  using ⟨set ?xs ⊆ inputs M2⟩ ⟨length ?xs ≤ |M2| * m⟩ by blast
have stf ∈ Lin M1 ?TS
  by (metis (no-types, lifting) ⟨map fst stf ∈ {xs. set xs ⊆ inputs M2 ∧ length xs ≤ |M2| * m}⟩
    ⟨stf ∈ L M1⟩ language-state-for-inputs-map-fst)

have stf ∉ L M2
  using ⟨sequence-to-failure M1 M2 stf⟩ by auto
then have stf ∉ Lin M2 ?TS
  by auto

show ¬ M1 ≲[?TS] M2
  using ⟨stf ∈ Lin M1 ?TS⟩ ⟨stf ∉ Lin M2 ?TS⟩ by blast
qed

lemma product-suite-completeness :
  assumes well-formed M1
  and well-formed M2
  and observable M1
  and observable M2
  and inputs M2 = inputs M1
  and |M1| ≤ m
shows M1 ≲ M2 ↔ M1 ≲[?TS] M2
  (is M1 ≲ M2 ↔ M1 ≲[?TS] M2)
proof
  show M1 ≲ M2 ⇒ M1 ≲[?TS] M2 — soundness holds trivially
  unfolding language-state-for-inputs.simps io-reduction.simps by blast
  show M1 ≲[?TS] M2 ⇒ M1 ≲ M2
  using product-suite-soundness[OF assms] by auto
qed

end
theory FSM-Product
imports FSM
begin

```

## 2 Product machines with an additional fail state

We extend the product machine for language intersection presented in theory FSM by an additional state that is reached only by sequences such that any proper prefix of the sequence is in the language intersection, whereas

the full sequence is only contained in the language of the machine B for which we want to check whether it is a reduction of some machine A.

To allow for free choice of the FAIL state, we define the following property that holds iff AB is the product machine of A and B extended with fail state FAIL.

```

fun productF :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  $\Rightarrow$  ('state1  $\times$  'state2)
 $\Rightarrow$  ('in, 'out, 'state1  $\times$  'state2) FSM  $\Rightarrow$  bool where
productF A B FAIL AB = (
  (inputs A = inputs B)
 $\wedge$  (fst FAIL  $\notin$  nodes A)
 $\wedge$  (snd FAIL  $\notin$  nodes B)
 $\wedge$  AB = (
  succ = ( $\lambda$  a (p1,p2) . (if (p1  $\in$  nodes A  $\wedge$  p2  $\in$  nodes B  $\wedge$  (fst a  $\in$  inputs A)
     $\wedge$  (snd a  $\in$  outputs A  $\cup$  outputs B))
    then (if (succ A a p1 = {}  $\wedge$  succ B a p2  $\neq$  {})
      then {FAIL}
      else (succ A a p1  $\times$  succ B a p2))
    else {})),
  inputs = inputs A,
  outputs = outputs A  $\cup$  outputs B,
  initial = (initial A, initial B)
) )

```

```

lemma productF-simps[simp]:
productF A B FAIL AB  $\Longrightarrow$  succ AB a (p1,p2) = (if (p1  $\in$  nodes A  $\wedge$  p2  $\in$  nodes B
   $\wedge$  (fst a  $\in$  inputs A)  $\wedge$  (snd a  $\in$  outputs A  $\cup$  outputs B))
  then (if (succ A a p1 = {}  $\wedge$  succ B a p2  $\neq$  {})
    then {FAIL}
    else (succ A a p1  $\times$  succ B a p2))
  else {})
productF A B FAIL AB  $\Longrightarrow$  inputs AB = inputs A
productF A B FAIL AB  $\Longrightarrow$  outputs AB = outputs A  $\cup$  outputs B
productF A B FAIL AB  $\Longrightarrow$  initial AB = (initial A, initial B)
unfolding productF.simps by simp+

```

```

lemma fail-next-productF :
assumes well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
shows succ PM a FAIL = {}
proof (cases ((fst FAIL)  $\in$  nodes M2  $\wedge$  (snd FAIL)  $\in$  nodes M1))
  case True
  then show ?thesis
  using assms by auto
next
  case False
  then show ?thesis
  using assms by (cases (succ M2 a (fst FAIL) = {}  $\wedge$  (fst a  $\in$  inputs M2)
     $\wedge$  (snd a  $\in$  outputs M2)); auto)
qed

```

```

lemma nodes-productF :
assumes well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
shows nodes PM  $\subseteq$  insert FAIL (nodes M2  $\times$  nodes M1)
proof
fix q assume q-assm : q  $\in$  nodes PM
then show q  $\in$  insert FAIL (nodes M2  $\times$  nodes M1)
using assms proof (cases)
  case initial
  then show ?thesis using assms by auto
next

```

```

case (execute p a)
then obtain  $p1\ p2\ x\ y\ q1\ q2$  where  $p\text{-a-split}[simp] : p = (p1,p2)$ 
 $a = ((x,y),q)$ 
 $q = (q1,q2)$ 

  by (metis eq-snd-iff)
have  $subnodes : p1 \in nodes\ M2 \wedge p2 \in nodes\ M1 \wedge x \in inputs\ M2 \wedge y \in outputs\ M2 \cup outputs\ M1$ 
proof (rule ccontr)
  assume  $\neg (p1 \in nodes\ M2 \wedge p2 \in nodes\ M1 \wedge x \in inputs\ M2 \wedge y \in outputs\ M2 \cup outputs\ M1)$ 
  then have  $succ\ PM\ (x,y)\ (p1,p2) = \{\}$ 
    using assms(3) by auto
  then show False
    using execute by auto
qed

show ?thesis proof (cases (succ M2 (x,y) p1 = \{\} \wedge succ M1 (x,y) p2 \neq \{\}))
  case True
    then have  $q = FAIL$ 
      using subnodes assms(3) execute by auto
    then show ?thesis
      by auto
  next
    case False
    then have  $succ\ PM\ (fst\ a)\ p = succ\ M2\ (x,y)\ p1 \times succ\ M1\ (x,y)\ p2$ 
      using subnodes assms(3) execute by auto
    then have  $q \in (succ\ M2\ (x,y)\ p1 \times succ\ M1\ (x,y)\ p2)$ 
      using execute by blast
    then have  $q\text{-succ} : (q1,q2) \in (succ\ M2\ (x,y)\ p1 \times succ\ M1\ (x,y)\ p2)$ 
      by simp

    have  $q1 \in succ\ M2\ (x,y)\ p1$ 
      using q-succ by simp
    then have  $q1 \in successors\ M2\ p1$ 
      by auto
    then have  $q1 \in reachable\ M2\ p1$ 
      by blast
    then have  $q1 \in reachable\ M2\ (initial\ M2)$ 
      using subnodes by blast
    then have  $nodes1 : q1 \in nodes\ M2$ 
      by blast

    have  $q2 \in succ\ M1\ (x,y)\ p2$ 
      using q-succ by simp
    then have  $q2 \in successors\ M1\ p2$ 
      by auto
    then have  $q2 \in reachable\ M1\ p2$ 
      by blast
    then have  $q2 \in reachable\ M1\ (initial\ M1)$ 
      using subnodes by blast
    then have  $nodes2 : q2 \in nodes\ M1$ 
      by blast

    show ?thesis
      using nodes1 nodes2 by auto
qed
qed
qed

```

```

lemma well-formed-productF[simp] :
  assumes well-formed M1
  and well-formed M2
  and productF M2 M1 FAIL PM
shows well-formed PM
unfolding well-formed.simps proof

```

```

have finite (nodes M1) finite (nodes M2)
  using assms by auto
then have finite (insert FAIL (nodes M2 × nodes M1))
  by simp
moreover have nodes PM ⊆ insert FAIL (nodes M2 × nodes M1)
  using nodes-productF assms by blast
moreover have inputs PM = inputs M2 outputs PM = outputs M2 ∪ outputs M1
  using assms by auto
ultimately show finite-FSM PM
  using infinite-subset assms by auto
next
have inputs PM = inputs M2 outputs PM = outputs M2 ∪ outputs M1
  using assms by auto
then show (∀ s1 x y. x ∉ inputs PM ∨ y ∉ outputs PM → succ PM (x, y) s1 = {})
  ∧ inputs PM ≠ {} ∧ outputs PM ≠ {}
  using assms by auto
qed

```

```

lemma observable-productF[simp] :
  assumes observable M1
  and     observable M2
  and     productF M2 M1 FAIL PM
shows observable PM
  unfolding observable.simps
proof -
  have ∀ t s . succ M1 t (fst s) = {} ∨ (∃ s2. succ M1 t (fst s) = {s2})
    using assms by auto
  moreover have ∀ t s . succ M2 t (snd s) = {} ∨ (∃ s2. succ M2 t (snd s) = {s2})
    using assms by auto
  ultimately have sub-succs : ∀ t s . succ M2 t (fst s) × succ M1 t (snd s) = {}
    ∨ (∃ s2 . succ M2 t (fst s) × succ M1 t (snd s) = {s2})
    by fastforce
  moreover have succ-split : ∀ t s . succ PM t s = {}
    ∨ succ PM t s = {FAIL}
    ∨ succ PM t s = succ M2 t (fst s) × succ M1 t (snd s)
    using assms by auto
  ultimately show ∀ t s. succ PM t s = {} ∨ (∃ s2. succ PM t s = {s2})
    by metis
qed

```

```

lemma no-transition-after-FAIL :
  assumes productF A B FAIL AB
  shows succ AB io FAIL = {}
  using assms by auto

```

```

lemma no-prefix-targets-FAIL :
  assumes productF M2 M1 FAIL PM
  and     path PM p q
  and     k < length p
shows target (take k p) q ≠ FAIL
proof
  assume assm : target (take k p) q = FAIL
  have path PM (take k p @ drop k p) q
    using assms by auto
  then have path PM (drop k p) (target (take k p) q)
    by blast
  then have path-from-FAIL : path PM (drop k p) FAIL
    using assm by auto

  have length (drop k p) ≠ 0
    using assms by auto
  then obtain io q where drop k p = (io, q) # (drop (Suc k) p)
    by (metis Cons-nth-drop-Suc assms(3) prod-cases3)
  then have succ PM io FAIL ≠ {}

```

```

using path-from-FAIL by auto

then show False
  using no-transition-after-FAIL assms by auto
qed

lemma productF-path-inclusion :
  assumes length w = length r1 length r1 = length r2
  and productF A B FAIL AB
  and well-formed A
  and well-formed B
  and path A (w || r1) p1 ∧ path B (w || r2) p2
  and p1 ∈ nodes A
  and p2 ∈ nodes B
shows path (AB) (w || r1 || r2) (p1, p2)
using assms proof (induction w r1 r2 arbitrary: p1 p2 rule: list-induct3)
  case Nil
  then show ?case by auto
next
  case (Cons w ws r1 r1s r2 r2s)
  then have path A ([w] || [r1]) p1 ∧ path B ([w] || [r2]) p2
  by auto
  then have succs : r1 ∈ succ A w p1 ∧ r2 ∈ succ B w p2
  by auto
  then have succ A w p1 ≠ {}
  by force
  then have w-elem : fst w ∈ inputs A ∧ snd w ∈ outputs A
  using Cons by (metis assms(4) prod.collapse well-formed.elims(2))
  then have (r1,r2) ∈ succ AB w (p1,p2)
  using Cons succs by auto
  then have path-head : path AB ([w] || [(r1,r2)]) (p1,p2)
  by auto

  have path A (ws || r1s) r1 ∧ path B (ws || r2s) r2
  using Cons by auto
  moreover have r1 ∈ nodes A ∧ r2 ∈ nodes B
  using succs Cons.prem1 succ-nodes[of r1 A w p1] succ-nodes[of r2 B w p2] by auto
  ultimately have path AB (ws || r1s || r2s) (r1,r2)
  using Cons by blast

  then show ?case
  using path-head by auto
qed

lemma productF-path-forward :
  assumes length w = length r1 length r1 = length r2
  and productF A B FAIL AB
  and well-formed A
  and well-formed B
  and (path A (w || r1) p1 ∧ path B (w || r2) p2)
  ∧ (target (w || r1 || r2) (p1, p2) = FAIL
  ∧ length w > 0
  ∧ path A (butlast (w || r1)) p1
  ∧ path B (butlast (w || r2)) p2
  ∧ succ A (last w) (target (butlast (w || r1)) p1) = {}
  ∧ succ B (last w) (target (butlast (w || r2)) p2) ≠ {})
  and p1 ∈ nodes A
  and p2 ∈ nodes B
shows path (AB) (w || r1 || r2) (p1, p2)
using assms proof (induction w r1 r2 arbitrary: p1 p2 rule: list-induct3)
  case Nil
  then show ?case by auto
next
  case (Cons w ws r1 r1s r2 r2s)
  then show ?case

```

```

proof (cases (path A (w # ws || r1 # r1s) p1 ∧ path B (w # ws || r2 # r2s) p2))
  case True
  then show ?thesis
    using Cons.productF-path-inclusion[of w # ws r1 # r1s r2 # r2s A B FAIL AB p1 p2]
    by auto
next
  case False
  then have fail-prop : target (w # ws || r1 # r1s || r2 # r2s) (p1, p2) = FAIL ∧
    0 < length (w # ws) ∧
    path A (butlast (w # ws || r1 # r1s)) p1 ∧
    path B (butlast (w # ws || r2 # r2s)) p2 ∧
    succ A (last (w # ws)) (target (butlast (w # ws || r1 # r1s)) p1) = {} ∧
    succ B (last (w # ws)) (target (butlast (w # ws || r2 # r2s)) p2) ≠ {}
    using Cons.premis by fastforce

then show ?thesis
proof (cases length ws)
  case 0
  then have empty[simp] : ws = [] r1s = [] r2s = []
    using Cons.hyps by auto
  then have fail-prop-0 : target ([w] || [r1] || [r2]) (p1, p2) = FAIL ∧
    0 < length ([w]) ∧
    path A [] p1 ∧
    path B [] p2 ∧
    succ A w p1 = {} ∧
    succ B w p2 ≠ {}
    using fail-prop by auto
  then have fst w ∈ inputs B ∧ snd w ∈ outputs B
    using Cons.premis by (metis prod.collapse well-formed.elims(2))
  then have inputs-0 : fst w ∈ inputs A ∧ snd w ∈ outputs B
    using Cons.premis by auto

  moreover have fail-elems-0 : (r1, r2) = FAIL
    using fail-prop by auto
  ultimately have succ AB w (p1, p2) = {FAIL}
    using fail-prop-0 Cons.premis by auto

  then have path AB ([w] || [r1] || [r2]) (p1, p2)
    using Cons.premis fail-elems-0 by auto
  then show ?thesis
    by auto
next
  case (Suc nat)

  then have path-r1 : path A ([w] || [r1]) p1
    using fail-prop
    by (metis Cons.hyps(1) FSM.nil FSM.path.intros(2) FSM.path-cons-elim Suc-neq-Zero
      butlast.simps(2) length-0-conv zip-Cons-Cons zip-Nil zip-eq)
  then have path-r1s : path A (butlast (ws || r1s)) r1
    using Suc
    by (metis (no-types, lifting) Cons.hyps(1) FSM.path-cons-elim Suc-neq-Zero butlast.simps(2)
      fail-prop length-0-conv snd-conv zip.simps(1) zip-Cons-Cons zip-eq)

  have path-r2 : path B ([w] || [r2]) p2
    using Suc fail-prop
    by (metis Cons.hyps(1) Cons.hyps(2) FSM.nil FSM.path.intros(2) FSM.path-cons-elim
      Suc-neq-Zero butlast.simps(2) length-0-conv zip-Cons-Cons zip-Nil zip-eq)
  then have path-r2s : path B (butlast (ws || r2s)) r2
    using Suc
    by (metis (no-types, lifting) Cons.hyps(1) Cons.hyps(2) FSM.path-cons-elim Suc-neq-Zero
      butlast.simps(2) fail-prop length-0-conv snd-conv zip.simps(1) zip-Cons-Cons zip-eq)

  have target (ws || r1s || r2s) (r1, r2) = FAIL
    using fail-prop by auto
  moreover have r1 ∈ nodes A

```

```

    using Cons.premis path-r1 by (metis FSM.path-cons-elim snd-conv succ-nodes zip-Cons-Cons)
  moreover have r2 ∈ nodes B
    using Cons.premis path-r2 by (metis FSM.path-cons-elim snd-conv succ-nodes zip-Cons-Cons)
  moreover have succ A (last ws) (target (butlast (ws || r1s)) r1) = {}
    by (metis (no-types, lifting) Cons.hyps(1) Suc Suc-neq-Zero butlast.simps(2) fail-prop
        fold-simps(2) last-ConsR list.size(3) snd-conv zip-Cons-Cons zip-Nil zip-eq)
  moreover have succ B (last ws) (target (butlast (ws || r2s)) r2) ≠ {}
    by (metis (no-types, lifting) Cons.hyps(1) Cons.hyps(2) Suc Suc-neq-Zero butlast.simps(2)
        fail-prop fold-simps(2) last-ConsR list.size(3) snd-conv zip-Cons-Cons zip-Nil zip-eq)

  have path AB (ws || r1s || r2s) (r1, r2)
    using Cons.IH Suc (succ B (last ws) (target (butlast (ws || r2s)) r2) ≠ {})
      assms(3) assms(4) assms(5) calculation(1-4) path-r1s path-r2s zero-less-Suc
    by presburger
  moreover have path AB ([w] || [r1] || [r2]) (p1,p2)
    using path-r1 path-r2 productF-path-inclusion[of [w] [r1] [r2] A B FAIL AB p1 p2]
      Cons.premis
    by auto
  ultimately show ?thesis
    by auto
qed
qed
qed

```

```

lemma butlast-zip-cons : length ws = length r1s ⟹ ws ≠ []
  ⟹ butlast (w # ws || r1 # r1s) = ((w,r1) # (butlast (ws || r1s)))

```

```

proof -
assume a1: length ws = length r1s
assume a2: ws ≠ []
  have length (w # ws) = length r1s + Suc 0
    using a1 by (metis list.size(4))
  then have f3: length (w # ws) = length (r1 # r1s)
    by (metis list.size(4))
  have f4: ws @ w # ws ≠ w # ws
    using a2 by (meson append-self-conv2)
  have length (ws @ w # ws) = length (r1s @ r1 # r1s)
    using a1 by auto
  then have ws @ w # ws || r1s @ r1 # r1s ≠ w # ws || r1 # r1s
    using f4 f3 by (meson zip-eq)
  then show ?thesis
    using a1 by simp
qed

```

```

lemma productF-succ-fail-imp :
  assumes productF A B FAIL AB
  and FAIL ∈ succ AB w (p1,p2)
  and well-formed A
  and well-formed B
shows p1 ∈ nodes A ∧ p2 ∈ nodes B ∧ (fst w ∈ inputs A) ∧ (snd w ∈ outputs A ∪ outputs B)
  ∧ succ AB w (p1,p2) = {FAIL} ∧ succ A w p1 = {} ∧ succ B w p2 ≠ {}
proof -
  have path-head : path AB ([w] || [FAIL]) (p1,p2)
    using assms by auto
  then have succ-nonempty : succ AB w (p1,p2) ≠ {}
    by force
  then have succ-if-1 : p1 ∈ nodes A ∧ p2 ∈ nodes B ∧ (fst w ∈ inputs A)
    ∧ (snd w ∈ outputs A ∪ outputs B)
    using assms by auto
  then have (p1,p2) ≠ FAIL
    using assms by auto

```

```

have succ A w p1  $\subseteq$  nodes A
  using assms succ-if-1 by (simp add: subsetI succ-nodes)
moreover have succ B w p2  $\subseteq$  nodes B
  using assms succ-if-1 by (simp add: subsetI succ-nodes)
ultimately have FAIL  $\notin$  (succ A w p1  $\times$  succ B w p2)
  using assms by auto
then have succ-no-inclusion : succ AB w (p1,p2)  $\neq$  (succ A w p1  $\times$  succ B w p2)
  using assms succ-if-1 by blast
moreover have succ AB w (p1,p2) = {}  $\vee$  succ AB w (p1,p2) = {FAIL}
   $\vee$  succ AB w (p1,p2) = (succ A w p1  $\times$  succ B w p2)
  using assms by simp
ultimately have succ-fail : succ AB w (p1,p2) = {FAIL}
  using succ-nonempty by simp

have succ A w p1 = {}  $\wedge$  succ B w p2  $\neq$  {}
proof (rule ccontr)
  assume  $\neg$  (succ A w p1 = {}  $\wedge$  succ B w p2  $\neq$  {})
  then have succ AB w (p1,p2) = (succ A w p1  $\times$  succ B w p2)
    using assms by auto
  then show False
    using succ-no-inclusion by simp
qed

then show ?thesis
  using succ-if-1 succ-fail by simp
qed

```

```

lemma productF-path-reverse :
  assumes length w = length r1 length r1 = length r2
  and productF A B FAIL AB
  and well-formed A
  and well-formed B
  and path AB (w || r1 || r2) (p1, p2)
  and p1  $\in$  nodes A
  and p2  $\in$  nodes B
shows (path A (w || r1) p1  $\wedge$  path B (w || r2) p2)
   $\vee$  (target (w || r1 || r2) (p1, p2) = FAIL
   $\wedge$  length w > 0
   $\wedge$  path A (butlast (w || r1)) p1
   $\wedge$  path B (butlast (w || r2)) p2
   $\wedge$  succ A (last w) (target (butlast (w || r1)) p1) = {}
   $\wedge$  succ B (last w) (target (butlast (w || r2)) p2)  $\neq$  {})
using assms proof (induction w r1 r2 arbitrary: p1 p2 rule: list-induct3)
  case Nil
  then show ?case by auto
next
  case (Cons w ws r1 r1s r2 r2s)

  have path-head : path AB ([w] || [(r1,r2)]) (p1,p2) using Cons by auto
  then have succ-nonempty : succ AB w (p1,p2)  $\neq$  {} by force
  then have succ-if-1 : p1  $\in$  nodes A  $\wedge$  p2  $\in$  nodes B  $\wedge$  (fst w  $\in$  inputs A)
     $\wedge$  (snd w  $\in$  outputs A  $\cup$  outputs B)
    using Cons by fastforce
  then have (p1,p2)  $\neq$  FAIL
    using Cons by auto

  have path-tail : path AB (ws || r1s || r2s) (r1,r2)
    using path-head Cons by auto

  show ?case
  proof (cases (r1,r2) = FAIL)
  case True
  have r1s = []
  proof (rule ccontr)

```

```

assume  $\neg (r1s = [])$ 
then have  $(\neg (ws = [])) \wedge (\neg (r1s = [])) \wedge (\neg (r2s = []))$ 
  using Cons.hyps by auto
moreover have path AB  $(ws \parallel r1s \parallel r2s)$  FAIL
  using True path-tail by simp
ultimately have path AB  $([hd\ ws] @ tl\ ws \parallel [hd\ r1s] @ tl\ r1s \parallel [hd\ r2s] @ tl\ r2s)$  FAIL
  by simp
then have path AB  $([hd\ ws] \parallel [hd\ r1s] \parallel [hd\ r2s])$  FAIL
  by auto
then have succ AB  $(hd\ ws)$  FAIL  $\neq \{\}$ 
  by auto
then show False using no-transition-after-FAIL
  using Cons.prems by auto
qed
then have tail-nil :  $ws = [] \wedge r1s = [] \wedge r2s = []$ 
  using Cons.hyps by simp

have succ-fail : FAIL  $\in succ\ AB\ w\ (p1, p2)$ 
  using path-head True by auto

then have succs :  $succ\ A\ w\ p1 = \{\} \wedge succ\ B\ w\ p2 \neq \{\}$ 
  using Cons.prems by  $(meson\ productF\ succ\ fail\ imp)$ 

have target  $(w \# ws \parallel r1 \# r1s \parallel r2 \# r2s)\ (p1, p2) = FAIL$ 
  using True tail-nil by simp
moreover have  $0 < length\ (w \# ws)$ 
  by simp
moreover have path A  $(butlast\ (w \# ws \parallel r1 \# r1s))\ p1$ 
  using tail-nil by auto
moreover have path B  $(butlast\ (w \# ws \parallel r2 \# r2s))\ p2$ 
  using tail-nil by auto
moreover have succ A  $(last\ (w \# ws))\ (target\ (butlast\ (w \# ws \parallel r1 \# r1s))\ p1) = \{\}$ 
  using succs tail-nil by simp
moreover have succ B  $(last\ (w \# ws))\ (target\ (butlast\ (w \# ws \parallel r2 \# r2s))\ p2) \neq \{\}$ 
  using succs tail-nil by simp
ultimately show ?thesis
  by simp
next
case False

have  $(r1, r2) \in succ\ AB\ w\ (p1, p2)$ 
  using path-head by auto
then have succ-not-fail :  $succ\ AB\ w\ (p1, p2) \neq \{FAIL\}$ 
  using succ-nonempty False by auto

have  $\neg (succ\ A\ w\ p1 = \{\} \wedge succ\ B\ w\ p2 \neq \{\})$ 
proof  $(rule\ ccontr)$ 
  assume  $\neg \neg (succ\ A\ w\ p1 = \{\} \wedge succ\ B\ w\ p2 \neq \{\})$ 
  then have  $succ\ AB\ w\ (p1, p2) = \{FAIL\}$ 
  using succ-if-1 Cons by auto
  then show False
  using succ-not-fail by simp
qed

then have  $succ\ AB\ w\ (p1, p2) = (succ\ A\ w\ p1 \times succ\ B\ w\ p2)$ 
  using succ-if-1 Cons by auto
then have  $(r1, r2) \in (succ\ A\ w\ p1 \times succ\ B\ w\ p2)$ 
  using Cons by auto
then have succs-next :  $r1 \in succ\ A\ w\ p1 \wedge r2 \in succ\ B\ w\ p2$ 
  by auto
then have nodes-next :  $r1 \in nodes\ A \wedge r2 \in nodes\ B$ 
  using Cons succ-nodes by metis

moreover have path-tail : path AB  $(ws \parallel r1s \parallel r2s)\ (r1, r2)$ 
  using Cons by auto
ultimately have prop-tail :

```

```

path A (ws || r1s) r1 ∧ path B (ws || r2s) r2 ∨
target (ws || r1s || r2s) (r1, r2) = FAIL ∧
0 < length ws ∧
path A (butlast (ws || r1s)) r1 ∧
path B (butlast (ws || r2s)) r2 ∧
succ A (last ws) (target (butlast (ws || r1s)) r1) = {} ∧
succ B (last ws) (target (butlast (ws || r2s)) r2) ≠ {}
using Cons.IH[of r1 r2] Cons.premis by auto

```

```

moreover have path A ([w] || [r1]) p1 ∧ path B ([w] || [r2]) p2
using succs-next by auto
then show ?thesis
proof (cases path A (ws || r1s) r1 ∧ path B (ws || r2s) r2)
case True
moreover have paths-head : path A ([w] || [r1]) p1 ∧ path B ([w] || [r2]) p2
using succs-next by auto
ultimately show ?thesis
by (metis (no-types) FSM.path.simps FSM.path-cons-elim True eq-snd-iff
paths-head zip-Cons-Cons)
next
case False

```

```

then have fail-prop : target (ws || r1s || r2s) (r1, r2) = FAIL ∧
0 < length ws ∧
path A (butlast (ws || r1s)) r1 ∧
path B (butlast (ws || r2s)) r2 ∧
succ A (last ws) (target (butlast (ws || r1s)) r1) = {} ∧
succ B (last ws) (target (butlast (ws || r2s)) r2) ≠ {}
using prop-tail by auto

```

```

then have paths-head : path A ([w] || [r1]) p1 ∧ path B ([w] || [r2]) p2
using succs-next by auto

```

```

have (last (w # ws)) = last ws
using fail-prop by simp
moreover have (target (butlast (w # ws || r1 # r1s)) p1) = (target (butlast (ws || r1s)) r1)
using fail-prop Cons.hyps(1) butlast-zip-cons by auto
moreover have (target (butlast (w # ws || r2 # r2s)) p2) = (target (butlast (ws || r2s)) r2)
using fail-prop Cons.hyps(1) Cons.hyps(2) butlast-zip-cons by auto
ultimately have succ A (last (w # ws)) (target (butlast (w # ws || r1 # r1s)) p1) = {}
∧ succ B (last (w # ws)) (target (butlast (w # ws || r2 # r2s)) p2) ≠ {}
using fail-prop by auto
moreover have path A (butlast (w # ws || r1 # r1s)) p1
using fail-prop paths-head by auto
moreover have path B (butlast (w # ws || r2 # r2s)) p2
using fail-prop paths-head by auto
moreover have target (w # ws || r1 # r1s || r2 # r2s) (p1, p2) = FAIL
using fail-prop paths-head by auto
ultimately show ?thesis
by simp
qed

```

**qed**  
**qed**

```

lemma butlast-zip[simp] :
assumes length xs = length ys
shows butlast (xs || ys) = (butlast xs || butlast ys)
using assms by (metis (no-types, lifting) map-butlast map-fst-zip map-snd-zip zip-map-fst-snd)

```

```

lemma productF-path-reverse-ob :
assumes length w = length r1 length r1 = length r2
and productF A B FAIL AB
and well-formed A

```

```

and    well-formed B
and    path AB (w || r1 || r2) (p1, p2)
and    p1 ∈ nodes A
and    p2 ∈ nodes B
obtains r2'
where path B (w || r2') p2 ∧ length w = length r2'
proof -
  have path-prop : (path A (w || r1) p1 ∧ path B (w || r2) p2)
    ∨ (target (w || r1 || r2) (p1, p2) = FAIL
      ∧ length w > 0
      ∧ path A (butlast (w || r1)) p1
      ∧ path B (butlast (w || r2)) p2
      ∧ succ A (last w) (target (butlast (w || r1)) p1) = {}
      ∧ succ B (last w) (target (butlast (w || r2)) p2) ≠ {})
  using assms productF-path-reverse[of w r1 r2 A B FAIL AB p1 p2] by simp
have ∃ r1'. path B (w || r1') p2 ∧ length w = length r1'
proof (cases path A (w || r1) p1 ∧ path B (w || r2) p2)
  case True
    then show ?thesis
      using assms by auto
  next
    case False
      then have B-prop : length w > 0
        ∧ path B (butlast (w || r2)) p2
        ∧ succ B (last w) (target (butlast (w || r2)) p2) ≠ {}
      using path-prop by auto
      then obtain rx where rx ∈ succ B (last w) (target (butlast (w || r2)) p2)
        by auto

      then have path B ([last w] || [rx]) (target (butlast (w || r2)) p2)
        using B-prop by auto
      then have path B ((butlast (w || r2)) @ ([last w] || [rx])) p2
        using B-prop by auto
      moreover have butlast (w || r2) = (butlast w || butlast r2)
        using assms by simp

      ultimately have path B ((butlast w) @ [last w] || (butlast r2) @ [rx]) p2
        using assms B-prop by auto
      moreover have (butlast w) @ [last w] = w
        using B-prop by simp
      moreover have length ((butlast r2) @ [rx]) = length w
        using assms B-prop by auto
      ultimately show ?thesis
        by auto
    qed
  then obtain r1' where path B (w || r1') p2 ∧ length w = length r1'
    by blast
  then show ?thesis
    using that by blast
qed

```

The following lemma formalizes the property of paths of the product machine as described in the section introduction.

```

lemma productF-path[iff] :
assumes length w = length r1 length r1 = length r2
and    productF A B FAIL AB
and    well-formed A
and    well-formed B
and    p1 ∈ nodes A
and    p2 ∈ nodes B
shows path AB (w || r1 || r2) (p1, p2) ↔ ((path A (w || r1) p1 ∧ path B (w || r2) p2)
  ∨ (target (w || r1 || r2) (p1, p2) = FAIL
    ∧ length w > 0
    ∧ path A (butlast (w || r1)) p1
    ∧ path B (butlast (w || r2)) p2
    ∧ succ A (last w) (target (butlast (w || r1)) p1) = {}
  )

```

$\wedge \text{succ } B (\text{last } w) (\text{target } (\text{butlast } (w \parallel r2)) p2) \neq \{\})$  (is  $?path \longleftrightarrow ?paths$ )

**proof**  
**assume**  $?path$   
**then show**  $?paths$  **using** *assms productF-path-reverse*[of  $w r1 r2 A B FAIL AB p1 p2$ ] **by** *simp*  
**next**  
**assume**  $?paths$   
**then show**  $?path$  **using** *assms productF-path-forward*[of  $w r1 r2 A B FAIL AB p1 p2$ ] **by** *simp*  
**qed**

**lemma** *path-last-succ* :  
**assumes**  $\text{path } A (ws \parallel r1s) p1$   
**and**  $\text{length } r1s = \text{length } ws$   
**and**  $\text{length } ws > 0$   
**shows**  $\text{last } r1s \in \text{succ } A (\text{last } ws) (\text{target } (\text{butlast } (ws \parallel r1s)) p1)$   
**proof** –  
**have**  $\text{path } A (\text{butlast } (ws \parallel r1s)) p1$   
 $\wedge \text{path } A [\text{last } (ws \parallel r1s)] (\text{target } (\text{butlast } (ws \parallel r1s)) p1)$   
**by** (*metis FSM.path-append-elim append-butlast-last-id assms length-greater-0-conv list.size(3) zip-Nil zip-eq*)  
  
**then have**  $\text{snd } (\text{last } (ws \parallel r1s)) \in$   
 $\text{succ } A (\text{fst } (\text{last } (ws \parallel r1s))) (\text{target } (\text{butlast } (ws \parallel r1s)) p1)$   
**by** *auto*  
**moreover have**  $ws \parallel r1s \neq []$   
**using** *assms(3) assms(2)* **by** (*metis length-zip list.size(3) min.idem neq0-conv*)  
**ultimately have**  $\text{last } r1s \in \text{succ } A (\text{last } ws) (\text{target } (\text{butlast } (ws \parallel r1s)) p1)$   
**by** (*simp add: assms(2)*)  
**then show**  $?thesis$   
**by** *auto*  
**qed**

**lemma** *zip-last* :  
**assumes**  $\text{length } r1 > 0$   
**and**  $\text{length } r1 = \text{length } r2$   
**shows**  $\text{last } (r1 \parallel r2) = (\text{last } r1, \text{last } r2)$   
**by** (*metis (no-types) assms(1) assms(2) less-nat-zero-code list.size(3) map-fst-zip zip-Nil zip-last*)

**lemma** *productF-path-reverse-ob-2* :  
**assumes**  $\text{length } w = \text{length } r1 \text{ length } r1 = \text{length } r2$   
**and**  $\text{productF } A B FAIL AB$   
**and**  $\text{well-formed } A$   
**and**  $\text{well-formed } B$   
**and**  $\text{path } AB (w \parallel r1 \parallel r2) (p1, p2)$   
**and**  $p1 \in \text{nodes } A$   
**and**  $p2 \in \text{nodes } B$   
**and**  $w \in \text{language-state } A p1$   
**and**  $\text{observable } A$   
**shows**  $\text{path } A (w \parallel r1) p1 \wedge \text{length } w = \text{length } r1 \text{ path } B (w \parallel r2) p2 \wedge \text{length } w = \text{length } r2$   
 $\text{target } (w \parallel r1) p1 = \text{fst } (\text{target } (w \parallel r1 \parallel r2) (p1, p2))$   
 $\text{target } (w \parallel r2) p2 = \text{snd } (\text{target } (w \parallel r1 \parallel r2) (p1, p2))$   
**proof** –  
  
**have**  $(\text{path } A (w \parallel r1) p1 \wedge \text{path } B (w \parallel r2) p2)$   
 $\vee (\text{target } (w \parallel r1 \parallel r2) (p1, p2) = FAIL$   
 $\wedge \text{length } w > 0$   
 $\wedge \text{path } A (\text{butlast } (w \parallel r1)) p1$   
 $\wedge \text{path } B (\text{butlast } (w \parallel r2)) p2$   
 $\wedge \text{succ } A (\text{last } w) (\text{target } (\text{butlast } (w \parallel r1)) p1) = \{\}$   
 $\wedge \text{succ } B (\text{last } w) (\text{target } (\text{butlast } (w \parallel r2)) p2) \neq \{\})$   
**using** *productF-path*[of  $w r1 r2 A B FAIL AB p1 p2$ ] *assms* **by** *blast*  
  
**moreover have**  $\text{path } A (\text{butlast } (w \parallel r1)) p1$

$\wedge \text{succ } A \text{ (last } w \text{) (target (butlast (w || r1)) p1) = \{\}}$   
 $\wedge \text{length } w > 0 \implies \text{False}$

**proof** –

**assume**  $\text{asm} : \text{path } A \text{ (butlast (w || r1)) p1}$   
 $\wedge \text{succ } A \text{ (last } w \text{) (target (butlast (w || r1)) p1) = \{\}}$   
 $\wedge \text{length } w > 0$

**obtain**  $r1'$  **where**  $r1'\text{-def} : \text{path } A \text{ (w || } r1') \text{ p1} \wedge \text{length } r1' = \text{length } w$   
**using**  $\text{assms}(9)$  **by** *auto*

**then have**  $\text{path } A \text{ (butlast (w || } r1') \text{) p1} \wedge \text{length (butlast } r1') = \text{length (butlast } w)$   
**by** (*metis*  $\text{FSM.path-append-elim append-butlast-last-id butlast.simps}(1) \text{length-butlast}$ )

**moreover have**  $\text{path } A \text{ (butlast (w || } r1) \text{) p1} \wedge \text{length (butlast } r1) = \text{length (butlast } w)$   
**using**  $\text{asm assms}(1)$  **by** *auto*

**ultimately have**  $\text{butlast } r1 = \text{butlast } r1'$   
**by** (*metis*  $\text{assms}(1) \text{assms}(10) \text{butlast-zip language-state observable-path-unique } r1'\text{-def}$ )

**then have**  $\text{butlast (w || } r1) = \text{butlast (w || } r1')$   
**using**  $\text{assms}(1) r1'\text{-def}$  **by** *simp*

**moreover have**  $\text{succ } A \text{ (last } w \text{) (target (butlast (w || } r1') \text{) p1) \neq \{\}}$   
**by** (*metis* ( $\text{no-types}$ )  $\text{asm empty-iff path-last-succ } r1'\text{-def}$ )

**ultimately show** *False*  
**using**  $\text{asm}$  **by** *auto*

**qed**

**ultimately have**  $\text{paths} : (\text{path } A \text{ (w || } r1) \text{ p1} \wedge \text{path } B \text{ (w || } r2) \text{ p2})$   
**by** *auto*

**show**  $\text{path } A \text{ (w || } r1) \text{ p1} \wedge \text{length } w = \text{length } r1$   
**using**  $\text{assms}(1) \text{paths}$  **by** *simp*

**show**  $\text{path } B \text{ (w || } r2) \text{ p2} \wedge \text{length } w = \text{length } r2$   
**using**  $\text{assms}(1) \text{assms}(2) \text{paths}$  **by** *simp*

**have**  $\text{length } w = 0 \implies \text{target (w || } r1 \text{ || } r2) (p1, p2) = (p1, p2)$   
**by** *simp*

**moreover have**  $\text{length } w > 0 \implies \text{target (w || } r1 \text{ || } r2) (p1, p2) = \text{last (} r1 \text{ || } r2)$

**proof** –

**assume**  $\text{length } w > 0$

**moreover have**  $\text{length } w = \text{length (} r1 \text{ || } r2)$   
**using**  $\text{assms}(1) \text{assms}(2)$  **by** *simp*

**ultimately show** *?thesis*  
**using**  $\text{target-alt-def}(2)[\text{of } w \text{ } r1 \text{ || } r2 (p1, p2)]$  **by** *simp*

**qed**

**ultimately have**  $\text{target (w || } r1) \text{ p1} = \text{fst (target (w || } r1 \text{ || } r2) (p1, p2))$   
 $\wedge \text{target (w || } r2) \text{ p2} = \text{snd (target (w || } r1 \text{ || } r2) (p1, p2))$

**proof** (*cases*  $\text{length } w$ )

**case**  $0$

**then show** *?thesis* **by** *simp*

**next**

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

**then have**  $\text{length } w > 0$  **by** *simp*

**have**  $\text{target (w || } r1 \text{ || } r2) (p1, p2) = \text{last (} r1 \text{ || } r2)$

**proof** –

**have**  $\text{length } w = \text{length (} r1 \text{ || } r2)$   
**using**  $\text{assms}(1) \text{assms}(2)$  **by** *simp*

**then show** *?thesis*  
**using**  $\langle \text{length } w > 0 \rangle \text{target-alt-def}(2)[\text{of } w \text{ } r1 \text{ || } r2 (p1, p2)]$  **by** *simp*

**qed**

**moreover have**  $\text{target (w || } r1) \text{ p1} = \text{last } r1$   
**using**  $\langle \text{length } w > 0 \rangle \text{target-alt-def}(2)[\text{of } w \text{ } r1 \text{ p1}] \text{assms}(1)$  **by** *simp*

**moreover have**  $\text{target (w || } r2) \text{ p2} = \text{last } r2$   
**using**  $\langle \text{length } w > 0 \rangle \text{target-alt-def}(2)[\text{of } w \text{ } r2 \text{ p2}] \text{assms}(1) \text{assms}(2)$  **by** *simp*

**moreover have**  $\text{last (} r1 \text{ || } r2) = (\text{last } r1, \text{last } r2)$   
**using**  $\langle \text{length } w > 0 \rangle \text{assms}(1) \text{assms}(2) \text{zip-last}[\text{of } r1 \text{ } r2]$  **by** *simp*

**ultimately show** *?thesis*  
**by** *simp*

```

qed

then show target (w || r1) p1 = fst (target (w || r1 || r2) (p1,p2))
      target (w || r2) p2 = snd (target (w || r1 || r2) (p1,p2))
  by simp+
qed

```

```

lemma productF-path-unzip :
  assumes productF A B FAIL AB
  and path AB (w || tr) q
  and length tr = length w
shows path AB (w || (map fst tr || map snd tr)) q
proof -
  have map fst tr || map snd tr = tr
    by auto
  then show ?thesis
    using assms by auto
qed

```

```

lemma productF-path-io-targets :
  assumes productF A B FAIL AB
  and io-targets AB (qA,qB) w = {(pA,pB)}
  and w ∈ language-state A qA
  and w ∈ language-state B qB
  and observable A
  and observable B
  and well-formed A
  and well-formed B
  and qA ∈ nodes A
  and qB ∈ nodes B
shows pA ∈ io-targets A qA w pB ∈ io-targets B qB w
proof -
  obtain tr where tr-def : target (w || tr) (qA,qB) = (pA,pB)
    ∧ path AB (w || tr) (qA,qB)
    ∧ length w = length tr using assms(2)

  by blast
  have path-A : path A (w || map fst tr) qA ∧ length w = length (map fst tr)
    using productF-path-reverse-ob-2[of w map fst tr map snd tr A B FAIL AB qA qB]
    assms tr-def by auto
  have path-B : path B (w || map snd tr) qB ∧ length w = length (map snd tr)
    using productF-path-reverse-ob-2[of w map fst tr map snd tr A B FAIL AB qA qB]
    assms tr-def by auto

  have targets : target (w || map fst tr) qA = pA ∧ target (w || map snd tr) qB = pB
  proof (cases tr)
    case Nil
    then have qA = pA ∧ qB = pB
      using tr-def by auto
    then show ?thesis
      by (simp add: local.Nil)
  next
    case (Cons a list)
    then have last tr = (pA,pB)
      using tr-def by (simp add: tr-def FSM.target-alt-def states-alt-def)

    moreover have target (w || map fst tr) qA = last (map fst tr)
      using Cons by (simp add: FSM.target-alt-def states-alt-def tr-def)
    moreover have last (map fst tr) = fst (last tr)
      using last-map Cons by blast
  qed

```

```

moreover have target (w || map snd tr) qB = last (map snd tr)
  using Cons by (simp add: FSM.target-alt-def states-alt-def tr-def)
moreover have last (map snd tr) = snd (last tr)
  using last-map Cons by blast

ultimately show ?thesis
  by simp
qed

show pA ∈ io-targets A qA w
  using path-A targets by auto
show pB ∈ io-targets B qB w
  using path-B targets by auto
qed

lemma productF-path-io-targets-reverse :
assumes productF A B FAIL AB
and pA ∈ io-targets A qA w
and pB ∈ io-targets B qB w
and w ∈ language-state A qA
and w ∈ language-state B qB
and observable A
and observable B
and well-formed A
and well-formed B
and qA ∈ nodes A
and qB ∈ nodes B
shows io-targets AB (qA,qB) w = {(pA,pB)}
proof -
obtain trA where path A (w || trA) qA
  length w = length trA
  target (w || trA) qA = pA
  using assms(2) by auto
obtain trB where path B (w || trB) qB
  length trA = length trB
  target (w || trB) qB = pB
  using ⟨length w = length trA⟩ assms(3) by auto

have path AB (w || trA || trB) (qA,qB)
  length (trA || trB) = length w
  using productF-path-inclusion
  [OF ⟨length w = length trA⟩ ⟨length trA = length trB⟩ assms(1) assms(8,9) - assms(10,11)]
  by (simp add: ⟨length trA = length trB⟩ ⟨length w = length trA⟩ ⟨path A (w || trA) qA⟩
    ⟨path B (w || trB) qB⟩)+

have target (w || trA || trB) (qA,qB) = (pA,pB)
  by (simp add: ⟨length trA = length trB⟩ ⟨length w = length trA⟩ ⟨target (w || trA) qA = pA⟩
    ⟨target (w || trB) qB = pB⟩)

have (pA,pB) ∈ io-targets AB (qA,qB) w
  by (metis ⟨length (trA || trB) = length w⟩ ⟨path AB (w || trA || trB) (qA, qB)⟩
    ⟨target (w || trA || trB) (qA, qB) = (pA, pB)⟩ io-target-from-path)

have observable AB
  by (metis (no-types) assms(1) assms(6) assms(7) observable-productF)

show ?thesis
  by (meson ⟨(pA, pB) ∈ io-targets AB (qA, qB) w⟩ ⟨observable AB⟩
    observable-io-target-is-singleton)
qed

```

## 2.1 Sequences to failure in the product machine

A sequence to a failure for A and B reaches the fail state of any product machine of A and B with added fail state.

```

lemma fail-reachable-by-sequence-to-failure :
  assumes sequence-to-failure M1 M2 io
  and well-formed M1
  and well-formed M2
  and productF M2 M1 FAIL PM
obtains p
where path PM (io||p) (initial PM) ∧ length p = length io ∧ target (io||p) (initial PM) = FAIL
proof –
  have io ≠ []
    using assms by auto
  then obtain io-init io-last where io-split[simp] : io = io-init @ [io-last]
    by (metis append-butlast-last-id)
  have io-init-inclusion : io-init ∈ language-state M1 (initial M1)
     $\wedge$  io-init ∈ language-state M2 (initial M2)
    using assms by auto

  have io-init @ [io-last] ∈ language-state M1 (initial M1)
    using assms by auto
  then obtain tr1-init tr1-last where tr1-def :
    path M1 (io-init @ [io-last] || tr1-init @ [tr1-last]) (initial M1)
     $\wedge$  length (tr1-init @ [tr1-last]) = length (io-init @ [io-last])
    by (metis append-butlast-last-id language-state-elim length-0-conv length-append-singleton
      nat.simps(3))

  then have path-init-1 : path M1 (io-init || tr1-init) (initial M1)
     $\wedge$  length tr1-init = length io-init
    by auto
  then have path M1 ([io-last] || [tr1-last]) (target (io-init || tr1-init) (initial M1))
    using tr1-def by auto
  then have succ-1 : succ M1 io-last (target (io-init || tr1-init) (initial M1)) ≠ {}
    by auto

  obtain tr2 where tr2-def : path M2 (io-init || tr2) (initial M2) ∧ length tr2 = length io-init
    using io-init-inclusion by auto
  have succ-2 : succ M2 io-last (target (io-init || tr2) (initial M2)) = {}
proof (rule ccontr)
  assume succ M2 io-last (target (io-init || tr2) (initial M2)) ≠ {}
  then obtain tr2-last where tr2-last ∈ succ M2 io-last (target (io-init || tr2) (initial M2))
    by auto
  then have path M2 ([io-last] || [tr2-last]) (target (io-init || tr2) (initial M2))
    by auto
  then have io-init @ [io-last] ∈ language-state M2 (initial M2)
    by (metis FSM.path-append language-state length-Cons length-append list.size(3) tr2-def
      zip-append)
  then show False
    using assms io-split by simp
qed

  have fail-lengths : length (io-init @ [io-last]) = length (tr2 @ [fst FAIL])
     $\wedge$  length (tr2 @ [fst FAIL]) = length (tr1-init @ [snd FAIL])
    using assms tr2-def tr1-def by auto
  then have fail-tgt : target (io-init @ [io-last] || tr2 @ [fst FAIL] || tr1-init @ [snd FAIL])
    (initial M2, initial M1) = FAIL
    by auto

  have fail-butlast-simp[simp] :
    butlast (io-init @ [io-last] || tr2 @ [fst FAIL]) = io-init || tr2
    butlast (io-init @ [io-last] || tr1-init @ [snd FAIL]) = io-init || tr1-init
    using fail-lengths by simp+

  have path M2 (butlast (io-init @ [io-last] || tr2 @ [fst FAIL])) (initial M2)
     $\wedge$  path M1 (butlast (io-init @ [io-last] || tr1-init @ [snd FAIL])) (initial M1)

```

```

using tr1-def tr2-def by auto
moreover have succ M2 (last (io-init @ [io-last]))
  (target (butlast (io-init @ [io-last] || tr2 @ [fst FAIL])) (initial M2)) = {}
using succ-2 by simp
moreover have succ M1 (last (io-init @ [io-last]))
  (target (butlast (io-init @ [io-last] || tr1-init @ [snd FAIL])) (initial M1))
  ≠ {}
using succ-1 by simp
moreover have initial M2 ∈ nodes M2 ∧ initial M1 ∈ nodes M1
by auto
ultimately have path PM (io-init @ [io-last] || tr2 @ [fst FAIL] || tr1-init @ [snd FAIL])
  (initial M2, initial M1)
using fail-lengths fail-tgt assms path-init-1 tr2-def productF-path-forward
  [of io-init @ [io-last] tr2 @ [fst FAIL] tr1-init @ [snd FAIL] M2 M1 FAIL PM
  initial M2 initial M1 ]
by simp

moreover have initial PM = (initial M2, initial M1)
using assms(4) productF-simps(4) by blast

ultimately have
  path PM (io-init @ [io-last] || tr2 @ [fst FAIL] || tr1-init @ [snd FAIL]) (initial PM)
  ∧ length (tr2 @ [fst FAIL] || tr1-init @ [snd FAIL]) = length (io-init @ [io-last])
  ∧ target (io-init @ [io-last] || tr2 @ [fst FAIL] || tr1-init @ [snd FAIL]) (initial PM) = FAIL
using fail-lengths fail-tgt by auto
then show ?thesis using that
using io-split by blast
qed

```

```

lemma fail-reachable :
assumes  $\neg M1 \preceq M2$ 
and well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
shows FAIL ∈ reachable PM (initial PM)
proof –
obtain io where sequence-to-failure M1 M2 io
using sequence-to-failure-ob assms by blast
then show ?thesis
using assms fail-reachable-by-sequence-to-failure[of M1 M2 io FAIL PM]
by (metis FSM.reachable.reflexive FSM.reachable-target)
qed

```

```

lemma fail-reachable-ob :
assumes  $\neg M1 \preceq M2$ 
and well-formed M1
and well-formed M2
and observable M2
and productF M2 M1 FAIL PM
obtains p
where path PM p (initial PM) target p (initial PM) = FAIL
using assms fail-reachable by (metis FSM.reachable-target-elim)

```

```

lemma fail-reachable-reverse :
assumes well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
and FAIL ∈ reachable PM (initial PM)
and observable M2
shows  $\neg M1 \preceq M2$ 
proof –
obtain pathF where pathF-def : path PM pathF (initial PM) ∧ target pathF (initial PM) = FAIL
using assms by auto

```

```

let ?io = map fst pathF
let ?tr2 = map fst (map snd pathF)
let ?tr1 = map snd (map snd pathF)

have initial PM ≠ FAIL
  using assms by auto
then have pathF ≠ []
  using pathF-def by auto
moreover have initial PM = (initial M2, initial M1)
  using assms by simp
ultimately have path M2 (?io || ?tr2) (initial M2) ∧ path M1 (?io || ?tr1) (initial M1) ∨
  target (?io || ?tr2 || ?tr1) (initial M2, initial M1) = FAIL ∧
  0 < length (?io) ∧
  path M2 (butlast (?io || ?tr2)) (initial M2) ∧
  path M1 (butlast (?io || ?tr1)) (initial M1) ∧
  succ M2 (last (?io)) (target (butlast (?io || ?tr2)) (initial M2)) = {} ∧
  succ M1 (last (?io)) (target (butlast (?io || ?tr1)) (initial M1)) ≠ {}
  using productF-path-reverse[of ?io ?tr2 ?tr1 M2 M1 FAIL PM initial M2 initial M1]
  using assms pathF-def
proof -
  have f1: path PM (?io || ?tr2 || ?tr1) (initial M2, initial M1)
    by (metis (no-types) ⟨initial PM = (initial M2, initial M1)⟩ pathF-def zip-map-fst-snd)
  have f2: length (?io) = length pathF ⟶ length (?io) = length (?tr2)
    by auto
  have length (?io) = length pathF ∧ length (?tr2) = length (?tr1)
    by auto
  then show ?thesis
    using f2 f1 ⟨productF M2 M1 FAIL PM⟩ ⟨well-formed M1⟩ ⟨well-formed M2⟩ by blast
qed

moreover have ¬ (path M2 (?io || ?tr2) (initial M2) ∧ path M1 (?io || ?tr1) (initial M1))
proof (rule ccontr)
  assume ¬ ¬ (path M2 (?io || ?tr2) (initial M2) ∧
    path M1 (?io || ?tr1) (initial M1))
  then have path M2 (?io || ?tr2) (initial M2)
    by simp
  then have target (?io || ?tr2) (initial M2) ∈ nodes M2
    by auto
  then have target (?io || ?tr2) (initial M2) ≠ fst FAIL
    using assms by auto
  then show False
    using pathF-def
  proof -
    have FAIL = target (map fst pathF || map fst (map snd pathF) || map snd (map snd pathF))
      (initial M2, initial M1)
      by (metis (no-types) ⟨initial PM = (initial M2, initial M1)⟩
        ⟨path PM pathF (initial PM) ∧ target pathF (initial PM) = FAIL⟩ zip-map-fst-snd)
    then show ?thesis
      using ⟨target (map fst pathF || map fst (map snd pathF)) (initial M2) ≠ fst FAIL⟩ by auto
  qed
qed

ultimately have fail-prop :
  target (?io || ?tr2 || ?tr1) (initial M2, initial M1) = FAIL ∧
  0 < length (?io) ∧
  path M2 (butlast (?io || ?tr2)) (initial M2) ∧
  path M1 (butlast (?io || ?tr1)) (initial M1) ∧
  succ M2 (last (?io)) (target (butlast (?io || ?tr2)) (initial M2)) = {} ∧
  succ M1 (last (?io)) (target (butlast (?io || ?tr1)) (initial M1)) ≠ {}
  by auto

then have ?io ∈ language-state M1 (initial M1)
proof -
  have f1: path PM (map fst pathF || map fst (map snd pathF) || map snd (map snd pathF))
    (initial M2, initial M1)

```

by (metis (no-types) ⟨initial PM = (initial M2, initial M1)⟩ pathF-def zip-map-fst-snd)  
 have  $\forall c f. c \neq \text{initial } (f::('a, 'b, 'c) \text{ FSM}) \vee c \in \text{nodes } f$   
 by blast  
 then show ?thesis  
 using f1 by (metis (no-types) assms(1) assms(2) assms(3) language-state length-map  
 productF-path-reverse-ob)

qed

moreover have  $?io \notin \text{language-state } M2$  (initial M2)

proof (rule ccontr)

assume  $\neg ?io \notin \text{language-state } M2$  (initial M2)

then have  $\text{assm} : ?io \in \text{language-state } M2$  (initial M2)

by simp

then obtain  $tr2'$  where  $tr2'\text{-def} : \text{path } M2 (?io \parallel tr2')$  (initial M2)  
 $\wedge \text{length } ?io = \text{length } tr2'$

by auto

then obtain  $tr2'\text{-init } tr2'\text{-last}$  where  $tr2'\text{-split} : tr2' = tr2'\text{-init} @ [tr2'\text{-last}]$

using fail-prop by (metis ⟨pathF  $\neq []$ ⟩ append-butlast-last-id length-0-conv map-is-Nil-conv)

have  $\text{butlast } ?io \in \text{language-state } M2$  (initial M2)

using fail-prop by auto

then have  $\{t. \text{path } M2 (\text{butlast } ?io \parallel t) \text{ (initial M2)} \wedge \text{length } (\text{butlast } ?io) = \text{length } t\}$   
 $= \{\text{butlast } ?tr2\}$

using assms(5) observable-path-unique[of butlast ?io M2 initial M2 butlast ?tr2]  
 fail-prop by fastforce

then have  $\forall t ts. \text{path } M2 ((\text{butlast } ?io) @ [\text{last } ?io] \parallel ts @ [t])$  (initial M2)  
 $\wedge \text{length } ((\text{butlast } ?io) @ [\text{last } ?io]) = \text{length } (ts @ [t])$   
 $\longrightarrow ts = \text{butlast } ?tr2$

by (metis (no-types, lifting) FSM.path-append-elim

⟨butlast (map fst pathF)  $\in \text{language-state } M2$  (initial M2)⟩ assms(5) butlast-snoc  
 butlast-zip fail-prop length-butlast length-map observable-path-unique zip-append)

then have  $tr2'\text{-init} = \text{butlast } ?tr2$

using  $tr2'\text{-def } tr2'\text{-split} \langle \text{pathF} \neq [] \rangle$  by auto

then have  $\text{path } M2 ((\text{butlast } ?io) @ [\text{last } ?io] \parallel (\text{butlast } ?tr2) @ [tr2'\text{-last}])$  (initial M2)  
 $\wedge \text{length } ((\text{butlast } ?io) @ [\text{last } ?io]) = \text{length } ((\text{butlast } ?tr2) @ [tr2'\text{-last}])$

using  $tr2'\text{-def } fail\text{-prop } tr2'\text{-split}$  by auto

then have  $\text{path } M2 ([\text{last } ?io] \parallel [tr2'\text{-last}])$   
 $(\text{target } (\text{butlast } ?io \parallel \text{butlast } ?tr2))$  (initial M2)  
 $\wedge \text{length } [\text{last } ?io] = \text{length } [tr2'\text{-last}]$

by auto

then have  $tr2'\text{-last} \in \text{succ } M2 (\text{last } (?io)) (\text{target } (\text{butlast } (?io \parallel ?tr2)))$  (initial M2))

by auto

then show False

using fail-prop by auto

qed

ultimately show ?thesis by auto

qed

lemma fail-reachable-iff[iff] :

assumes well-formed M1

and well-formed M2

and productF M2 M1 FAIL PM

and observable M2

shows  $\text{FAIL} \in \text{reachable } PM$  (initial PM)  $\longleftrightarrow \neg M1 \preceq M2$

proof

show  $\text{FAIL} \in \text{reachable } PM$  (initial PM)  $\implies \neg M1 \preceq M2$

using assms fail-reachable-reverse by blast

show  $\neg M1 \preceq M2 \implies \text{FAIL} \in \text{reachable } PM$  (initial PM)

using assms fail-reachable by blast

qed

```

lemma reaching-path-length :
  assumes productF A B FAIL AB
  and well-formed A
  and well-formed B
  and q2 ∈ reachable AB q1
  and q2 ≠ FAIL
  and q1 ∈ nodes AB
shows ∃ p . path AB p q1 ∧ target p q1 = q2 ∧ length p < card (nodes A) * card (nodes B)
proof -
  obtain p where p-def : path AB p q1 ∧ target p q1 = q2 ∧ distinct (q1 # states p q1)
  using assms reaching-path-without-repetition by (metis well-formed-productF)

  have FAIL ∉ set (q1 # states p q1)
  proof (cases p)
    case Nil
    then have q1 = q2
      using p-def by auto
    then have q1 ≠ FAIL
      using assms by auto
    then show ?thesis
      using Nil by auto
  next
    case (Cons a list)
    have FAIL ∉ set (butlast (q1 # states p q1))
    proof (rule ccontr)
      assume assm : ¬ FAIL ∉ set (butlast (q1 # states p q1))
      then obtain i where i-def : i < length (butlast (q1 # states p q1))
        ∧ butlast (q1 # states p q1) ! i = FAIL
        by (metis distinct-Ex1 distinct-butlast p-def)
      then have i < Suc (length (butlast p))
        using local.Cons by fastforce
      then have i < length p
        by (metis append-butlast-last-id length-append-singleton list.simps(3) local.Cons)

      then have butlast (q1 # states p q1) ! i = target (take i p) q1
      using i-def assm proof (induction i)
        case 0
        then show ?case by auto
      next
        case (Suc i)
        then show ?case by (metis Suc-lessD nth-Cons-Suc nth-butlast states-target-index)
      qed

      then have target (take i p) q1 = FAIL using i-def by auto
      moreover have ∀ k . k < length p → target (take k p) q1 ≠ FAIL
        using no-prefix-targets-FAIL[of A B FAIL AB p q1] assms p-def by auto
      ultimately show False
        by (metis assms(5) linorder-neqE-nat nat-less-le order-refl p-def take-all)
      qed

      moreover have last (q1 # states p q1) ≠ FAIL
        using assms(5) local.Cons p-def transition-system-universal.target-alt-def by force
      ultimately show ?thesis
        by (metis (no-types, lifting) UnE append-butlast-last-id list.set(1) list.set(2)
          list.simps(3) set-append singletonD)
    qed

  moreover have set (q1 # states p q1) ⊆ nodes AB
    using assms by (metis FSM.nodes-states insert-subset list.simps(15) p-def)
  ultimately have states-subset : set (q1 # states p q1) ⊆ nodes A × nodes B
    using nodes-productF assms by blast

  have finite-nodes : finite (nodes A × nodes B)

```

```

using assms(2) assms(3) by auto
have length p ≤ length (states p q1)
by simp
then have length p < card (nodes A) * card (nodes B)
by (metis (no-types) finite-nodes states-subset card-cartesian-product card-mono distinct-card impossible-Cons less-le-trans not-less p-def)

then show ?thesis
using p-def by blast
qed

```

```

lemma reaching-path-fail-length :
assumes productF A B FAIL AB
and well-formed A
and well-formed B
and q2 ∈ reachable AB q1
and q1 ∈ nodes AB
shows  $\exists p . \text{path } AB \ p \ q1 \wedge \text{target } p \ q1 = q2 \wedge \text{length } p \leq \text{card } (\text{nodes } A) * \text{card } (\text{nodes } B)$ 
proof (cases q2 = FAIL)
case True

```

```

then have q2-def : q2 = FAIL
by simp
then show ?thesis
proof (cases q1 = q2)
case True
then show ?thesis by auto
next
case False
then obtain px where px-def : path AB px q1 ∧ target px q1 = q2
using assms by auto
then have px-nonempty : px ≠ []
using q2-def False by auto
let ?qx = target (butlast px) q1
have ?qx ∈ reachable AB q1
using px-def px-nonempty
by (metis FSM.path-append-elim FSM.reachable.reflexive FSM.reachable-target append-butlast-last-id)
moreover have ?qx ≠ FAIL
using False q2-def assms
by (metis One-nat-def Suc-pred butlast-conv-take length-greater-0-conv lessI no-prefix-targets-FAIL px-def px-nonempty)
ultimately obtain px' where px'-def : path AB px' q1
 $\wedge \text{target } px' \ q1 = ?qx$ 
 $\wedge \text{length } px' < \text{card } (\text{nodes } A) * \text{card } (\text{nodes } B)$ 
using assms reaching-path-length[of A B FAIL AB ?qx q1] by blast

```

```

have px-split : path AB ((butlast px) @ [last px]) q1
 $\wedge \text{target } ((\text{butlast } px) @ [\text{last } px]) \ q1 = q2$ 
using px-def px-nonempty by auto
then have path AB [last px] ?qx ∧ target [last px] ?qx = q2
using px-nonempty
proof –
have target [last px] (target (butlast px) q1) = q2
using px-split by force
then show ?thesis
using px-split by blast
qed

```

```

then have path AB (px' @ [last px]) q1 ∧ target (px' @ [last px]) q1 = q2
using px'-def by auto
moreover have length (px' @ [last px]) ≤ card (nodes A) * card (nodes B)
using px'-def by auto
ultimately show ?thesis
by blast

```

```

qed
next
case False
then show ?thesis
using assms reaching-path-length by (metis less-imp-le)
qed

lemma productF-language :
  assumes productF A B FAIL AB
  and well-formed A
  and well-formed B
  and  $io \in L A \cap L B$ 
shows  $io \in L AB$ 
proof -
  obtain trA trB where tr-def :  $path A (io \parallel trA) (initial A) \wedge length\ io = length\ trA$ 
   $path B (io \parallel trB) (initial B) \wedge length\ io = length\ trB$ 
  using assms by blast
  then have  $path\ AB\ (io \parallel trA \parallel trB)\ (initial\ A,\ initial\ B)$ 
  using assms by (metis FSM.nodes.initial productF-path-inclusion)
  then show ?thesis
  using tr-def by (metis assms(1) language-state length-zip min.idem productF-simps(4))
qed

lemma productF-language-state-intermediate :
  assumes  $vs @ xs \in L M2 \cap L M1$ 
  and productF M2 M1 FAIL PM
  and observable M2
  and well-formed M2
  and observable M1
  and well-formed M1
obtains q2 q1 tr
where io-targets PM (initial PM) vs = {(q2,q1)}
 $path\ PM\ (xs \parallel tr)\ (q2,q1)$ 
 $length\ xs = length\ tr$ 
proof -
  have  $vs @ xs \in L PM$ 
  using productF-language[OF assms(2,4,6,1)] by simp
  then obtain trVX where  $path\ PM\ (vs @ xs \parallel trVX)\ (initial\ PM) \wedge length\ trVX = length\ (vs @ xs)$ 
  by auto
  then have  $tgt-VX : io-targets\ PM\ (initial\ PM)\ (vs @ xs) = \{target\ (vs @ xs \parallel trVX)\ (initial\ PM)\}$ 
  by (metis assms(2) assms(3) assms(5) obs-target-is-io-targets observable-productF)

  have  $vs \in L PM$  using  $\langle vs @ xs \in L PM \rangle$ 
  by (meson language-state-prefix)
  then obtain trV where  $path\ PM\ (vs \parallel trV)\ (initial\ PM) \wedge length\ trV = length\ vs$ 
  by auto
  then have  $tgt-V : io-targets\ PM\ (initial\ PM)\ vs = \{target\ (vs \parallel trV)\ (initial\ PM)\}$ 
  by (metis assms(2) assms(3) assms(5) obs-target-is-io-targets observable-productF)

  let ?q2 = fst (target (vs || trV) (initial PM))
  let ?q1 = snd (target (vs || trV) (initial PM))

  have observable PM
  by (meson assms(2,3,5) observable-productF)

  have  $io-targets\ PM\ (?q2,?q1)\ xs = \{target\ (vs @ xs \parallel trVX)\ (initial\ PM)\}$ 
  using observable-io-targets-split[OF  $\langle observable\ PM \rangle\ tgt-VX\ tgt-V$ ] by simp

  then have  $xs \in language-state\ PM\ (?q2,?q1)$ 
  by auto

  then obtain tr where  $path\ PM\ (xs \parallel tr)\ (?q2,?q1)$ 
   $length\ xs = length\ tr$ 
  by auto

```

**then show** *?thesis*  
**by** (*metis prod.collapse tgt-V that*)  
**qed**

**lemma** *sequence-to-failure-reaches-FAIL* :  
**assumes** *sequence-to-failure M1 M2 io*  
**and** *OFSM M1*  
**and** *OFSM M2*  
**and** *productF M2 M1 FAIL PM*  
**shows** *FAIL ∈ io-targets PM (initial PM) io*  
**proof** –  
**obtain** *p* **where** *path PM (io || p) (initial PM)*  
 $\wedge$  *length p = length io*  
 $\wedge$  *target (io || p) (initial PM) = FAIL*  
**using** *fail-reachable-by-sequence-to-failure[OF assms(1)]*  
**using** *assms(2) assms(3) assms(4)* **by** *blast*  
**then show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *sequence-to-failure-reaches-FAIL-ob* :  
**assumes** *sequence-to-failure M1 M2 io*  
**and** *OFSM M1*  
**and** *OFSM M2*  
**and** *productF M2 M1 FAIL PM*  
**shows** *io-targets PM (initial PM) io = {FAIL}*  
**proof** –  
**have** *FAIL ∈ io-targets PM (initial PM) io*  
**using** *sequence-to-failure-reaches-FAIL[OF assms(1–4)]* **by** *assumption*  
**have** *observable PM*  
**by** (*meson assms(2) assms(3) assms(4) observable-productF*)  
**show** *?thesis*  
**by** (*meson ⟨FAIL ∈ io-targets PM (initial PM) io⟩ ⟨observable PM⟩*  
*observable-io-target-is-singleton*)  
**qed**

**lemma** *sequence-to-failure-alt-def* :  
**assumes** *io-targets PM (initial PM) io = {FAIL}*  
**and** *OFSM M1*  
**and** *OFSM M2*  
**and** *productF M2 M1 FAIL PM*  
**shows** *sequence-to-failure M1 M2 io*  
**proof** –  
**obtain** *p* **where** *path PM (io || p) (initial PM)*  
 $\wedge$  *length p = length io*  
 $\wedge$  *target (io || p) (initial PM) = FAIL*  
**using** *assms(1)* **by** (*metis io-targets-elim singletonI*)  
**have** *io ≠ []*  
**proof**  
**assume** *io = []*  
**then have** *io-targets PM (initial PM) io = {initial PM}*  
**by** *auto*  
**moreover have** *initial PM ≠ FAIL*  
**proof** –  
**have** *initial PM = (initial M2, initial M1)*  
**using** *assms(4)* **by** *auto*  
**then have** *initial PM ∈ (nodes M2 × nodes M1)*  
**by** (*simp add: FSM.nodes.initial*)  
**moreover have** *FAIL ∉ (nodes M2 × nodes M1)*  
**using** *assms(4)* **by** *auto*  
**ultimately show** *?thesis*  
**by** *auto*

```

qed
ultimately show False
  using assms(1) by blast
qed
then have  $0 < \text{length } io$ 
  by blast

have target (butlast (io||p)) (initial PM)  $\neq$  FAIL
  using no-prefix-targets-FAIL[OF assms(4)  $\langle$ path PM (io || p) (initial PM) $\rangle$ , of (length io) - 1]
  by (metis (no-types, lifting)  $\langle$  $0 < \text{length } io$  $\rangle$   $\langle$ length p = length io $\rangle$  butlast-conv-take
    diff-less length-map less-numeral-extra(1) map-fst-zip)
have target (butlast (io||p)) (initial PM)  $\in$  nodes PM
  by (metis FSM.nodes.initial FSM.nodes-target FSM.path-append-elim
     $\langle$ path PM (io || p) (initial PM) $\rangle$  append-butlast-last-id butlast.simps(1))
moreover have nodes PM  $\subseteq$  insert FAIL (nodes M2  $\times$  nodes M1)
  using nodes-productF[OF - - assms(4)] assms(2) assms(3) by linarith
ultimately have target (butlast (io||p)) (initial PM)  $\in$  insert FAIL (nodes M2  $\times$  nodes M1)
  by blast

have target (butlast (io||p)) (initial PM)  $\in$  (nodes M2  $\times$  nodes M1)
  using  $\langle$ target (butlast (io || p)) (initial PM)  $\in$  insert FAIL (nodes M2  $\times$  nodes M1) $\rangle$ 
     $\langle$ target (butlast (io || p)) (initial PM)  $\neq$  FAIL $\rangle$ 
  by blast
then obtain s2 s1 where target (butlast (io||p)) (initial PM) = (s2,s1)
   $s2 \in$  nodes M2 s1  $\in$  nodes M1
  by blast

have length (butlast io) = length (map fst (butlast p))
  length (map fst (butlast p)) = length (map snd (butlast p))
  by (simp add:  $\langle$ length p = length io $\rangle$ )+

have path PM (butlast (io||p)) (initial PM)
  by (metis FSM.path-append-elim  $\langle$ path PM (io || p) (initial PM) $\rangle$  append-butlast-last-id
    butlast.simps(1))
then have path PM ( $((\text{butlast } io) \parallel (\text{map fst } (\text{butlast } p)) \parallel (\text{map snd } (\text{butlast } p)))$ )
  (initial M2, initial M1)
  using  $\langle$ length p = length io $\rangle$  assms(4) by auto
have target (butlast io || map fst (butlast p) || map snd (butlast p)) (initial M2, initial M1)
   $\neq$  FAIL
  using  $\langle$ length p = length io $\rangle$   $\langle$ target (butlast (io || p)) (initial PM)  $\neq$  FAIL $\rangle$  assms(4)
  by auto

have path M2 (butlast io || map fst (butlast p)) (initial M2)  $\wedge$ 
  path M1 (butlast io || map snd (butlast p)) (initial M1)  $\vee$ 
  target (butlast io || map fst (butlast p) || map snd (butlast p)) (initial M2, initial M1)
  = FAIL
  using productF-path-reverse
    [OF  $\langle$ length (butlast io) = length (map fst (butlast p)) $\rangle$ 
       $\langle$ length (map fst (butlast p)) = length (map snd (butlast p)) $\rangle$ 
      assms(4) - -
       $\langle$ path PM ( $((\text{butlast } io) \parallel (\text{map fst } (\text{butlast } p)) \parallel (\text{map snd } (\text{butlast } p)))$ )
        (initial M2, initial M1) $\rangle$  - -]
  using assms(2) assms(3) by auto
then have path M2 (butlast io || map fst (butlast p)) (initial M2)
  path M1 (butlast io || map snd (butlast p)) (initial M1)
  using  $\langle$ target (butlast io || map fst (butlast p) || map snd (butlast p))
    (initial M2, initial M1)  $\neq$  FAIL $\rangle$ 
  by auto

then have butlast io  $\in$  L M2  $\cap$  L M1
  using  $\langle$ length (butlast io) = length (map fst (butlast p)) $\rangle$  by auto

have path PM (io || map fst p || map snd p) (initial M2, initial M1)
  using  $\langle$ path PM (io || p) (initial PM) $\rangle$  assms(4) by auto
have length io = length (map fst p)
  length (map fst p) = length (map snd p)

```

```

by (simp add: ⟨length p = length io⟩)+

obtain p1' where path M1 (io || p1') (initial M1) ∧ length io = length p1'
using productF-path-reverse-ob
  [OF ⟨length io = length (map fst p)⟩
    ⟨length (map fst p) = length (map snd p)⟩ assms(4) - -
    ⟨path PM (io || map fst p || map snd p) (initial M2, initial M1)⟩]
using assms(2) assms(3) by blast
then have io ∈ L M1
  by auto

moreover have io ∉ L M2
proof
  assume io ∈ L M2 — only possible if io does not target FAIL
  then obtain p2' where path M2 (io || p2') (initial M2) length io = length p2'
    by auto
  then have length p2' = length p1'
    using ⟨path M1 (io || p1') (initial M1) ∧ length io = length p1'⟩
    by auto

  have path PM (io || p2' || p1') (initial M2, initial M1)
    using productF-path-inclusion[OF ⟨length io = length p2'⟩ ⟨length p2' = length p1'⟩ assms(4),
      of initial M2 initial M1]
      ⟨path M1 (io || p1') (initial M1) ∧ length io = length p1'⟩
      ⟨path M2 (io || p2') (initial M2)⟩ assms(2) assms(3)
    by blast

  have target (io || p2' || p1') (initial M2, initial M1) ∈ (nodes M2 × nodes M1)
    using ⟨length io = length p2'⟩ ⟨path M1 (io || p1') (initial M1) ∧ length io = length p1'⟩
      ⟨path M2 (io || p2') (initial M2)⟩
    by auto
  moreover have FAIL ∉ (nodes M2 × nodes M1)
    using assms(4) by auto
  ultimately have target (io || p2' || p1') (initial M2, initial M1) ≠ FAIL
    by blast

  have length io = length (p2' || p1')
    by (simp add: ⟨length io = length p2'⟩ ⟨length p2' = length p1'⟩)
  have target (io || p2' || p1') (initial M2, initial M1)
    ∈ io-targets PM (initial M2, initial M1) io
    using ⟨path PM (io || p2' || p1') (initial M2, initial M1)⟩ ⟨length io = length (p2' || p1')⟩
    unfolding io-targets.simps by blast

  have io-targets PM (initial PM) io ≠ {FAIL}
    using ⟨target (io || p2' || p1') (initial M2, initial M1)
      ∈ io-targets PM (initial M2, initial M1) io⟩
      ⟨target (io || p2' || p1') (initial M2, initial M1) ≠ FAIL⟩ assms(4)
    by auto
  then show False
    using assms(1) by blast
qed

ultimately have io ∈ L M1 - L M2
  by blast

show sequence-to-failure M1 M2 io
  using ⟨butlast io ∈ L M2 ∩ L M1⟩ ⟨io ∈ L M1 - L M2⟩ by auto
qed

end
theory ATC
imports ../FSM/FSM
begin

```

### 3 Adaptive test cases

Adaptive test cases (ATCs) are tree-like structures that label nodes with inputs and edges with outputs such that applying an ATC to some FSM is performed by applying the label of its root node and then applying the ATC connected to the root node by an edge labeled with the observed output of the FSM. The result of such an application is here called an ATC-reaction.

ATCs are here modelled to have edges for every possible output from each non-leaf node. This is not a restriction on the definition of ATCs by Hierons [2] as a missing edge can be expressed by an edge to a leaf.

**datatype** (*'in, 'out*) *ATC* = *Leaf* | *Node 'in 'out*  $\Rightarrow$  (*'in, 'out*) *ATC*

**inductive** *atc-reaction* :: (*'in, 'out, 'state*) *FSM*  $\Rightarrow$  *'state*  $\Rightarrow$  (*'in, 'out*) *ATC*  
 $\Rightarrow$  (*'in*  $\times$  *'out*) *list*  $\Rightarrow$  *bool*

**where**

*leaf*[*intro!*]: *atc-reaction* *M* *q1* *Leaf* [] |

*node*[*intro!*]: *q2*  $\in$  *succ* *M* (*x,y*) *q1*

$\Rightarrow$  *atc-reaction* *M* *q2* (*f y*) *io*

$\Rightarrow$  *atc-reaction* *M* *q1* (*Node* *x f*) ((*x,y*)#*io*)

**inductive-cases** *leaf-elim*[*elim!*] : *atc-reaction* *M* *q1* *Leaf* []

**inductive-cases** *node-elim*[*elim!*] : *atc-reaction* *M* *q1* (*Node* *x f*) ((*x,y*)#*io*)

#### 3.1 Properties of ATC-reactions

**lemma** *atc-reaction-empty*[*simp*] :

**assumes** *atc-reaction* *M* *q* *t* []

**shows** *t* = *Leaf*

**using** *assms* *atc-reaction.simps* **by** *force*

**lemma** *atc-reaction-nonempty-no-leaf* :

**assumes** *atc-reaction* *M* *q* *t* (*Cons* *a* *io*)

**shows** *t*  $\neq$  *Leaf*

**using** *assms*

**proof** –

**have**  $\bigwedge f c a ps. \neg$  *atc-reaction* *f* (*c::'c*) (*a::('a, 'b)* *ATC*) *ps*  $\vee$  *a*  $\neq$  *Leaf*  $\vee$  *a*  $\neq$  *Leaf*  $\vee$  *ps* = []

**using** *atc-reaction.simps* **by** *fastforce*

**then show** *?thesis*

**using** *assms* **by** *blast*

**qed**

**lemma** *atc-reaction-nonempty*[*elim*] :

**assumes** *atc-reaction* *M* *q1* *t* (*Cons* (*x,y*) *io*)

**obtains** *q2* *f*

**where** *t* = *Node* *x f* *q2*  $\in$  *succ* *M* (*x,y*) *q1* *atc-reaction* *M* *q2* (*f y*) *io*

**proof** –

**obtain** *x2 f* **where** *t* = *Node* *x2 f*

**using** *assms* **by** (*metis* *ATC.exhaust* *atc-reaction-nonempty-no-leaf*)

**moreover have** *x* = *x2*

**using** *assms* *calculation* *atc-reaction.cases* **by** *fastforce*

**ultimately show** *?thesis*

**using** *assms* **using** *that* **by** *blast*

**qed**

**lemma** *atc-reaction-path-ex* :

**assumes** *atc-reaction* *M* *q1* *t* *io*

**shows**  $\exists$  *tr* . *path* *M* (*io* || *tr*) *q1*  $\wedge$  *length* *io* = *length* *tr*

**using** *assms* **proof** (*induction* *io* *arbitrary*: *q1* *t* *rule*: *list.induct*)

**case** *Nil*

**then show** *?case* **by** (*simp* *add*: *FSM.nil*)

**next**

**case** (*Cons* *io-hd* *io-tl*)

**then obtain** *x y* **where** *io-hd-def* : *io-hd* = (*x,y*)

**by** (*meson* *surj-pair*)

**then obtain** *f* **where** *f-def* : *t* = (*Node* *x f*)

**using** *Cons* *atc-reaction-nonempty* **by** *metis*

**then obtain** *q2* **where** *q2-def* : *q2*  $\in$  *succ* *M* (*x,y*) *q1* *atc-reaction* *M* *q2* (*f y*) *io-tl*

```

  using Cons io-hd-def atc-reaction-nonempty by auto
then obtain tr-tl where tr-tl-def : path M (io-tl || tr-tl) q2 length io-tl = length tr-tl
  using Cons.IH[of q2 f y] by blast
then have path M (io-hd # io-tl || q2 # tr-tl) q1
  using Cons q2-def by (simp add: FSM.path.intros(2) io-hd-def)
then show ?case using tr-tl-def by fastforce
qed

```

```

lemma atc-reaction-path[elim] :
  assumes atc-reaction M q1 t io
obtains tr
  where path M (io || tr) q1 length io = length tr
by (meson assms atc-reaction-path-ex)

```

### 3.2 Applicability

An ATC can be applied to an FSM if each node-label is contained in the input alphabet of the FSM.

```

inductive subtest :: ('in, 'out) ATC ⇒ ('in, 'out) ATC ⇒ bool where
  t ∈ range f ⇒ subtest t (Node x f)

```

```

lemma accp-subtest : Wellfounded.accp subtest t
proof (induction t)
  case Leaf
  then show ?case by (meson ATC.distinct(1) accp.simps subtest.cases)
next
  case (Node x f)
  have IH: Wellfounded.accp subtest t if t ∈ range f for t
    using Node[of t] and that by (auto simp: eq-commute)
  show ?case by (rule accpI) (auto intro: IH elim!: subtest.cases)
qed

```

```

definition subtest-rel where subtest-rel = {(t, Node x f) | f x t. t ∈ range f}

```

```

lemma subtest-rel-altdef: subtest-rel = {(s, t) | s t. subtest s t}
by (auto simp: subtest-rel-def subtest.simps)

```

```

lemma subtest-relI [intro]: t ∈ range f ⇒ (t, Node x f) ∈ subtest-rel
by (simp add: subtest-rel-def)

```

```

lemma subtest-relI' [intro]: t = f y ⇒ (t, Node x f) ∈ subtest-rel
by (auto simp: subtest-rel-def ran-def)

```

```

lemma wf-subtest-rel [simp, intro]: wf subtest-rel
  using accp-subtest unfolding subtest-rel-altdef accp-eq-acc wf-iff-acc
  by auto

```

```

function inputs-atc :: ('a, 'b) ATC ⇒ 'a set where
  inputs-atc Leaf = {} |
  inputs-atc (Node x f) = insert x (⋃ (image inputs-atc (range f)))
by pat-completeness auto
termination by (relation subtest-rel) auto

```

```

fun applicable :: ('in, 'out, 'state) FSM ⇒ ('in, 'out) ATC ⇒ bool where
  applicable M t = (inputs-atc t ⊆ inputs M)

```

```

fun applicable-set :: ('in, 'out, 'state) FSM ⇒ ('in, 'out) ATC set ⇒ bool where
  applicable-set M Ω = (∀ t ∈ Ω . applicable M t)

```

```

lemma applicable-subtest :
  assumes applicable M (Node x f)
shows applicable M (f y)
using assms inputs-atc.simps
by (simp add: Sup-le-iff)

```

### 3.3 Application function IO

Function IO collects all ATC-reactions of some FSM to some ATC.

```
fun IO :: ('in, 'out, 'state) FSM  $\Rightarrow$  'state  $\Rightarrow$  ('in, 'out) ATC  $\Rightarrow$  ('in  $\times$  'out) list set where
  IO M q t = { tr . atc-reaction M q t tr }
```

```
fun IO-set :: ('in, 'out, 'state) FSM  $\Rightarrow$  'state  $\Rightarrow$  ('in, 'out) ATC set  $\Rightarrow$  ('in  $\times$  'out) list set
where
  IO-set M q  $\Omega$  =  $\bigcup$  { IO M q t | t . t  $\in$   $\Omega$  }
```

```
lemma IO-language : IO M q t  $\subseteq$  language-state M q
by (metis atc-reaction-path IO.elims language-state mem-Collect-eq subsetI)
```

```
lemma IO-leaf[simp] : IO M q Leaf = {[]}
```

**proof**

```
show IO M q Leaf  $\subseteq$  {[]}
```

```
proof (rule ccontr)
```

```
  assume assm :  $\neg$  IO M q Leaf  $\subseteq$  {[]}
```

```
  then obtain io-hd io-tl where elem-ex : Cons io-hd io-tl  $\in$  IO M q Leaf
```

```
    by (metis (no-types, opaque-lifting) insertI1 neq-Nil-conv subset-eq)
```

```
  then show False
```

```
    using atc-reaction-nonempty-no-leaf assm by (metis IO.simps mem-Collect-eq)
```

```
qed
```

**next**

```
show {[]} $\subseteq$  IO M q Leaf by auto
```

qed

```
lemma IO-applicable-nonempty :
```

```
  assumes applicable M t
```

```
  and completely-specified M
```

```
  and q1  $\in$  nodes M
```

```
  shows IO M q1 t  $\neq$  {}
```

```
using assms proof (induction t arbitrary: q1)
```

```
  case Leaf
```

```
  then show ?case by auto
```

**next**

```
  case (Node x f)
```

```
  then have x  $\in$  inputs M by auto
```

```
  then obtain y q2 where x-appl : q2  $\in$  succ M (x, y) q1
```

```
    using Node unfolding completely-specified.simps by blast
```

```
  then have applicable M (f y)
```

```
    using applicable-subtest Node by metis
```

```
  moreover have q2  $\in$  nodes M
```

```
    using Node(4)  $\langle$ q2  $\in$  succ M (x, y) q1 $\rangle$  FSM.nodes.intros(2)[of q1 M ((x,y),q2)] by auto
```

```
  ultimately have IO M q2 (f y)  $\neq$  {}
```

```
    using Node by auto
```

```
  then show ?case unfolding IO.simps
```

```
    using x-appl by blast
```

qed

```
lemma IO-in-language :
```

```
  IO M q t  $\subseteq$  LS M q
```

```
  unfolding IO.simps by blast
```

```
lemma IO-set-in-language :
```

```
  IO-set M q  $\Omega$   $\subseteq$  LS M q
```

```
  using IO-in-language[of M q] unfolding IO-set.simps by blast
```

### 3.4 R-distinguishability

A non-empty ATC r-distinguishes two states of some FSM if there exists no shared ATC-reaction.

```
fun r-dist :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('in, 'out) ATC  $\Rightarrow$  'state  $\Rightarrow$  'state  $\Rightarrow$  bool where
  r-dist M t s1 s2 = (t  $\neq$  Leaf  $\wedge$  IO M s1 t  $\cap$  IO M s2 t = {})
```

```

fun r-dist-set :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('in, 'out) ATC set  $\Rightarrow$  'state  $\Rightarrow$  'state  $\Rightarrow$  bool where
r-dist-set M T s1 s2 = ( $\exists$  t  $\in$  T . r-dist M t s1 s2)

```

```

lemma r-dist-dist :
  assumes applicable M t
  and    completely-specified M
  and    r-dist M t q1 q2
  and    q1  $\in$  nodes M
shows   q1  $\neq$  q2
proof (rule ccontr)
  assume  $\neg$ (q1  $\neq$  q2)
  then have q1 = q2
    by simp
  then have IO M q1 t = {}
    using assms by simp
  moreover have IO M q1 t  $\neq$  {}
    using assms IO-applicable-nonempty by auto
  ultimately show False
    by simp
qed

```

```

lemma r-dist-set-dist :
  assumes applicable-set M  $\Omega$ 
  and    completely-specified M
  and    r-dist-set M  $\Omega$  q1 q2
  and    q1  $\in$  nodes M
shows   q1  $\neq$  q2
using assms r-dist-dist by (metis applicable-set.elims(2) r-dist-set.elims(2))

```

```

lemma r-dist-set-dist-disjoint :
  assumes applicable-set M  $\Omega$ 
  and    completely-specified M
  and     $\forall$  t1  $\in$  T1 .  $\forall$  t2  $\in$  T2 . r-dist-set M  $\Omega$  t1 t2
  and    T1  $\subseteq$  nodes M
shows   T1  $\cap$  T2 = {}
  by (metis assms disjoint-iff-not-equal r-dist-set-dist subsetCE)

```

### 3.5 Response sets

The following functions calculate the sets of all ATC-reactions observed by applying some set of ATCs on every state reached in some FSM using a given set of IO-sequences.

```

fun B :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('in * 'out) list  $\Rightarrow$  ('in, 'out) ATC set
       $\Rightarrow$  ('in * 'out) list set where
  B M io  $\Omega$  =  $\bigcup$  (image ( $\lambda$  s . IO-set M s  $\Omega$ ) (io-targets M (initial M) io))

```

```

fun D :: ('in, 'out, 'state) FSM  $\Rightarrow$  'in list set  $\Rightarrow$  ('in, 'out) ATC set
       $\Rightarrow$  ('in * 'out) list set set where
  D M ISeqs  $\Omega$  = image ( $\lambda$  io . B M io  $\Omega$ ) (LSin M (initial M) ISeqs)

```

```

fun append-io-B :: ('in, 'out, 'state) FSM  $\Rightarrow$  ('in * 'out) list  $\Rightarrow$  ('in, 'out) ATC set
       $\Rightarrow$  ('in * 'out) list set where
  append-io-B M io  $\Omega$  = { io@res | res . res  $\in$  B M io  $\Omega$  }

```

```

lemma B-dist' :
  assumes df: B M io1  $\Omega$   $\neq$  B M io2  $\Omega$ 
shows   (io-targets M (initial M) io1)  $\neq$  (io-targets M (initial M) io2)
using assms by force

```

```

lemma B-dist :

```

**assumes**  $io\text{-targets } M \text{ (initial } M) io1 = \{q1\}$   
**and**  $io\text{-targets } M \text{ (initial } M) io2 = \{q2\}$   
**and**  $B M io1 \Omega \neq B M io2 \Omega$   
**shows**  $q1 \neq q2$   
**using** *assms* **by force**

**lemma** *D-bound* :

**assumes** *wf*: *well-formed*  $M$   
**and** *ob*: *observable*  $M$   
**and** *fi*: *finite*  $ISeqs$   
**shows**  $finite (D M ISeqs \Omega) \text{ card } (D M ISeqs \Omega) \leq \text{card } (nodes M)$   
**proof** –  
**have**  $D M ISeqs \Omega \subseteq \text{image } (\lambda s . IO\text{-set } M s \Omega) (nodes M)$   
**proof**  
**fix**  $RS$  **assume**  $RS\text{-def} : RS \in D M ISeqs \Omega$   
**then obtain**  $xs \ ys$  **where**  $RS\text{-tr} : RS = B M (xs \parallel ys) \Omega$   
 $(xs \in ISeqs \wedge \text{length } xs = \text{length } ys$   
 $\wedge (xs \parallel ys) \in \text{language-state } M \text{ (initial } M))$   
**by auto**  
**then obtain**  $qx$  **where**  $qx\text{-def} : io\text{-targets } M \text{ (initial } M) (xs \parallel ys) = \{qx\}$   
**by** (*meson io-targets-observable-singleton-ex ob*)  
**then have**  $RS = IO\text{-set } M qx \Omega$   
**using**  $RS\text{-tr}$  **by auto**  
**moreover have**  $qx \in nodes M$   
**by** (*metis FSM.nodes.initial io-targets-nodes qx-def singletonI*)  
**ultimately show**  $RS \in \text{image } (\lambda s . IO\text{-set } M s \Omega) (nodes M)$   
**by auto**  
**qed**  
**moreover have**  $finite (nodes M)$   
**using** *assms* **by auto**  
**ultimately show**  $finite (D M ISeqs \Omega) \text{ card } (D M ISeqs \Omega) \leq \text{card } (nodes M)$   
**by** (*meson finite-imageI infinite-super surj-card-le*)  
**qed**

**lemma** *append-io-B-in-language* :

$append\text{-io-B } M io \Omega \subseteq L M$   
**proof**  
**fix**  $x$  **assume**  $x \in append\text{-io-B } M io \Omega$   
**then obtain**  $res$  **where**  $x = io@res \text{ res} \in B M io \Omega$   
**unfolding** *append-io-B.simps* **by blast**  
**then obtain**  $q$  **where**  $q \in io\text{-targets } M \text{ (initial } M) io \text{ res} \in IO\text{-set } M q \Omega$   
**unfolding** *B.simps* **by blast**  
**then have**  $res \in LS M q$   
**using** *IO-set-in-language[of M q  $\Omega$ ]* **by blast**  
  
**obtain**  $pIO$  **where**  $path M (io \parallel pIO) \text{ (initial } M)$   
 $\text{length } pIO = \text{length } io \text{ target } (io \parallel pIO) \text{ (initial } M) = q$   
**using**  $\langle q \in io\text{-targets } M \text{ (initial } M) io \rangle$  **by auto**  
**moreover obtain**  $pRes$  **where**  $path M (res \parallel pRes) q \text{ length } pRes = \text{length } res$   
**using**  $\langle res \in LS M q \rangle$  **by auto**  
**ultimately have**  $io@res \in L M$   
**using** *FSM.path-append[of M io||pIO initial M res||pRes]*  
**by** (*metis language-state length-append zip-append*)  
**then show**  $x \in L M$   
**using**  $\langle x = io@res \rangle$  **by blast**  
**qed**

**lemma** *append-io-B-nonempty* :

**assumes** *applicable-set*  $M \Omega$   
**and** *completely-specified*  $M$   
**and**  $io \in \text{language-state } M \text{ (initial } M)$   
**and**  $\Omega \neq \{\}$   
**shows**  $append\text{-io-B } M io \Omega \neq \{\}$

**proof** –  
**obtain**  $t \in \Omega$   
**using**  $assms(4)$  **by**  $blast$   
**then have**  $applicable\ M\ t$   
**using**  $assms(1)$  **by**  $simp$   
**moreover obtain**  $tr$  **where**  $path\ M\ (io\ ||\ tr)\ (initial\ M) \wedge length\ tr = length\ io$   
**using**  $assms(3)$  **by**  $auto$   
**moreover have**  $target\ (io\ ||\ tr)\ (initial\ M) \in nodes\ M$   
**using**  $calculation(2)$  **by**  $blast$   
**ultimately have**  $IO\ M\ (target\ (io\ ||\ tr)\ (initial\ M))\ t \neq \{\}$   
**using**  $assms(2)$   $IO$ - $applicable$ - $nonempty$  **by**  $simp$   
**then obtain**  $io'$  **where**  $io' \in IO\ M\ (target\ (io\ ||\ tr)\ (initial\ M))\ t$   
**by**  $blast$   
**then have**  $io' \in IO$ - $set\ M\ (target\ (io\ ||\ tr)\ (initial\ M))\ \Omega$   
**using**  $\langle t \in \Omega \rangle$  **unfolding**  $IO$ - $set$ . $simps$  **by**  $blast$   
**moreover have**  $(target\ (io\ ||\ tr)\ (initial\ M)) \in io$ - $targets\ M\ (initial\ M)\ io$   
**using**  $\langle path\ M\ (io\ ||\ tr)\ (initial\ M) \wedge length\ tr = length\ io \rangle$  **by**  $auto$   
**ultimately have**  $io' \in B\ M\ io\ \Omega$   
**unfolding**  $B$ . $simps$  **by**  $blast$   
**then have**  $io@io' \in append$ - $io$ - $B\ M\ io\ \Omega$   
**unfolding**  $append$ - $io$ - $B$ . $simps$  **by**  $blast$   
**then show**  $?thesis$  **by**  $blast$   
**qed**

**lemma**  $append$ - $io$ - $B$ - $prefix$ - $in$ - $language$  :  
**assumes**  $append$ - $io$ - $B\ M\ io\ \Omega \neq \{\}$   
**shows**  $io \in L\ M$

**proof** –  
**obtain**  $res$  **where**  $io\ @\ res \in append$ - $io$ - $B\ M\ io\ \Omega \wedge res \in B\ M\ io\ \Omega$   
**using**  $assms$  **by**  $auto$   
**then have**  $io$ - $targets\ M\ (initial\ M)\ io \neq \{\}$   
**by**  $auto$   
**then obtain**  $q$  **where**  $q \in io$ - $targets\ M\ (initial\ M)\ io$   
**by**  $blast$   
**then obtain**  $tr$  **where**  $target\ (io\ ||\ tr)\ (initial\ M) = q \wedge path\ M\ (io\ ||\ tr)\ (initial\ M)$   
 $\wedge length\ tr = length\ io$  **by**  $auto$   
**then show**  $?thesis$  **by**  $auto$   
**qed**

### 3.6 Characterizing sets

A set of ATCs is a characterizing set for some FSM if for every pair of r-distinguishable states it contains an ATC that r-distinguishes them.

**fun**  $characterizing$ - $atc$ - $set$  ::  $(in, out, state)\ FSM \Rightarrow (in, out)\ ATC\ set \Rightarrow bool$  **where**  
 $characterizing$ - $atc$ - $set\ M\ \Omega = (applicable$ - $set\ M\ \Omega \wedge (\forall s1 \in (nodes\ M) . \forall s2 \in (nodes\ M) .$   
 $(\exists td . r$ - $dist\ M\ td\ s1\ s2) \longrightarrow (\exists tt \in \Omega . r$ - $dist\ M\ tt\ s1\ s2)))$

### 3.7 Reduction over ATCs

Some state is a an ATC-reduction of another over some set of ATCs if for every contained ATC every ATC-reaction to it of the former state is also an ATC-reaction of the latter state.

**fun**  $atc$ - $reduction$  ::  $(in, out, state)\ FSM \Rightarrow state \Rightarrow (in, out, state)\ FSM \Rightarrow state$   
 $\Rightarrow (in, out)\ ATC\ set \Rightarrow bool$  **where**  
 $atc$ - $reduction\ M2\ s2\ M1\ s1\ \Omega = (\forall t \in \Omega . IO\ M2\ s2\ t \subseteq IO\ M1\ s1\ t)$

— r-distinguishability holds for atc-reductions

**lemma**  $atc$ - $rdist$ - $dist$ [ $intro$ ] :  
**assumes**  $wf2$  :  $well$ - $formed\ M2$   
**and**  $cs2$  :  $completely$ - $specified\ M2$   
**and**  $ap2$  :  $applicable$ - $set\ M2\ \Omega$   
**and**  $el$ - $t1$  :  $t1 \in nodes\ M2$   
**and**  $red1$  :  $atc$ - $reduction\ M2\ t1\ M1\ s1\ \Omega$

```

and   red2 : atc-reduction M2 t2 M1 s2 Ω
and   rdist : r-dist-set M1 Ω s1 s2
and   t1 ∈ nodes M2
shows r-dist-set M2 Ω t1 t2
proof -
  obtain td where td-def : td ∈ Ω ∧ r-dist M1 td s1 s2
    using rdist by auto
  then have IO M1 s1 td ∩ IO M1 s2 td = {}
    using td-def by simp
  moreover have IO M2 t1 td ⊆ IO M1 s1 td
    using red1 td-def by auto
  moreover have IO M2 t2 td ⊆ IO M1 s2 td
    using red2 td-def by auto
  ultimately have no-inter : IO M2 t1 td ∩ IO M2 t2 td = {}
    by blast

  then have td ≠ Leaf
    by auto
  then have IO M2 t1 td ≠ {}
    by (meson ap2 IO-applicable-nonempty applicable-set.elims(2) cs2 td-def assms(8))
  then have IO M2 t1 td ≠ IO M2 t2 td
    using no-inter by auto
  then show ?thesis
    using no-inter td-def by auto
qed

```

### 3.8 Reduction over ATCs applied after input sequences

The following functions check whether some FSM is a reduction of another over a given set of input sequences while furthermore the response sets obtained by applying a set of ATCs after every input sequence to the first FSM are subsets of the analogously constructed response sets of the second FSM.

```

fun atc-io-reduction-on :: ('in, 'out, 'state1) FSM ⇒ ('in, 'out, 'state2) FSM ⇒ 'in list
  ⇒ ('in, 'out) ATC set ⇒ bool where
  atc-io-reduction-on M1 M2 iseq Ω = (Lin M1 {iseq} ⊆ Lin M2 {iseq}
  ∧ (∀ io ∈ Lin M1 {iseq} . B M1 io Ω ⊆ B M2 io Ω))

```

```

fun atc-io-reduction-on-sets :: ('in, 'out, 'state1) FSM ⇒ 'in list set ⇒ ('in, 'out) ATC set
  ⇒ ('in, 'out, 'state2) FSM ⇒ bool where
  atc-io-reduction-on-sets M1 TS Ω M2 = (∀ iseq ∈ TS . atc-io-reduction-on M1 M2 iseq Ω)

```

**notation**

```

atc-io-reduction-on-sets (⟨(- ≤[-.-] -)⟩ [1000,1000,1000,1000])

```

**lemma** io-reduction-from-atc-io-reduction :

```

assumes atc-io-reduction-on-sets M1 T Ω M2
and   finite T
shows io-reduction-on M1 T M2
using assms(2,1) proof (induction T)
  case empty
  then show ?case by auto
next
  case (insert t T)
  then have atc-io-reduction-on M1 M2 t Ω
    by auto
  then have Lin M1 {t} ⊆ Lin M2 {t}
    using atc-io-reduction-on.simps by blast

  have Lin M1 T ⊆ Lin M2 T
    using insert.IH
  proof -
  have atc-io-reduction-on-sets M1 T Ω M2
    by (meson contra-subsetD insert.premis atc-io-reduction-on-sets.simps subset-insertI)
  then show ?thesis

```

using *insert.IH* by *blast*  
 qed  
 then have  $L_{in} M1 T \subseteq L_{in} M2 (insert\ t\ T)$   
 by (*meson insert-iff language-state-for-inputs-in-language-state*  
*language-state-for-inputs-map-fst language-state-for-inputs-map-fst-contained*  
*subsetCE subsetI*)  
 moreover have  $L_{in} M1 \{t\} \subseteq L_{in} M2 (insert\ t\ T)$   
 proof –  
 obtain  $pps :: ('a \times 'b) list\ set \Rightarrow ('a \times 'b) list\ set \Rightarrow ('a \times 'b) list$  **where**  
 $\forall x0\ x1. (\exists v2. v2 \in x1 \wedge v2 \notin x0) = (pps\ x0\ x1 \in x1 \wedge pps\ x0\ x1 \notin x0)$   
 by *moura*  
 then have  $\forall P\ Pa. pps\ Pa\ P \in P \wedge pps\ Pa\ P \notin Pa \vee P \subseteq Pa$   
 by *blast*  
 moreover  
 { **assume**  $map\ fst\ (pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ \{t\})) \notin insert\ t\ T$   
 then have  $pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ \{t\}) \notin L_{in}\ M1\ \{t\}$   
 $\vee pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ \{t\}) \in L_{in}\ M2\ (insert\ t\ T)$   
 by (*metis (no-types) insertI1 language-state-for-inputs-map-fst-contained singletonD*) }  
 ultimately show *?thesis*  
 by (*meson*  $\langle L_{in}\ M1\ \{t\} \subseteq L_{in}\ M2\ \{t\} \rangle$  *language-state-for-inputs-in-language-state*  
*language-state-for-inputs-map-fst set-rev-mp*)  
 qed

ultimately show *?case*  
 proof –  
 have  $f1: \forall ps\ P\ Pa. (ps::('a \times 'b) list) \notin P \vee \neg P \subseteq Pa \vee ps \in Pa$   
 by *blast*  
 obtain  $pps :: ('a \times 'b) list\ set \Rightarrow ('a \times 'b) list\ set \Rightarrow ('a \times 'b) list$  **where**  
 $\forall x0\ x1. (\exists v2. v2 \in x1 \wedge v2 \notin x0) = (pps\ x0\ x1 \in x1 \wedge pps\ x0\ x1 \notin x0)$   
 by *moura*  
 moreover  
 { **assume**  $pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T))$   
 $\notin L_{in}\ M1\ \{t\}$   
 moreover  
 { **assume**  $map\ fst\ (pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T)))$   
 $\notin \{t\}$   
 then have  $map\ fst\ (pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T)))$   
 $(L_{in}\ M1\ (insert\ t\ T)) \neq t$   
 by *blast*  
 then have  $pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T))$   
 $\notin L_{in}\ M1\ (insert\ t\ T)$   
 $\vee pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T))$   
 $\in L_{in}\ M2\ (insert\ t\ T)$   
 using  $f1$  by (*meson*  $\langle L_{in}\ M1\ T \subseteq L_{in}\ M2\ (insert\ t\ T) \rangle$   
*insertE language-state-for-inputs-in-language-state*  
*language-state-for-inputs-map-fst*  
*language-state-for-inputs-map-fst-contained*) }  
 ultimately have *io-reduction-on M1 (insert t T) M2*  
 $\vee pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T))$   
 $\notin L_{in}\ M1\ (insert\ t\ T)$   
 $\vee pps\ (L_{in}\ M2\ (insert\ t\ T))\ (L_{in}\ M1\ (insert\ t\ T))$   
 $\in L_{in}\ M2\ (insert\ t\ T)$   
 using  $f1$  by (*meson language-state-for-inputs-in-language-state*  
*language-state-for-inputs-map-fst*) }  
 ultimately show *?thesis*  
 using  $f1$  by (*meson*  $\langle L_{in}\ M1\ \{t\} \subseteq L_{in}\ M2\ (insert\ t\ T) \rangle$  *subsetI*)  
 qed  
 qed

lemma *atc-io-reduction-on-subset* :  
 assumes *atc-io-reduction-on-sets M1 T Ω M2*  
 and  $T' \subseteq T$   
 shows *atc-io-reduction-on-sets M1 T' Ω M2*  
 using *assms unfolding atc-io-reduction-on-sets.simps* by *blast*

```

lemma atc-reaction-reduction[intro] :
  assumes ls : language-state M1 q1  $\subseteq$  language-state M2 q2
  and el1 : q1  $\in$  nodes M1
  and el2 : q2  $\in$  nodes M2
  and rct : atc-reaction M1 q1 t io
  and ob2 : observable M2
  and ob1 : observable M1
shows atc-reaction M2 q2 t io
using assms proof (induction t arbitrary: io q1 q2)
  case Leaf
  then have io = []
    by (metis atc-reaction-nonempty-no-leaf list.exhaust)
  then show ?case
    by (simp add: leaf)
next
case (Node x f)
then obtain io-hd io-tl where io-split : io = io-hd # io-tl
  by (metis ATC.distinct(1) atc-reaction-empty list.exhaust)
moreover obtain y where y-def : io-hd = (x,y)
  using Node calculation by (metis ATC.inject atc-reaction-nonempty surj-pair)
ultimately obtain q1x where q1x-def : q1x  $\in$  succ M1 (x,y) q1 atc-reaction M1 q1x (f y) io-tl
  using Node.prem(4) by blast

then have pt1 : path M1 ((x,y) || [q1x]) q1
  by auto
then have ls1 : [(x,y)]  $\in$  language-state M1 q1
  unfolding language-state-def path-def using list.simps(9) by force
moreover have q1x  $\in$  io-targets M1 q1 [(x,y)]
  unfolding io-targets.simps
proof -
  have f1: length [(x, y)] = length [q1x]
    by simp
  have q1x = target ((x, y) || [q1x]) q1
    by simp
  then show q1x  $\in$  {target ((x, y) || cs) q1 | cs. path M1 ((x, y) || cs) q1
     $\wedge$  length [(x, y)] = length cs}
    using f1 pt1 by blast
qed
ultimately have tgt1 : io-targets M1 q1 [(x,y)] = {q1x}
  using Node.prem(5) io-targets-observable-singleton-ex q1x-def
  by (metis (no-types, lifting) singletonD)

then have ls2 : [(x,y)]  $\in$  language-state M2 q2
  using Node.prem(1) ls1 by auto
then obtain q2x where q2x-def : q2x  $\in$  succ M2 (x,y) q2
  unfolding language-state-def path-def
  using transition-system.path.cases by fastforce
then have pt2 : path M2 ((x,y) || [q2x]) q2
  by auto
then have q2x  $\in$  io-targets M2 q2 [(x,y)]
  using ls2 unfolding io-targets.simps
proof -
  have f1: length [(x, y)] = length [q2x]
    by simp
  have q2x = target ((x, y) || [q2x]) q2
    by simp
  then show q2x  $\in$  {target ((x, y) || cs) q2 | cs. path M2 ((x, y) || cs) q2
     $\wedge$  length [(x, y)] = length cs}
    using f1 pt2 by blast
qed
then have tgt2 : io-targets M2 q2 [(x,y)] = {q2x}
  using Node.prem(5) io-targets-observable-singleton-ex ls2 q2x-def
  by (metis (no-types, lifting) singletonD)

```

**then have** *language-state*  $M1$   $q1x \subseteq$  *language-state*  $M2$   $q2x$   
**using** *language-state-inclusion-of-state-reached-by-same-sequence*  
 $[of\ M1\ q1\ M2\ q2\ [(x,y)]\ q1x\ q2x]$   
 $tgt1\ tgt2\ Node.premis$  **by** *auto*  
**moreover have**  $q1x \in nodes\ M1$   
**using**  $q1x-def(1)\ Node.premis(2)$  **by** (*metis insertI1 io-targets-nodes tgt1*)  
**moreover have**  $q2x \in nodes\ M2$   
**using**  $q2x-def(1)\ Node.premis(3)$  **by** (*metis insertI1 io-targets-nodes tgt2*)  
**ultimately have**  $q2x \in succ\ M2\ (x,y)\ q2 \wedge atc-reaction\ M2\ q2x\ (f\ y)\ io-tl$   
**using**  $Node.IH[of\ f\ y\ q1x\ q2x\ io-tl]\ ob1\ ob2\ q1x-def(2)\ q2x-def$  **by** *blast*

**then show** *atc-reaction*  $M2$   $q2$  (*Node*  $x\ f$ ) *io* **using** *io-split y-def* **by** *blast*  
**qed**

**lemma** *IO-reduction* :

**assumes**  $ls : language-state\ M1\ q1 \subseteq language-state\ M2\ q2$   
**and**  $el1 : q1 \in nodes\ M1$   
**and**  $el2 : q2 \in nodes\ M2$   
**and**  $ob1 : observable\ M1$   
**and**  $ob2 : observable\ M2$   
**shows**  $IO\ M1\ q1\ t \subseteq IO\ M2\ q2\ t$   
**using** *assms atc-reaction-reduction unfolding IO.simps* **by** *auto*

**lemma** *IO-set-reduction* :

**assumes**  $ls : language-state\ M1\ q1 \subseteq language-state\ M2\ q2$   
**and**  $el1 : q1 \in nodes\ M1$   
**and**  $el2 : q2 \in nodes\ M2$   
**and**  $ob1 : observable\ M1$   
**and**  $ob2 : observable\ M2$   
**shows**  $IO-set\ M1\ q1\ \Omega \subseteq IO-set\ M2\ q2\ \Omega$   
**proof** –  
**have**  $\forall t \in \Omega . IO\ M1\ q1\ t \subseteq IO\ M2\ q2\ t$   
**using** *assms IO-reduction* **by** *metis*  
**then show** *?thesis*  
**unfolding** *IO-set.simps* **by** *blast*  
**qed**

**lemma** *B-reduction* :

**assumes**  $red : M1 \preceq M2$   
**and**  $ob1 : observable\ M1$   
**and**  $ob2 : observable\ M2$   
**shows**  $B\ M1\ io\ \Omega \subseteq B\ M2\ io\ \Omega$   
**proof**  
**fix**  $xy$  **assume**  $xy-asm : xy \in B\ M1\ io\ \Omega$   
**then obtain**  $q1x$  **where**  $q1x-def : q1x \in (io-targets\ M1\ (initial\ M1)\ io) \wedge xy \in IO-set\ M1\ q1x\ \Omega$   
**unfolding** *B.simps* **by** *auto*  
**then obtain**  $tr1$  **where**  $tr1-def : path\ M1\ (io\ ||\ tr1)\ (initial\ M1) \wedge length\ io = length\ tr1$   
**by** *auto*  
  
**then have**  $q1x-ob : io-targets\ M1\ (initial\ M1)\ io = \{q1x\}$   
**using** *assms*  
**by** (*metis io-targets-observable-singleton-ex language-state q1x-def singleton-iff*)  
  
**then have**  $ls1 : io \in language-state\ M1\ (initial\ M1)$   
**by** *auto*  
**then have**  $ls2 : io \in language-state\ M2\ (initial\ M2)$   
**using** *red* **by** *auto*  
  
**then obtain**  $tr2$  **where**  $tr2-def : path\ M2\ (io\ ||\ tr2)\ (initial\ M2) \wedge length\ io = length\ tr2$   
**by** *auto*  
**then obtain**  $q2x$  **where**  $q2x-def : q2x \in (io-targets\ M2\ (initial\ M2)\ io)$   
**by** *auto*

**then have**  $q2x\text{-ob} : \text{io-targets } M2 \text{ (initial } M2) \text{ io} = \{q2x\}$   
**using**  $tr2\text{-def } \text{assms}$   
**by** ( $\text{metis io-targets-observable-singleton-ex language-state singleton-iff}$ )

**then have**  $\text{language-state } M1 \ q1x \subseteq \text{language-state } M2 \ q2x$   
**using**  $q1x\text{-ob } \text{assms}$  **unfolding**  $\text{io-reduction.simps}$   
**by** ( $\text{simp add: language-state-inclusion-of-state-reached-by-same-sequence}$ )

**then have**  $\text{IO-set } M1 \ q1x \ \Omega \subseteq \text{IO-set } M2 \ q2x \ \Omega$   
**using**  $\text{assms IO-set-reduction}$  **by** ( $\text{metis FSM.nodes.initial io-targets-nodes } q1x\text{-def } q2x\text{-def}$ )

**moreover have**  $B \ M1 \ \text{io} \ \Omega = \text{IO-set } M1 \ q1x \ \Omega$   
**using**  $q1x\text{-ob}$  **by**  $\text{auto}$

**moreover have**  $B \ M2 \ \text{io} \ \Omega = \text{IO-set } M2 \ q2x \ \Omega$   
**using**  $q2x\text{-ob}$  **by**  $\text{auto}$

**ultimately have**  $B \ M1 \ \text{io} \ \Omega \subseteq B \ M2 \ \text{io} \ \Omega$   
**by**  $\text{simp}$

**then show**  $xy \in B \ M2 \ \text{io} \ \Omega$  **using**  $xy\text{-assm}$   
**by**  $\text{blast}$

**qed**

**lemma**  $\text{append-io-B-reduction} :$   
**assumes**  $\text{red} : M1 \preceq M2$   
**and**  $ob1 : \text{observable } M1$   
**and**  $ob2 : \text{observable } M2$   
**shows**  $\text{append-io-B } M1 \ \text{io} \ \Omega \subseteq \text{append-io-B } M2 \ \text{io} \ \Omega$   
**proof**  
**fix**  $ioR$  **assume**  $ioR\text{-assm} : ioR \in \text{append-io-B } M1 \ \text{io} \ \Omega$   
**then obtain**  $res$  **where**  $res\text{-def} : ioR = \text{io} @ res$   $res \in B \ M1 \ \text{io} \ \Omega$   
**by**  $\text{auto}$   
**then have**  $res \in B \ M2 \ \text{io} \ \Omega$   
**using**  $\text{assms B-reduction}$  **by** ( $\text{metis (no-types, opaque-lifting) subset-iff}$ )  
**then show**  $ioR \in \text{append-io-B } M2 \ \text{io} \ \Omega$   
**using**  $ioR\text{-assm } res\text{-def}$  **by**  $\text{auto}$

**qed**

**lemma**  $\text{atc-io-reduction-on-reduction[intro]} :$   
**assumes**  $\text{red} : M1 \preceq M2$   
**and**  $ob1 : \text{observable } M1$   
**and**  $ob2 : \text{observable } M2$   
**shows**  $\text{atc-io-reduction-on } M1 \ M2 \ \text{iseq} \ \Omega$   
**unfolding**  $\text{atc-io-reduction-on.simps}$  **proof**  
**show**  $L_{in} \ M1 \ \{\text{iseq}\} \subseteq L_{in} \ M2 \ \{\text{iseq}\}$   
**using**  $\text{red}$  **by**  $\text{auto}$

**next**  
**show**  $\forall io \in L_{in} \ M1 \ \{\text{iseq}\}. B \ M1 \ \text{io} \ \Omega \subseteq B \ M2 \ \text{io} \ \Omega$   
**using**  $B\text{-reduction } \text{assms}$  **by**  $\text{blast}$

**qed**

**lemma**  $\text{atc-io-reduction-on-sets-reduction[intro]} :$   
**assumes**  $\text{red} : M1 \preceq M2$   
**and**  $ob1 : \text{observable } M1$   
**and**  $ob2 : \text{observable } M2$   
**shows**  $\text{atc-io-reduction-on-sets } M1 \ TS \ \Omega \ M2$   
**using**  $\text{assms atc-io-reduction-on-reduction}$  **by** ( $\text{metis atc-io-reduction-on-sets.elims(3)}$ )

**lemma**  $\text{atc-io-reduction-on-sets-via-LS}_{in} :$   
**assumes**  $\text{atc-io-reduction-on-sets } M1 \ TS \ \Omega \ M2$   
**shows**  $(L_{in} \ M1 \ TS \cup (\bigcup_{io \in L_{in} \ M1 \ TS}. B \ M1 \ \text{io} \ \Omega))$   
 $\subseteq (L_{in} \ M2 \ TS \cup (\bigcup_{io \in L_{in} \ M2 \ TS}. B \ M2 \ \text{io} \ \Omega))$

**proof** –  
**have**  $\forall \text{iseq} \in TS. (L_{in} \ M1 \ \{\text{iseq}\} \subseteq L_{in} \ M2 \ \{\text{iseq}\})$   
 $\wedge (\forall io \in L_{in} \ M1 \ \{\text{iseq}\}. B \ M1 \ \text{io} \ \Omega \subseteq B \ M2 \ \text{io} \ \Omega)$

```

    using assms by auto
  then have  $\forall \text{iseq} \in TS . (\bigcup_{io \in L_{in}} M1 \{iseq\}. B M1 io \Omega)$ 
            $\subseteq (\bigcup_{io \in L_{in}} M2 \{iseq\}. B M2 io \Omega)$ 

    by blast
  moreover have  $\forall \text{iseq} \in TS . (\bigcup_{io \in L_{in}} M2 \{iseq\}. B M2 io \Omega)$ 
            $\subseteq (\bigcup_{io \in L_{in}} M2 TS. B M2 io \Omega)$ 

    unfolding language-state-for-inputs.simps by blast
  ultimately have elem-subset :  $\forall \text{iseq} \in TS .$ 
            $(\bigcup_{io \in L_{in}} M1 \{iseq\}. B M1 io \Omega)$ 
            $\subseteq (\bigcup_{io \in L_{in}} M2 TS. B M2 io \Omega)$ 

    by blast

show ?thesis
proof
  fix x assume  $x \in L_{in} M1 TS \cup (\bigcup_{io \in L_{in}} M1 TS. B M1 io \Omega)$ 
  then show  $x \in L_{in} M2 TS \cup (\bigcup_{io \in L_{in}} M2 TS. B M2 io \Omega)$ 
  proof (cases  $x \in L_{in} M1 TS$ )
    case True
    then obtain iseq where  $iseq \in TS \ x \in L_{in} M1 \{iseq\}$ 
    unfolding language-state-for-inputs.simps by blast
    then have atc-io-reduction-on  $M1 M2 \text{iseq} \Omega$ 
    using assms by auto
    then have  $L_{in} M1 \{iseq\} \subseteq L_{in} M2 \{iseq\}$ 
    by auto
    then have  $x \in L_{in} M2 TS$ 
    by (metis (no-types, lifting) UN-I
       $\langle \bigwedge \text{thesis}. (\bigwedge \text{iseq}. \llbracket \text{iseq} \in TS; x \in L_{in} M1 \{iseq\} \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$ 
       $\langle \forall \text{iseq} \in TS. L_{in} M1 \{iseq\} \subseteq L_{in} M2 \{iseq\} \wedge (\forall io \in L_{in} M1 \{iseq\}. B M1 io \Omega \subseteq B M2 io \Omega) \rangle$ 
      language-state-for-input-alt-def language-state-for-inputs-alt-def set-rev-mp)
    then show ?thesis
    by blast
  next
  case False
  then have  $x \in (\bigcup_{io \in L_{in}} M1 TS. B M1 io \Omega)$ 
  using  $\langle x \in L_{in} M1 TS \cup (\bigcup_{io \in L_{in}} M1 TS. B M1 io \Omega) \rangle$  by blast
  then obtain io where  $io \in L_{in} M1 TS \ x \in B M1 io \Omega$ 
  by blast
  then obtain iseq where  $iseq \in TS \ io \in L_{in} M1 \{iseq\}$ 
  unfolding language-state-for-inputs.simps by blast
  have  $x \in (\bigcup_{io \in L_{in}} M1 \{iseq\}. B M1 io \Omega)$ 
  using  $\langle io \in L_{in} M1 \{iseq\} \rangle \langle x \in B M1 io \Omega \rangle$  by blast
  then have  $x \in (\bigcup_{io \in L_{in}} M2 TS. B M2 io \Omega)$ 
  using  $\langle \text{iseq} \in TS \rangle$  elem-subset by blast
  then show ?thesis
  by blast
qed
qed
qed

end
theory ASC-LB
imports ../ATC/ATC ../FSM/FSM-Product
begin

```

## 4 The lower bound function

This theory defines the lower bound function **LB** and its properties.

Function **LB** calculates a lower bound on the number of states of some FSM in order for some sequence to not contain certain repetitions.

## 4.1 Permutation function Perm

Function `Perm` calculates all possible reactions of an FSM to a set of inputs sequences such that every set in the calculated set of reactions contains exactly one reaction for each input sequence.

```
fun Perm :: 'in list set  $\Rightarrow$  ('in, 'out, 'state) FSM  $\Rightarrow$  ('in  $\times$  'out) list set set where
  Perm V M = {image f V | f .  $\forall v \in V . f v \in \text{language-state-for-input } M \text{ (initial } M) v$  }
```

**lemma** *perm-empty* :

**assumes** *is-det-state-cover* M2 V

**and**  $V'' \in \text{Perm } V M1$

**shows**  $\square \in V''$

**proof** –

**have** *init-seq* :  $\square \in V$  **using** *det-state-cover-empty* *assms* **by** *simp*

**obtain** f **where** *f-def* :  $V'' = \text{image } f V$

$\wedge (\forall v \in V . f v \in \text{language-state-for-input } M1 \text{ (initial } M1) v)$

**using** *assms* **by** *auto*

**then** **have**  $f \square = \square$

**using** *init-seq* **by** (*metis language-state-for-input-empty singleton-iff*)

**then** **show** *?thesis*

**using** *init-seq f-def* **by** (*metis image-eqI*)

**qed**

**lemma** *perm-elem-finite* :

**assumes** *is-det-state-cover* M2 V

**and** *well-formed* M2

**and**  $V'' \in \text{Perm } V M1$

**shows** *finite*  $V''$

**proof** –

**obtain** f **where** *is-det-state-cover-ass* M2 f  $\wedge V = f' d\text{-reachable } M2 \text{ (initial } M2)$

**using** *assms* **by** *auto*

**moreover** **have** *finite* (*d-reachable* M2 (*initial* M2))

**proof** –

**have** *finite* (*nodes* M2)

**using** *assms* **by** *auto*

**moreover** **have** *nodes* M2 = *reachable* M2 (*initial* M2)

**by** *auto*

**ultimately** **have** *finite* (*reachable* M2 (*initial* M2))

**by** *simp*

**moreover** **have** *d-reachable* M2 (*initial* M2)  $\subseteq$  *reachable* M2 (*initial* M2)

**by** *auto*

**ultimately** **show** *?thesis*

**using** *infinite-super* **by** *blast*

**qed**

**ultimately** **have** *finite* V

**by** *auto*

**moreover** **obtain**  $f''$  **where**  $V'' = \text{image } f'' V$

$\wedge (\forall v \in V . f'' v \in \text{language-state-for-input } M1 \text{ (initial } M1) v)$

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

**ultimately** **show** *?thesis*

**by** *simp*

**qed**

**lemma** *perm-inputs* :

**assumes**  $V'' \in \text{Perm } V M$

**and**  $vs \in V''$

**shows** *map fst*  $vs \in V$

**proof** –

**obtain** f **where** *f-def* :  $V'' = \text{image } f V$

$\wedge (\forall v \in V . f v \in \text{language-state-for-input } M \text{ (initial } M) v)$

**using** *assms* **by** *auto*

**then** **obtain** v **where** *v-def* :  $v \in V \wedge f v = vs$

**using** *assms* **by** *auto*

**then** **have**  $vs \in \text{language-state-for-input } M \text{ (initial } M) v$

**using** *f-def* **by** *auto*

**then** **show** *?thesis*

**using** *v-def* **unfolding** *language-state-for-input.simps* **by** *auto*

qed

**lemma** *perm-inputs-diff* :

**assumes**  $V'' \in \text{Perm } V M$

**and**  $vs1 \in V''$

**and**  $vs2 \in V''$

**and**  $vs1 \neq vs2$

**shows**  $\text{map fst } vs1 \neq \text{map fst } vs2$

**proof** –

**obtain**  $f$  **where**  $f\text{-def} : V'' = \text{image } f V$

$\wedge (\forall v \in V . f v \in \text{language-state-for-input } M (\text{initial } M) v)$

**using** *assms* **by** *auto*

**then obtain**  $v1 v2$  **where**  $v\text{-def} : v1 \in V \wedge f v1 = vs1 \wedge v2 \in V \wedge f v2 = vs2$

**using** *assms* **by** *auto*

**then have**  $vs1 \in \text{language-state-for-input } M (\text{initial } M) v1$

$vs2 \in \text{language-state-for-input } M (\text{initial } M) v2$

**using**  $f\text{-def}$  **by** *auto*

**moreover have**  $v1 \neq v2$

**using**  $v\text{-def}$  *assms*(4) **by** *blast*

**ultimately show** *?thesis*

**by** *auto*

qed

**lemma** *perm-language* :

**assumes**  $V'' \in \text{Perm } V M$

**and**  $vs \in V''$

**shows**  $vs \in L M$

**proof** –

**obtain**  $f$  **where**  $f\text{-def} : \text{image } f V = V''$

$\wedge (\forall v \in V . f v \in \text{language-state-for-input } M (\text{initial } M) v)$

**using** *assms*(1) **by** *auto*

**then have**  $\exists v . f v = vs \wedge f v \in \text{language-state-for-input } M (\text{initial } M) v$

**using** *assms*(2) **by** *blast*

**then show** *?thesis*

**by** *auto*

qed

## 4.2 Helper predicates

The following predicates are used to combine often repeated assumption.

**abbreviation** *asc-fault-domain*  $M2 M1 m \equiv (\text{inputs } M2 = \text{inputs } M1 \wedge \text{card } (\text{nodes } M1) \leq m)$

**lemma** *asc-fault-domain-props*[*elim!*] :

**assumes** *asc-fault-domain*  $M2 M1 m$

**shows**  $\text{inputs } M2 = \text{inputs } M1$

$\text{card } (\text{nodes } M1) \leq$  **using** *assms* **by** *auto*

**abbreviation**

*test-tools*  $M2 M1 \text{ FAIL } PM V \Omega \equiv ($

$\text{productF } M2 M1 \text{ FAIL } PM$

$\wedge \text{is-det-state-cover } M2 V$

$\wedge \text{applicable-set } M2 \Omega$

)

**lemma** *test-tools-props*[*elim*] :

**assumes** *test-tools*  $M2 M1 \text{ FAIL } PM V \Omega$

**and** *asc-fault-domain*  $M2 M1 m$

**shows**  $\text{productF } M2 M1 \text{ FAIL } PM$

$\text{is-det-state-cover } M2 V$

$\text{applicable-set } M2 \Omega$

$\text{applicable-set } M1 \Omega$

**proof** –

**show**  $\text{productF } M2 M1 \text{ FAIL } PM$  **using** *assms*(1) **by** *blast*

**show**  $\text{is-det-state-cover } M2 V$  **using** *assms*(1) **by** *blast*

**show**  $\text{applicable-set } M2 \Omega$  **using** *assms*(1) **by** *blast*

**then show**  $\text{applicable-set } M1 \Omega$

```

unfolding applicable-set.simps applicable.simps
using asc-fault-domain-props(1)[OF assms(2)] by simp
qed

```

```

lemma perm-nonempty :
  assumes is-det-state-cover M2 V
  and OFSM M1
  and OFSM M2
  and inputs M1 = inputs M2
shows Perm V M1 ≠ {}
proof –
  have finite (nodes M2)
    using assms(3) by auto
  moreover have d-reachable M2 (initial M2) ⊆ nodes M2
    by auto
  ultimately have finite V
    using det-state-cover-card[OF assms(1)]
    by (metis assms(1) finite-imageI infinite-super is-det-state-cover.elims(2))

  have  $\square \in V$ 
    using assms(1) det-state-cover-empty by blast

```

```

have  $\bigwedge VS . VS \subseteq V \wedge VS \neq \{\} \implies Perm VS M1 \neq \{\}$ 

```

```

proof –
  fix VS assume VS ⊆ V ∧ VS ≠ {}
  then have finite VS using  $\langle finite V \rangle$ 
    using infinite-subset by auto
  then show Perm VS M1 ≠ {}
    using  $\langle VS \subseteq V \wedge VS \neq \{\} \rangle \langle finite VS \rangle$ 
  proof (induction VS)
    case empty
    then show ?case by auto
  next
    case (insert vs F)
    then have vs ∈ V by blast

    obtain q2 where d-reaches M2 (initial M2) vs q2
      using det-state-cover-d-reachable[OF assms(1) ⟨vs ∈ V⟩] by blast
    then obtain vs' vsP where io-path : length vs = length vs'
       $\wedge$  length vs = length vsP
       $\wedge$  (path M2 ((vs || vs') || vsP) (initial M2))
       $\wedge$  target ((vs || vs') || vsP) (initial M2) = q2

    by auto

```

```

have well-formed M2
  using assms by auto

```

```

have map fst (map fst ((vs || vs') || vsP)) = vs

```

```

proof –
  have length (vs || vs') = length vsP
    using io-path by simp
  then show ?thesis
    using io-path by auto

```

```

qed

```

```

moreover have set (map fst (map fst ((vs || vs') || vsP))) ⊆ inputs M2
  using path-input-containment[OF ⟨well-formed M2⟩, of (vs || vs') || vsP initial M2]
  io-path
  by linarith
ultimately have set vs ⊆ inputs M2
  by presburger

```

```

then have set vs ⊆ inputs M1

```

```

using assms by auto

then have  $L_{in} M1 \{vs\} \neq \{\}$ 
  using assms(2) language-state-for-inputs-nonempty
  by (metis FSM.nodes.initial)
then have language-state-for-input M1 (initial M1) vs  $\neq \{\}$ 
  by auto
then obtain vs' where  $vs' \in \text{language-state-for-input } M1 \text{ (initial } M1) \text{ vs}$ 
  by blast

show ?case
proof (cases  $F = \{\}$ )
  case True
  moreover obtain f where  $f \text{ vs} = vs'$ 
  by force
  ultimately have  $\text{image } f \text{ (insert vs } F) \in \text{Perm (insert vs } F) M1$ 
  using Perm.simps  $\langle vs' \in \text{language-state-for-input } M1 \text{ (initial } M1) \text{ vs} \rangle$  by blast
  then show ?thesis by blast
next
  case False
  then obtain  $F''$  where  $F'' \in \text{Perm } F M1$ 
  using insert.IH insert.hyps(1) insert.prems(1) by blast
  then obtain f where  $F'' = \text{image } f F$ 
    ( $\forall v \in F . f v \in \text{language-state-for-input } M1 \text{ (initial } M1) v$ )
  by auto
  let ?f =  $f \text{ vs} := vs^\wedge$ 
  have  $\forall v \in (\text{insert vs } F) . ?f v \in \text{language-state-for-input } M1 \text{ (initial } M1) v$ 
  proof
    fix v assume  $v \in \text{insert vs } F$ 
    then show ?f  $v \in \text{language-state-for-input } M1 \text{ (initial } M1) v$ 
    proof (cases  $v = vs$ )
      case True
      then show ?thesis
      using  $\langle vs' \in \text{language-state-for-input } M1 \text{ (initial } M1) \text{ vs} \rangle$  by auto
    next
      case False
      then have  $v \in F$ 
      using  $\langle v \in \text{insert vs } F \rangle$  by blast
      then show ?thesis
      using False  $\langle \forall v \in F. f v \in \text{language-state-for-input } M1 \text{ (initial } M1) v \rangle$  by auto
    qed
  qed
  then have  $\text{image } ?f \text{ (insert vs } F) \in \text{Perm (insert vs } F) M1$ 
  using Perm.simps by blast
  then show ?thesis
  by blast
qed
qed
qed
qed

then show ?thesis
  using  $\langle [] \in V \rangle$  by blast
qed

```

```

lemma perm-lem :
  assumes is-det-state-cover M2 V
  and OFSM M1
  and OFSM M2
  and inputs M1 = inputs M2
  and  $vs \in V$ 
  and  $vs' \in \text{language-state-for-input } M1 \text{ (initial } M1) \text{ vs}$ 
obtains  $V''$ 
where  $V'' \in \text{Perm } V M1 \text{ vs}' \in V''$ 
proof -

```

```

obtain  $V''$  where  $V'' \in \text{Perm } V M1$ 
  using perm-nonempty[OF assms(1-4)] by blast
then obtain  $f$  where  $V'' = \text{image } f V$ 
   $(\forall v \in V . f v \in \text{language-state-for-input } M1 \text{ (initial } M1) v)$ 
  by auto

let  $?f = f(vs := vs')$ 

have  $\forall v \in V . (?f v) \in (\text{language-state-for-input } M1 \text{ (initial } M1) v)$ 
  using  $\langle \forall v \in V . (f v) \in (\text{language-state-for-input } M1 \text{ (initial } M1) v) \rangle$  assms(6) by fastforce

then have  $(\text{image } ?f V) \in \text{Perm } V M1$ 
  unfolding Perm.simps by blast
moreover have  $vs' \in \text{image } ?f V$ 
  by (metis assms(5) fun-upd-same imageI)
ultimately show ?thesis
  using that by blast
qed

```

### 4.3 Function R

Function R calculates the set of suffixes of a sequence that reach a given state if applied after a given other sequence.

```

fun  $R :: ('in, 'out, 'state) \text{FSM} \Rightarrow 'state \Rightarrow ('in \times 'out) \text{list}$ 
   $\Rightarrow ('in \times 'out) \text{list} \Rightarrow ('in \times 'out) \text{list set}$ 
  where
   $R M s vs xs = \{ vs @ xs' \mid xs' . xs' \neq []$ 
     $\wedge \text{prefix } xs' xs$ 
     $\wedge s \in \text{io-targets } M \text{ (initial } M) (vs @ xs') \}$ 

```

**lemma** *finite-R* : *finite* ( $R M s vs xs$ )

```

proof –
  have  $R M s vs xs \subseteq \{ vs @ xs' \mid xs' . \text{prefix } xs' xs \}$ 
  by auto
  then have  $R M s vs xs \subseteq \text{image } (\lambda xs' . vs @ xs') \{ xs' . \text{prefix } xs' xs \}$ 
  by auto
  moreover have  $\{ xs' . \text{prefix } xs' xs \} = \{ \text{take } n \ xs \mid n . n \leq \text{length } xs \}$ 
  proof
    show  $\{ xs' . \text{prefix } xs' xs \} \subseteq \{ \text{take } n \ xs \mid n . n \leq \text{length } xs \}$ 
    proof
      fix  $xs'$  assume  $xs' \in \{ xs' . \text{prefix } xs' xs \}$ 
      then obtain  $zs'$  where  $xs' @ zs' = xs$ 
      by (metis (full-types) mem-Collect-eq prefixE)
      then obtain  $i$  where  $xs' = \text{take } i \ xs \wedge i \leq \text{length } xs$ 
      by (metis (full-types) append-eq-conv-conj le-cases take-all)
      then show  $xs' \in \{ \text{take } n \ xs \mid n . n \leq \text{length } xs \}$ 
      by auto
    qed
    show  $\{ \text{take } n \ xs \mid n . n \leq \text{length } xs \} \subseteq \{ xs' . \text{prefix } xs' xs \}$ 
    using take-is-prefix by force
  qed
  moreover have finite  $\{ \text{take } n \ xs \mid n . n \leq \text{length } xs \}$ 
  by auto
  ultimately show ?thesis
  by auto
qed

```

**lemma** *card-union-of-singletons* :

```

  assumes  $\forall S \in SS . (\exists t . S = \{t\})$ 
shows  $\text{card } (\bigcup SS) = \text{card } SS$ 
proof –
  let  $?f = \lambda x . \{x\}$ 
  have bij-betw  $?f (\bigcup SS) SS$ 

```

**unfolding** *bij-betw-def inj-on-def* **using** *assms* **by** *fastforce*  
**then show** *?thesis*  
**using** *bij-betw-same-card* **by** *blast*  
**qed**

**lemma** *card-union-of-distinct* :  
**assumes**  $\forall S1 \in SS . \forall S2 \in SS . S1 = S2 \vee f S1 \cap f S2 = \{\}$   
**and** *finite SS*  
**and**  $\forall S \in SS . f S \neq \{\}$   
**shows** *card (image f SS) = card SS*  
**proof** –  
**from** *assms(2)* **have**  $\forall S1 \in SS . \forall S2 \in SS . S1 = S2 \vee f S1 \cap f S2 = \{\}$   
 $\implies \forall S \in SS . f S \neq \{\} \implies ?thesis$   
**proof** (*induction SS*)  
**case** *empty*  
**then show** *?case by auto*  
**next**  
**case** (*insert x F*)  
**then have**  $\neg (\exists y \in F . f y = f x)$   
**by** *auto*  
**then have**  $f x \notin \text{image } f F$   
**by** *auto*  
**then have** *card (image f (insert x F)) = Suc (card (image f F))*  
**using** *insert by auto*  
**moreover have** *card (f ` F) = card F*  
**using** *insert by auto*  
**moreover have** *card (insert x F) = Suc (card F)*  
**using** *insert by auto*  
**ultimately show** *?case*  
**by** *simp*  
**qed**  
**then show** *?thesis*  
**using** *assms* **by** *simp*  
**qed**

**lemma** *R-count* :  
**assumes**  $(vs @ xs) \in L M1 \cap L M2$   
**and** *observable M1*  
**and** *observable M2*  
**and** *well-formed M1*  
**and** *well-formed M2*  
**and**  $s \in \text{nodes } M2$   
**and** *productF M2 M1 FAIL PM*  
**and** *io-targets PM (initial PM) vs = {(q2,q1)}*  
**and** *path PM (xs || tr) (q2,q1)*  
**and** *length xs = length tr*  
**and** *distinct (states (xs || tr) (q2,q1))*  
**shows** *card ( $\bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (R M2 s vs xs)) = \text{card } (R M2 s vs xs)$*   
— each sequence in the set calculated by R reaches a different state in M1  
**proof** –

— Proof sketch: - states of PM reached by the sequences calculated by R can differ only in their second value - the sequences in the set calculated by R reach different states in PM due to distinctness

**have** *obs-PM : observable PM* **using** *observable-productF assms(2) assms(3) assms(7)* **by** *blast*

**have** *state-component-2 :  $\forall io \in (R M2 s vs xs) . \text{io-targets } M2 (\text{initial } M2) io = \{s\}$*

**proof**  
**fix** *io* **assume**  $io \in R M2 s vs xs$   
**then have**  $s \in \text{io-targets } M2 (\text{initial } M2) io$   
**by** *auto*  
**moreover have**  $io \in \text{language-state } M2 (\text{initial } M2)$   
**using** *calculation by auto*  
**ultimately show**  $\text{io-targets } M2 (\text{initial } M2) io = \{s\}$   
**using** *assms(3) io-targets-observable-singleton-ex* **by** (*metis singletonD*)  
**qed**

**moreover have**  $\forall io \in R\ M2\ s\ vs\ xs . io\text{-targets}\ PM\ (initial\ PM)\ io$   
 $= io\text{-targets}\ M2\ (initial\ M2)\ io \times io\text{-targets}\ M1\ (initial\ M1)\ io$

**proof**

**fix**  $io$  **assume**  $io\text{-assm} : io \in R\ M2\ s\ vs\ xs$

**then have**  $io\text{-prefix} : prefix\ io\ (vs\ @\ xs)$

**by**  $auto$

**then have**  $io\text{-lang}\text{-subs} : io \in L\ M1 \wedge io \in L\ M2$

**using**  $assms(1)$  **unfolding**  $prefix\text{-def}$  **by**  $(metis\ IntE\ language\text{-state}\ language\text{-state}\text{-split})$

**then have**  $io\text{-lang}\text{-inter} : io \in L\ M1 \cap L\ M2$

**by**  $simp$

**then have**  $io\text{-lang}\text{-pm} : io \in L\ PM$

**using**  $productF\text{-language}\ assms$  **by**  $blast$

**moreover obtain**  $p2\ p1$  **where**  $(p2, p1) \in io\text{-targets}\ PM\ (initial\ PM)\ io$

**by**  $(metis\ assms(2)\ assms(3)\ assms(7)\ calculation\ insert\text{-absorb}\ insert\text{-ident}\ insert\text{-not}\text{-empty}\ io\text{-targets}\text{-observable}\text{-singleton}\text{-ob}\ observable\text{-productF}\ singleton\text{-insert}\text{-inj}\text{-eq}\ subrelI)$

**ultimately have**  $targets\text{-pm} : io\text{-targets}\ PM\ (initial\ PM)\ io = \{(p2, p1)\}$

**using**  $assms\ io\text{-targets}\text{-observable}\text{-singleton}\text{-ex}\ singletonD$  **by**  $(metis\ observable\text{-productF})$

**then obtain**  $trP$  **where**  $trP\text{-def} : target\ (io\ ||\ trP)\ (initial\ PM) = (p2, p1)$

$\wedge path\ PM\ (io\ ||\ trP)\ (initial\ PM)$

$\wedge length\ io = length\ trP$

**proof** –

**assume**  $a1 : \bigwedge trP. target\ (io\ ||\ trP)\ (initial\ PM) = (p2, p1)$

$\wedge path\ PM\ (io\ ||\ trP)\ (initial\ PM)$

$\wedge length\ io = length\ trP \implies thesis$

**have**  $\exists ps. target\ (io\ ||\ ps)\ (initial\ PM) = (p2, p1)$

$\wedge path\ PM\ (io\ ||\ ps)\ (initial\ PM) \wedge length\ io = length\ ps$

**using**  $\langle (p2, p1) \in io\text{-targets}\ PM\ (initial\ PM)\ io \rangle$  **by**  $auto$

**then show**  $?thesis$

**using**  $a1$  **by**  $blast$

**qed**

**then have**  $trP\text{-unique} : \{tr . path\ PM\ (io\ ||\ tr)\ (initial\ PM) \wedge length\ io = length\ tr\}$

$= \{trP\}$

**using**  $observable\text{-productF}\ observable\text{-path}\text{-unique}\text{-ex}[of\ PM\ io\ initial\ PM]$

$io\text{-lang}\text{-pm}\ assms(2)\ assms(3)\ assms(7)$

**proof** –

**obtain**  $pps :: ('d \times 'c)$  **list** **where**

$f1 : \{ps. path\ PM\ (io\ ||\ ps)\ (initial\ PM) \wedge length\ io = length\ ps\} = \{pps\}$

$\vee \neg observable\ PM$

**by**  $(metis\ (no\text{-types})\ \langle \bigwedge thesis. \llbracket observable\ PM; io \in L\ PM; \bigwedge tr.$

$\{t. path\ PM\ (io\ ||\ t)\ (initial\ PM)$

$\wedge length\ io = length\ t\} = \{tr\} \implies thesis \rrbracket \implies thesis)$

$io\text{-lang}\text{-pm})$

**have**  $f2 : observable\ PM$

**by**  $(meson\ \langle observable\ M1 \rangle\ \langle observable\ M2 \rangle\ \langle productF\ M2\ M1\ FAIL\ PM \rangle\ observable\text{-productF})$

**then have**  $trP \in \{pps\}$

**using**  $f1\ trP\text{-def}$  **by**  $blast$

**then show**  $?thesis$

**using**  $f2\ f1$  **by**  $force$

**qed**

**obtain**  $trIO2$  **where**  $trIO2\text{-def} : \{tr . path\ M2\ (io||tr)\ (initial\ M2) \wedge length\ io = length\ tr\}$   
 $= \{trIO2\}$

**using**  $observable\text{-path}\text{-unique}\text{-ex}[of\ M2\ io\ initial\ M2]\ io\text{-lang}\text{-subs}\ assms(3)$  **by**  $blast$

**obtain**  $trIO1$  **where**  $trIO1\text{-def} : \{tr . path\ M1\ (io||tr)\ (initial\ M1) \wedge length\ io = length\ tr\}$   
 $= \{trIO1\}$

**using**  $observable\text{-path}\text{-unique}\text{-ex}[of\ M1\ io\ initial\ M1]\ io\text{-lang}\text{-subs}\ assms(2)$  **by**  $blast$

**have**  $path\ PM\ (io\ ||\ trIO2\ ||\ trIO1)\ (initial\ M2, initial\ M1)$

$\wedge length\ io = length\ trIO2$

$\wedge length\ trIO2 = length\ trIO1$

**proof** –

**have**  $f1 : path\ M2\ (io\ ||\ trIO2)\ (initial\ M2) \wedge length\ io = length\ trIO2$

**using**  $trIO2\text{-def}$  **by**  $auto$

**have**  $f2 : path\ M1\ (io\ ||\ trIO1)\ (initial\ M1) \wedge length\ io = length\ trIO1$

```

    using trIO1-def by auto
  then have length trIO2 = length trIO1
    using f1 by presburger
  then show ?thesis
    using f2 f1 assms(4) assms(5) assms(7) by blast
qed
then have trP-split : path PM (io || trIO2 || trIO1) (initial PM)
  ∧ length io = length trIO2
  ∧ length trIO2 = length trIO1
  using assms(7) by auto
then have trP-zip : trIO2 || trIO1 = trP
  using trP-def trP-unique using length-zip by fastforce

have target (io || trIO2) (initial M2) = p2
  ∧ path M2 (io || trIO2) (initial M2)
  ∧ length io = length trIO2
  using trP-zip trP-split assms(7) trP-def trIO2-def by auto
then have p2 ∈ io-targets M2 (initial M2) io
  by auto
then have targets-2 : io-targets M2 (initial M2) io = {p2}
  by (metis state-component-2 io-assm singletonD)

have target (io || trIO1) (initial M1) = p1
  ∧ path M1 (io || trIO1) (initial M1)
  ∧ length io = length trIO1
  using trP-zip trP-split assms(7) trP-def trIO1-def by auto
then have p1 ∈ io-targets M1 (initial M1) io
  by auto
then have targets-1 : io-targets M1 (initial M1) io = {p1}
  by (metis io-lang-subst assms(2) io-targets-observable-singleton-ex singletonD)

have io-targets M2 (initial M2) io × io-targets M1 (initial M1) io = {(p2,p1)}
  using targets-2 targets-1 by simp
then show io-targets PM (initial PM) io
  = io-targets M2 (initial M2) io × io-targets M1 (initial M1) io
  using targets-pm by simp
qed

ultimately have state-components : ∀ io ∈ R M2 s vs xs . io-targets PM (initial PM) io
  = {s} × io-targets M1 (initial M1) io
  by auto

then have ⋃ (image (io-targets PM (initial PM)) (R M2 s vs xs))
  = ⋃ (image (λ io . {s} × io-targets M1 (initial M1) io) (R M2 s vs xs))
  by auto
then have ⋃ (image (io-targets PM (initial PM)) (R M2 s vs xs))
  = {s} × ⋃ (image (io-targets M1 (initial M1)) (R M2 s vs xs))
  by auto
then have card (⋃ (image (io-targets PM (initial PM)) (R M2 s vs xs)))
  = card (⋃ (image (io-targets M1 (initial M1)) (R M2 s vs xs)))
  by (metis (no-types) card-cartesian-product-singleton)

moreover have card (⋃ (image (io-targets PM (initial PM)) (R M2 s vs xs)))
  = card (R M2 s vs xs)
proof (rule ccontr)
  assume assm : card (⋃ (io-targets PM (initial PM) ‘ R M2 s vs xs )) ≠ card (R M2 s vs xs)

  have ∀ io ∈ R M2 s vs xs . io ∈ L PM
  proof
    fix io assume io-assm : io ∈ R M2 s vs xs
    then have prefix io (vs @ xs)
      by auto
    then have io ∈ L M1 ∧ io ∈ L M2
      using assms(1) unfolding prefix-def by (metis IntE language-state language-state-split)
    then show io ∈ L PM
      using productF-language assms by blast
  end
end

```

**qed**  
**then have**  $singletons : \forall io \in R\ M2\ s\ vs\ xs . (\exists t . io\text{-targets}\ PM\ (initial\ PM)\ io = \{t\})$   
**using**  $io\text{-targets-observable-singleton-ex}\ observable\ productF\ assms$  **by**  $metis$   
**then have**  $card\text{-targets} : card\ (\bigcup (io\text{-targets}\ PM\ (initial\ PM)\ 'R\ M2\ s\ vs\ xs))$   
 $= card\ (image\ (io\text{-targets}\ PM\ (initial\ PM))\ (R\ M2\ s\ vs\ xs))$   
**using**  $finite\text{-}R\ card\text{-union-of-singletons}$   
 $[of\ image\ (io\text{-targets}\ PM\ (initial\ PM))\ (R\ M2\ s\ vs\ xs)]$   
**by**  $simp$

**moreover have**  $card\ (image\ (io\text{-targets}\ PM\ (initial\ PM))\ (R\ M2\ s\ vs\ xs)) \leq card\ (R\ M2\ s\ vs\ xs)$   
**using**  $finite\text{-}R$  **by**  $(metis\ card\text{-image-le})$   
**ultimately have**  $card\text{-le} : card\ (\bigcup (io\text{-targets}\ PM\ (initial\ PM)\ 'R\ M2\ s\ vs\ xs))$   
 $< card\ (R\ M2\ s\ vs\ xs)$   
**using**  $assm$  **by**  $linarith$

**have**  $\exists io1 \in (R\ M2\ s\ vs\ xs) . \exists io2 \in (R\ M2\ s\ vs\ xs) . io1 \neq io2$   
 $\wedge io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2 \neq \{\}$   
**proof**  $(rule\ ccontr)$   
**assume**  $\neg (\exists io1 \in R\ M2\ s\ vs\ xs . \exists io2 \in R\ M2\ s\ vs\ xs . io1 \neq io2$   
 $\wedge io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2 \neq \{\})$   
**then have**  $\forall io1 \in R\ M2\ s\ vs\ xs . \forall io2 \in R\ M2\ s\ vs\ xs . io1 = io2$   
 $\vee io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2 = \{\}$   
**by**  $blast$   
**moreover have**  $\forall io \in R\ M2\ s\ vs\ xs . io\text{-targets}\ PM\ (initial\ PM)\ io \neq \{\}$   
**by**  $(metis\ insert\ not\ empty\ singletons)$   
**ultimately have**  $card\ (image\ (io\text{-targets}\ PM\ (initial\ PM))\ (R\ M2\ s\ vs\ xs))$   
 $= card\ (R\ M2\ s\ vs\ xs)$   
**using**  $finite\text{-}R[of\ M2\ s\ vs\ xs]\ card\text{-union-of-distinct}$   
 $[of\ R\ M2\ s\ vs\ xs\ (io\text{-targets}\ PM\ (initial\ PM))]$   
**by**  $blast$   
**then show**  $False$   
**using**  $card\text{-le}\ card\text{-targets}$  **by**  $linarith$   
**qed**

**then have**  $\exists io1\ io2 . io1 \in (R\ M2\ s\ vs\ xs)$   
 $\wedge io2 \in (R\ M2\ s\ vs\ xs)$   
 $\wedge io1 \neq io2$   
 $\wedge io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2 \neq \{\}$   
**by**  $blast$   
**moreover have**  $\forall io1\ io2 . (io1 \in (R\ M2\ s\ vs\ xs) \wedge io2 \in (R\ M2\ s\ vs\ xs) \wedge io1 \neq io2)$   
 $\longrightarrow length\ io1 \neq length\ io2$   
**proof**  $(rule\ ccontr)$   
**assume**  $\neg (\forall io1\ io2 . io1 \in R\ M2\ s\ vs\ xs \wedge io2 \in R\ M2\ s\ vs\ xs \wedge io1 \neq io2$   
 $\longrightarrow length\ io1 \neq length\ io2)$   
**then obtain**  $io1\ io2$  **where**  $io\text{-def} : io1 \in R\ M2\ s\ vs\ xs$   
 $\wedge io2 \in R\ M2\ s\ vs\ xs$   
 $\wedge io1 \neq io2$   
 $\wedge length\ io1 = length\ io2$   
**by**  $auto$   
**then have**  $prefix\ io1\ (vs\ @\ xs) \wedge prefix\ io2\ (vs\ @\ xs)$   
**by**  $auto$   
**then have**  $io1 = take\ (length\ io1)\ (vs\ @\ xs) \wedge io2 = take\ (length\ io2)\ (vs\ @\ xs)$   
**by**  $(metis\ append\ eq\ conv\ conj\ prefixE)$   
**then show**  $False$   
**using**  $io\text{-def}$  **by**  $auto$   
**qed**

**ultimately obtain**  $io1\ io2$  **where**  $rep\text{-ios-def} :$   
 $io1 \in (R\ M2\ s\ vs\ xs)$   
 $\wedge io2 \in (R\ M2\ s\ vs\ xs)$   
 $\wedge length\ io1 < length\ io2$   
 $\wedge io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2 \neq \{\}$   
**by**  $(metis\ inf\ sup\ aci(1)\ linorder\ neqE\ nat)$

**obtain**  $rep$  **where**  $(s,rep) \in io\text{-targets}\ PM\ (initial\ PM)\ io1 \cap io\text{-targets}\ PM\ (initial\ PM)\ io2$   
**proof** –

```

assume  $a1$ :  $\bigwedge rep. (s, rep) \in io\text{-targets } PM \text{ (initial } PM) io1 \cap io\text{-targets } PM \text{ (initial } PM) io2$ 
            $\implies thesis$ 
have  $\exists f. Sigma \{s\} f \cap (io\text{-targets } PM \text{ (initial } PM) io1 \cap io\text{-targets } PM \text{ (initial } PM) io2)$ 
            $\neq \{\}$ 
  by (metis (no-types) inf.left-idem rep-ios-def state-components)
then show  $?thesis$ 
  using  $a1$  by blast
qed
then have  $rep\text{-state} : io\text{-targets } PM \text{ (initial } PM) io1 = \{(s, rep)\}$ 
            $\wedge io\text{-targets } PM \text{ (initial } PM) io2 = \{(s, rep)\}$ 
  by (metis Int-iff rep-ios-def singletonD singletons)

obtain  $io1X io2X$  where  $rep\text{-ios-split} : io1 = vs @ io1X$ 
            $\wedge prefix\ io1X\ xs$ 
            $\wedge io2 = vs @ io2X$ 
            $\wedge prefix\ io2X\ xs$ 

  using rep-ios-def by auto
then have  $length\ io1 > length\ vs$ 
  using rep-ios-def by auto

```

— get a path from (initial PM) to (q2,q1)

```

have  $vs@xs \in L\ PM$ 
  by (metis (no-types) assms(1) assms(4) assms(5) assms(7) inf-commute productF-language)
then have  $vs \in L\ PM$ 
  by (meson language-state-prefix)
then obtain  $trV$  where  $trV\text{-def} : \{tr . path\ PM \ (vs \parallel tr) \text{ (initial } PM) \wedge length\ vs = length\ tr\}$ 
            $= \{trV\}$ 
  using observable-path-unique-ex[of PM vs initial PM]
            $assms(2) assms(3) assms(7) observable-productF$ 
  by blast
let  $?qv = target\ (vs \parallel trV) \text{ (initial } PM)$ 

have  $?qv \in io\text{-targets } PM \text{ (initial } PM) vs$ 
  using trV-def by auto
then have  $qv\text{-simp}[simp] : ?qv = (q2, q1)$ 
  using singletons assms by blast
then have  $?qv \in nodes\ PM$ 
  using trV-def assms by blast

```

— get a path using io1X from the state reached by vs in PM

```

obtain  $tr1X\text{-all}$  where  $tr1X\text{-all-def} : path\ PM \ (vs @ io1X \parallel tr1X\text{-all}) \text{ (initial } PM)$ 
            $\wedge length\ (vs @ io1X) = length\ tr1X\text{-all}$ 
  using rep-ios-def rep-ios-split by auto
let  $?tr1X = drop\ (length\ vs) tr1X\text{-all}$ 
have  $take\ (length\ vs) tr1X\text{-all} = trV$ 
proof –
  have  $path\ PM \ (vs \parallel take\ (length\ vs) tr1X\text{-all}) \text{ (initial } PM)$ 
            $\wedge length\ vs = length\ (take\ (length\ vs) tr1X\text{-all})$ 
  using tr1X-all-def trV-def
  by (metis (no-types, lifting) FSM.path-append-elim append-eq-conv-conj
            $length\text{-take}\ zip\text{-append1})$ 
  then show  $take\ (length\ vs) tr1X\text{-all} = trV$ 
  using trV-def by blast
qed
then have  $tr1X\text{-def} : path\ PM \ (io1X \parallel ?tr1X) ?qv \wedge length\ io1X = length\ ?tr1X$ 
proof –
  have  $length\ tr1X\text{-all} = length\ vs + length\ io1X$ 
  using tr1X-all-def by auto
then have  $length\ io1X = length\ tr1X\text{-all} - length\ vs$ 
  by presburger
then show  $?thesis$ 
  by (metis (no-types) FSM.path-append-elim <take (length vs) tr1X-all = trV>)

```

$length\text{-}drop\ tr1X\text{-}all\text{-}def\ zip\text{-}append1)$   
**qed**  
**then have**  $io1X\text{-}lang : io1X \in language\text{-}state\ PM\ ?qv$   
**by** *auto*  
**then obtain**  $tr1X'$  **where**  $tr1X'\text{-}def : \{tr . path\ PM\ (io1X\ ||\ tr)\ ?qv \wedge length\ io1X = length\ tr\}$   
 $= \{tr1X'\}$   
**using**  $observable\text{-}path\text{-}unique\text{-}ex[of\ PM\ io1X\ ?qv]$   
 $assms(2)\ assms(3)\ assms(7)\ observable\text{-}productF$   
**by** *blast*  
**moreover have**  $?tr1X \in \{tr . path\ PM\ (io1X\ ||\ tr)\ ?qv \wedge length\ io1X = length\ tr\}$   
**using**  $tr1X\text{-}def$  **by** *auto*  
**ultimately have**  $tr1x\text{-}unique : tr1X' = ?tr1X$   
**by** *simp*

— get a path using  $io2X$  from the state reached by  $vs$  in  $PM$

**obtain**  $tr2X\text{-}all$  **where**  $tr2X\text{-}all\text{-}def : path\ PM\ (vs\ @\ io2X\ ||\ tr2X\text{-}all)\ (initial\ PM)$   
 $\wedge length\ (vs\ @\ io2X) = length\ tr2X\text{-}all$   
**using**  $rep\text{-}ios\text{-}def\ rep\text{-}ios\text{-}split$  **by** *auto*  
**let**  $?tr2X = drop\ (length\ vs)\ tr2X\text{-}all$   
**have**  $take\ (length\ vs)\ tr2X\text{-}all = trV$   
**proof** —  
**have**  $path\ PM\ (vs\ ||\ take\ (length\ vs)\ tr2X\text{-}all)\ (initial\ PM)$   
 $\wedge length\ vs = length\ (take\ (length\ vs)\ tr2X\text{-}all)$   
**using**  $tr2X\text{-}all\text{-}def\ trV\text{-}def$   
**by**  $(metis\ (no\text{-}types,\ lifting)\ FSM.path\text{-}append\text{-}elim\ append\text{-}eq\text{-}conv\text{-}conj$   
 $length\text{-}take\ zip\text{-}append1)$   
**then show**  $take\ (length\ vs)\ tr2X\text{-}all = trV$   
**using**  $trV\text{-}def$  **by** *blast*

**qed**  
**then have**  $tr2X\text{-}def : path\ PM\ (io2X\ ||\ ?tr2X)\ ?qv \wedge length\ io2X = length\ ?tr2X$   
**proof** —  
**have**  $length\ tr2X\text{-}all = length\ vs + length\ io2X$   
**using**  $tr2X\text{-}all\text{-}def$  **by** *auto*  
**then have**  $length\ io2X = length\ tr2X\text{-}all - length\ vs$   
**by** *presburger*  
**then show**  $?thesis$   
**by**  $(metis\ (no\text{-}types)\ FSM.path\text{-}append\text{-}elim\ \langle take\ (length\ vs)\ tr2X\text{-}all = trV \rangle$   
 $length\text{-}drop\ tr2X\text{-}all\text{-}def\ zip\text{-}append1)$

**qed**  
**then have**  $io2X\text{-}lang : io2X \in language\text{-}state\ PM\ ?qv$  **by** *auto*  
**then obtain**  $tr2X'$  **where**  $tr2X'\text{-}def : \{tr . path\ PM\ (io2X\ ||\ tr)\ ?qv \wedge length\ io2X = length\ tr\}$   
 $= \{tr2X'\}$   
**using**  $observable\text{-}path\text{-}unique\text{-}ex[of\ PM\ io2X\ ?qv]$   $assms(2)\ assms(3)\ assms(7)\ observable\text{-}productF$   
**by** *blast*  
**moreover have**  $?tr2X \in \{tr . path\ PM\ (io2X\ ||\ tr)\ ?qv \wedge length\ io2X = length\ tr\}$   
**using**  $tr2X\text{-}def$  **by** *auto*  
**ultimately have**  $tr2x\text{-}unique : tr2X' = ?tr2X$   
**by** *simp*

— both paths reach the same state

**have**  $io\text{-}targets\ PM\ (initial\ PM)\ (vs\ @\ io1X) = \{(s,rep)\}$   
**using**  $rep\text{-}state\ rep\text{-}ios\text{-}split$  **by** *auto*  
**moreover have**  $io\text{-}targets\ PM\ (initial\ PM)\ vs = \{?qv\}$   
**using**  $assms(8)$  **by** *auto*  
**ultimately have**  $rep\text{-}via\text{-}1 : io\text{-}targets\ PM\ ?qv\ io1X = \{(s,rep)\}$   
**by**  $(meson\ obs\text{-}PM\ observable\text{-}io\text{-}targets\text{-}split)$   
**then have**  $rep\text{-}tgt\text{-}1 : target\ (io1X\ ||\ tr1X')\ ?qv = (s,rep)$   
**using**  $obs\text{-}PM\ observable\text{-}io\text{-}target\text{-}unique\text{-}target[of\ PM\ ?qv\ io1X\ (s,rep)]\ tr1X'\text{-}def$  **by** *blast*  
**have**  $length\text{-}1 : length\ (io1X\ ||\ tr1X') > 0$   
**using**  $\langle length\ vs < length\ io1 \rangle\ rep\text{-}ios\text{-}split\ tr1X\text{-}def\ tr1x\text{-}unique$  **by** *auto*  
  
**have**  $tr1X\text{-}alt\text{-}def : tr1X' = take\ (length\ io1X)\ tr$   
**by**  $(metis\ (no\text{-}types)\ assms(10)\ assms(9)\ obs\text{-}PM\ observable\text{-}path\text{-}prefix\ qv\text{-}simp$   
 $rep\text{-}ios\text{-}split\ tr1X\text{-}def\ tr1x\text{-}unique)$

**moreover have**  $io1X = take (length\ io1X)\ xs$   
**using**  $rep\ ios\ split$  **by**  $(metis\ append\ eq\ conv\ conj\ prefixE)$   
**ultimately have**  $(io1X \parallel tr1X') = take (length\ io1X)\ (xs \parallel tr)$   
**by**  $(metis\ take\ zip)$   
**moreover have**  $length\ (xs \parallel tr) \geq length\ (io1X \parallel tr1X')$   
**by**  $(metis\ (no\ types)\ \langle io1X = take (length\ io1X)\ xs \rangle\ assms(10)\ length\ take\ length\ zip\ nat\ le\ linear\ take\ all\ tr1X\ def\ tr1x\ unique)$   
**ultimately have**  $rep\ idx\ 1 : (states\ (xs \parallel tr)\ ?qv) ! ((length\ io1X) - 1) = (s, rep)$   
**by**  $(metis\ (no\ types,\ lifting)\ One\ nat\ def\ Suc\ less\ eq\ Suc\ pred\ rep\ tgt\ 1\ length\ 1\ less\ Suc\ eq\ le\ map\ snd\ zip\ scan\ length\ scan\ nth\ states\ alt\ def\ tr1X\ def\ tr1x\ unique)$

**have**  $io\ targets\ PM\ (initial\ PM)\ (vs\ @\ io2X) = \{(s, rep)\}$   
**using**  $rep\ state\ rep\ ios\ split$  **by**  $auto$   
**moreover have**  $io\ targets\ PM\ (initial\ PM)\ vs = \{?qv\}$   
**using**  $assms(8)$  **by**  $auto$   
**ultimately have**  $rep\ via\ 2 : io\ targets\ PM\ ?qv\ io2X = \{(s, rep)\}$   
**by**  $(meson\ obs\ PM\ observable\ io\ targets\ split)$   
**then have**  $rep\ tgt\ 2 : target\ (io2X \parallel tr2X')\ ?qv = (s, rep)$   
**using**  $obs\ PM\ observable\ io\ target\ unique\ target[of\ PM\ ?qv\ io2X\ (s, rep)]\ tr2X'\ def$  **by**  $blast$   
**moreover have**  $length\ 2 : length\ (io2X \parallel tr2X') > 0$   
**by**  $(metis\ \langle length\ vs < length\ io1 \rangle\ append.\ right\ neutral\ length\ 0\ conv\ length\ zip\ less\ asym\ min.\ idem\ neq0\ conv\ rep\ ios\ def\ rep\ ios\ split\ tr2X\ def\ tr2x\ unique)$

**have**  $tr2X\ alt\ def : tr2X' = take (length\ io2X)\ tr$   
**by**  $(metis\ (no\ types)\ assms(10)\ assms(9)\ obs\ PM\ observable\ path\ prefix\ qv\ simp\ rep\ ios\ split\ tr2X\ def\ tr2x\ unique)$   
**moreover have**  $io2X = take (length\ io2X)\ xs$   
**using**  $rep\ ios\ split$  **by**  $(metis\ append\ eq\ conv\ conj\ prefixE)$   
**ultimately have**  $(io2X \parallel tr2X') = take (length\ io2X)\ (xs \parallel tr)$   
**by**  $(metis\ take\ zip)$   
**moreover have**  $length\ (xs \parallel tr) \geq length\ (io2X \parallel tr2X')$   
**using**  $calculation$  **by**  $auto$   
**ultimately have**  $rep\ idx\ 2 : (states\ (xs \parallel tr)\ ?qv) ! ((length\ io2X) - 1) = (s, rep)$   
**by**  $(metis\ (no\ types,\ lifting)\ One\ nat\ def\ Suc\ less\ eq\ Suc\ pred\ rep\ tgt\ 2\ length\ 2\ less\ Suc\ eq\ le\ map\ snd\ zip\ scan\ length\ scan\ nth\ states\ alt\ def\ tr2X\ def\ tr2x\ unique)$

— thus the distinctness assumption is violated

**have**  $length\ io1X \neq length\ io2X$   
**by**  $(metis\ \langle io1X = take (length\ io1X)\ xs \rangle\ \langle io2X = take (length\ io2X)\ xs \rangle\ less\ irrefl\ rep\ ios\ def\ rep\ ios\ split)$   
**moreover have**  $(states\ (xs \parallel tr)\ ?qv) ! ((length\ io1X) - 1) = (states\ (xs \parallel tr)\ ?qv) ! ((length\ io2X) - 1)$   
**using**  $rep\ idx\ 1\ rep\ idx\ 2$  **by**  $simp$   
**ultimately have**  $\neg (distinct\ (states\ (xs \parallel tr)\ ?qv))$   
**by**  $(metis\ Suc\ less\ eq\ \langle io1X = take (length\ io1X)\ xs \rangle\ \langle io1X \parallel tr1X' = take (length\ io1X)\ (xs \parallel tr) \rangle\ \langle io2X = take (length\ io2X)\ xs \rangle\ \langle io2X \parallel tr2X' = take (length\ io2X)\ (xs \parallel tr) \rangle\ \langle length\ (io1X \parallel tr1X') \leq length\ (xs \parallel tr) \rangle\ \langle length\ (io2X \parallel tr2X') \leq length\ (xs \parallel tr) \rangle\ assms(10)\ diff\ Suc\ 1\ distinct\ conv\ nth\ gr0\ conv\ Suc\ le\ imp\ less\ Suc\ length\ 1\ length\ 2\ length\ take\ map\ snd\ zip\ scan\ length\ states\ alt\ def)$   
**then show**  $False$   
**by**  $(metis\ assms(11)\ states\ alt\ def)$   
**qed**

**ultimately show**  $?thesis$   
**by**  $linarith$   
**qed**

**lemma**  $R\ state\ component\ 2 :$   
**assumes**  $io \in (R\ M2\ s\ vs\ xs)$   
**and**  $observable\ M2$   
**shows**  $io\ targets\ M2\ (initial\ M2)\ io = \{s\}$

**proof** –  
**have**  $s \in \text{io-targets } M2$  (*initial*  $M2$ )  $\text{io}$   
**using**  $\text{assms}(1)$  **by** *auto*  
**moreover have**  $\text{io} \in \text{language-state } M2$  (*initial*  $M2$ )  
**using** *calculation* **by** *auto*  
**ultimately show**  $\text{io-targets } M2$  (*initial*  $M2$ )  $\text{io} = \{s\}$   
**using**  $\text{assms}(2)$   $\text{io-targets-observable-singleton-ex}$  **by** (*metis singletonD*)  
**qed**

**lemma** *R-union-card-is-suffix-length* :

**assumes**  $\text{OFSM } M2$   
**and**  $\text{io}@xs \in L M2$   
**shows**  $\text{sum } (\lambda q . \text{card } (R M2 q \text{io } xs)) (\text{nodes } M2) = \text{length } xs$   
**using**  $\text{assms}$  **proof** (*induction*  $xs$  *rule: rev-induct*)  
**case** *Nil*  
**show** *?case*  
**by** (*simp add: sum.neutral*)  
**next**  
**case** (*snoc*  $x xs$ )  
  
**have** *finite* ( $\text{nodes } M2$ )  
**using**  $\text{assms}$  **by** *auto*  
  
**have**  $R\text{-update} : \bigwedge q . R M2 q \text{io } (xs@[x]) = (\text{if } (q \in \text{io-targets } M2$  (*initial*  $M2$ )  $(\text{io} @ xs @ [x]))$   
 $\text{then insert } (\text{io}@xs@[x]) (R M2 q \text{io } xs)$   
 $\text{else } R M2 q \text{io } xs)$   
  
**by** *auto*  
  
**obtain**  $q$  **where**  $\text{io-targets } M2$  (*initial*  $M2$ )  $(\text{io} @ xs @ [x]) = \{q\}$   
**by** (*meson*  $\text{assms}(1)$   $\text{io-targets-observable-singleton-ex}$   $\text{snoc.prem}(2)$ )  
  
**then have**  $R M2 q \text{io } (xs@[x]) = \text{insert } (\text{io}@xs@[x]) (R M2 q \text{io } xs)$   
**using**  $R\text{-update}$  **by** *auto*  
**moreover have**  $(\text{io}@xs@[x]) \notin (R M2 q \text{io } xs)$   
**by** *auto*  
**ultimately have**  $\text{card } (R M2 q \text{io } (xs@[x])) = \text{Suc } (\text{card } (R M2 q \text{io } xs))$   
**by** (*metis card-insert-disjoint finite-R*)  
  
**have**  $q \in \text{nodes } M2$   
**by** (*metis* (*full-types*)  $\text{FSM.nodes.initial}$   $\langle \text{io-targets } M2$  (*initial*  $M2$ )  $(\text{io}@xs @ [x]) = \{q\} \rangle$   
 $\text{insertI1 io-targets-nodes}$ )  
  
**have**  $\forall q' . q' \neq q \longrightarrow R M2 q' \text{io } (xs@[x]) = R M2 q' \text{io } xs$   
**using**  $\langle \text{io-targets } M2$  (*initial*  $M2$ )  $(\text{io}@xs @ [x]) = \{q\} \rangle$   $R\text{-update}$   
**by** *auto*  
**then have**  $\forall q' . q' \neq q \longrightarrow \text{card } (R M2 q' \text{io } (xs@[x])) = \text{card } (R M2 q' \text{io } xs)$   
**by** *auto*  
  
**then have**  $(\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } (xs@[x])))$   
 $= (\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } xs))$   
**by** *auto*  
**moreover have**  $(\sum_{q \in \text{nodes } M2} \text{card } (R M2 q \text{io } (xs@[x])))$   
 $= (\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } (xs@[x]))) + (\text{card } (R M2 q \text{io } (xs@[x])))$   
 $(\sum_{q \in \text{nodes } M2} \text{card } (R M2 q \text{io } xs))$   
 $= (\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } xs)) + (\text{card } (R M2 q \text{io } xs))$   
**proof** –  
**have**  $\forall C c f . (\text{infinite } C \vee (c::'c) \notin C) \vee \text{sum } f C = (f c::\text{nat}) + \text{sum } f (C - \{c\})$   
**by** (*meson*  $\text{sum.remove}$ )  
**then show**  $(\sum_{q \in \text{nodes } M2} \text{card } (R M2 q \text{io } (xs@[x])))$   
 $= (\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } (xs@[x]))) + (\text{card } (R M2 q \text{io } (xs@[x])))$   
 $(\sum_{q \in \text{nodes } M2} \text{card } (R M2 q \text{io } xs))$   
 $= (\sum_{q \in (\text{nodes } M2 - \{q\})} \text{card } (R M2 q \text{io } xs)) + (\text{card } (R M2 q \text{io } xs))$   
**using**  $\langle \text{finite } (\text{nodes } M2) \rangle$   $\langle q \in \text{nodes } M2 \rangle$  **by**  $\text{presburger+}$   
**qed**  
**ultimately have**  $(\sum_{q \in \text{nodes } M2} \text{card } (R M2 q \text{io } (xs@[x])))$

$$= \text{Suc} \left( \sum_{q \in \text{nodes } M2} \text{card} (R M2 q \text{ io } xs) \right)$$
**using**  $\langle \text{card} (R M2 q \text{ io } (xs@[x])) = \text{Suc} (\text{card} (R M2 q \text{ io } xs)) \rangle$   
**by** *presburger*

**have**  $(\sum_{q \in \text{nodes } M2} \text{card} (R M2 q \text{ io } xs)) = \text{length } xs$   
**using** *snoc.IH snoc.premis language-state-prefix[of io@xs [x] M2 initial M2]*  
**proof** –  
**show** *?thesis*  
**by** (*metis (no-types)*  $\langle (io @ xs) @ [x] \in L M2 \implies io @ xs \in L M2 \rangle$   
 $\langle \text{OFSM } M2 \rangle \langle io @ xs @ [x] \in L M2 \rangle \text{append.assoc snoc.IH}$ )  
**qed**

**show** *?case*  
**proof** –  
**show** *?thesis*  
**by** (*metis (no-types)*  
 $\langle (\sum_{q \in \text{nodes } M2} \text{card} (R M2 q \text{ io } (xs @ [x]))) = \text{Suc} (\sum_{q \in \text{nodes } M2} \text{card} (R M2 q \text{ io } xs)) \rangle$   
 $\langle (\sum_{q \in \text{nodes } M2} \text{card} (R M2 q \text{ io } xs)) = \text{length } xs \rangle \text{length-append-singleton}$ )  
**qed**  
**qed**

**lemma** *R-state-repetition-via-long-sequence* :  
**assumes** *OFSM M*  
**and**  $\text{card} (\text{nodes } M) \leq m$   
**and**  $\text{Suc} (m * m) \leq \text{length } xs$   
**and**  $vs@xs \in L M$   
**shows**  $\exists q \in \text{nodes } M . \text{card} (R M q \text{ vs } xs) > m$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\exists q \in \text{nodes } M . m < \text{card} (R M q \text{ vs } xs))$   
**then have**  $\forall q \in \text{nodes } M . \text{card} (R M q \text{ vs } xs) \leq m$   
**by** *auto*  
**then have**  $\text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M) \leq \text{sum} (\lambda q . m) (\text{nodes } M)$   
**by** (*meson sum-mono*)  
**moreover have**  $\text{sum} (\lambda q . m) (\text{nodes } M) \leq m * m$   
**using** *assms(2)* **by** *auto*  
**ultimately have**  $\text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M) \leq m * m$   
**by** *presburger*

**moreover have**  $\text{Suc} (m*m) \leq \text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M)$   
**using** *R-union-card-is-suffix-length[OF assms(1), of vs xs] assms(4,3)* **by** *auto*  
**ultimately show** *False* **by** *simp*  
**qed**

**lemma** *R-state-repetition-distribution* :  
**assumes** *OFSM M*  
**and**  $\text{Suc} (\text{card} (\text{nodes } M) * m) \leq \text{length } xs$   
**and**  $vs@xs \in L M$   
**shows**  $\exists q \in \text{nodes } M . \text{card} (R M q \text{ vs } xs) > m$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\exists q \in \text{nodes } M . m < \text{card} (R M q \text{ vs } xs))$   
**then have**  $\forall q \in \text{nodes } M . \text{card} (R M q \text{ vs } xs) \leq m$   
**by** *auto*  
**then have**  $\text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M) \leq \text{sum} (\lambda q . m) (\text{nodes } M)$   
**by** (*meson sum-mono*)  
**moreover have**  $\text{sum} (\lambda q . m) (\text{nodes } M) \leq \text{card} (\text{nodes } M) * m$   
**using** *assms(2)* **by** *auto*  
**ultimately have**  $\text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M) \leq \text{card} (\text{nodes } M) * m$   
**by** *presburger*

**moreover have**  $\text{Suc} (\text{card} (\text{nodes } M)*m) \leq \text{sum} (\lambda q . \text{card} (R M q \text{ vs } xs)) (\text{nodes } M)$   
**using** *R-union-card-is-suffix-length[OF assms(1), of vs xs] assms(3,2)* **by** *auto*  
**ultimately show** *False*  
**by** *simp*  
**qed**

## 4.4 Function RP

Function RP extends function MR by adding all elements from a set of IO-sequences that also reach the given state.

```
fun RP :: ('in, 'out, 'state) FSM ⇒ 'state ⇒ ('in × 'out) list
  ⇒ ('in × 'out) list ⇒ ('in × 'out) list set
  ⇒ ('in × 'out) list set
```

where

```
RP M s vs xs V'' = R M s vs xs
  ∪ {vs' ∈ V'' . io-targets M (initial M) vs' = {s}}
```

lemma RP-from-R:

assumes *is-det-state-cover* M2 V

and V'' ∈ Perm V M1

shows RP M2 s vs xs V'' = R M2 s vs xs

∨ (∃ vs' ∈ V'' . vs' ∉ R M2 s vs xs ∧ RP M2 s vs xs V'' = insert vs' (R M2 s vs xs))

proof (rule ccontr)

assume *assm* : ¬ (RP M2 s vs xs V'' = R M2 s vs xs ∨

(∃ vs' ∈ V'' . vs' ∉ R M2 s vs xs ∧ RP M2 s vs xs V'' = insert vs' (R M2 s vs xs)))

moreover have R M2 s vs xs ⊆ RP M2 s vs xs V''

by simp

moreover have RP M2 s vs xs V'' ⊆ R M2 s vs xs ∪ V''

by auto

ultimately obtain vs1 vs2 where *vs-def* :

vs1 ≠ vs2 ∧ vs1 ∈ V'' ∧ vs2 ∈ V''

∧ vs1 ∉ R M2 s vs xs ∧ vs2 ∉ R M2 s vs xs

∧ vs1 ∈ RP M2 s vs xs V'' ∧ vs2 ∈ RP M2 s vs xs V''

by blast

then have *io-targets* M2 (initial M2) vs1 = {s} ∧ *io-targets* M2 (initial M2) vs2 = {s}

by (metis (mono-tags, lifting) RP.simps Un-iff mem-Collect-eq)

then have *io-targets* M2 (initial M2) vs1 = *io-targets* M2 (initial M2) vs2

by simp

obtain *f* where *f-def* : *is-det-state-cover-ass* M2 *f* ∧ V = *f* ' *d-reachable* M2 (initial M2)

using *assms* by auto

moreover have V = *image* *f* (*d-reachable* M2 (initial M2))

using *f-def* by blast

moreover have *map fst* vs1 ∈ V ∧ *map fst* vs2 ∈ V

using *assms*(2) *perm-inputs* *vs-def* by blast

ultimately obtain r1 r2 where *r-def* :

*f* r1 = *map fst* vs1 ∧ r1 ∈ *d-reachable* M2 (initial M2)

*f* r2 = *map fst* vs2 ∧ r2 ∈ *d-reachable* M2 (initial M2)

by force

then have *d-reaches* M2 (initial M2) (*map fst* vs1) r1

*d-reaches* M2 (initial M2) (*map fst* vs2) r2

by (metis *f-def* *is-det-state-cover-ass.elims*(2))+

then have *io-targets* M2 (initial M2) vs1 ⊆ {r1}

using *d-reaches-io-target*[of M2 initial M2 *map fst* vs1 r1 *map snd* vs1] by simp

moreover have *io-targets* M2 (initial M2) vs2 ⊆ {r2}

using *d-reaches-io-target*[of M2 initial M2 *map fst* vs2 r2 *map snd* vs2]

⟨*d-reaches* M2 (initial M2) (*map fst* vs2) r2⟩ by auto

ultimately have r1 = r2

using ⟨*io-targets* M2 (initial M2) vs1 = {s} ∧ *io-targets* M2 (initial M2) vs2 = {s}⟩ by auto

have *map fst* vs1 ≠ *map fst* vs2

using *assms*(2) *perm-inputs-diff* *vs-def* by blast

then have r1 ≠ r2

using *r-def*(1) *r-def*(2) by force

then show False

using ⟨r1 = r2⟩ by auto

qed

**lemma** *finite-RP* :  
**assumes** *is-det-state-cover*  $M2\ V$   
**and**  $V'' \in Perm\ V\ M1$   
**shows** *finite* ( $RP\ M2\ s\ vs\ xs\ V''$ )  
**using** *assms RP-from-R finite-R* **by** (*metis finite-insert*)

**lemma** *RP-count* :  
**assumes**  $(vs\ @\ xs) \in L\ M1 \cap L\ M2$   
**and** *observable*  $M1$   
**and** *observable*  $M2$   
**and** *well-formed*  $M1$   
**and** *well-formed*  $M2$   
**and**  $s \in nodes\ M2$   
**and** *productF*  $M2\ M1\ FAIL\ PM$   
**and** *io-targets*  $PM\ (initial\ PM)\ vs = \{(q2, q1)\}$   
**and** *path*  $PM\ (xs\ ||\ tr)\ (q2, q1)$   
**and**  $length\ xs = length\ tr$   
**and** *distinct*  $(states\ (xs\ ||\ tr)\ (q2, q1))$   
**and** *is-det-state-cover*  $M2\ V$   
**and**  $V'' \in Perm\ V\ M1$   
**and**  $\forall s' \in set\ (states\ (xs\ ||\ map\ fst\ tr)\ q2) . \neg (\exists v \in V . d-reaches\ M2\ (initial\ M2)\ v\ s')$   
**shows**  $card\ (\bigcup (image\ (io-targets\ M1\ (initial\ M1))\ (RP\ M2\ s\ vs\ xs\ V'')))) = card\ (RP\ M2\ s\ vs\ xs\ V'')$   
— each sequence in the set calculated by RP reaches a different state in M1  
**proof** —

— Proof sketch: - RP calculates either the same set as R or the set of R and an additional element - in the first case, the result for R applies - in the second case, the additional element is not contained in the set calculated by R due to the assumption that no state reached by a non-empty prefix of xs after vs is also reached by some sequence in V (see the last two assumptions)

**have** *RP-cases* :  $RP\ M2\ s\ vs\ xs\ V'' = R\ M2\ s\ vs\ xs$   
 $\vee (\exists vs' \in V'' . vs' \notin R\ M2\ s\ vs\ xs$   
 $\wedge RP\ M2\ s\ vs\ xs\ V'' = insert\ vs'\ (R\ M2\ s\ vs\ xs))$

**using** *RP-from-R assms* **by** *metis*  
**show** *?thesis*  
**proof** (*cases*  $RP\ M2\ s\ vs\ xs\ V'' = R\ M2\ s\ vs\ xs$ )  
**case** *True*  
**then show** *?thesis* **using** *R-count assms* **by** *metis*  
**next**  
**case** *False*  
**then obtain**  $vs'$  **where** *vs'-def* :  $vs' \in V''$   
 $\wedge vs' \notin R\ M2\ s\ vs\ xs$   
 $\wedge RP\ M2\ s\ vs\ xs\ V'' = insert\ vs'\ (R\ M2\ s\ vs\ xs)$   
**using** *RP-cases* **by** *auto*

**have** *obs-PM* : *observable*  $PM$   
**using** *observable-productF assms(2) assms(3) assms(7)* **by** *blast*

**have** *state-component-2* :  $\forall io \in (R\ M2\ s\ vs\ xs) . io-targets\ M2\ (initial\ M2)\ io = \{s\}$   
**proof**  
**fix**  $io$  **assume**  $io \in R\ M2\ s\ vs\ xs$   
**then have**  $s \in io-targets\ M2\ (initial\ M2)\ io$   
**by** *auto*  
**moreover have**  $io \in language-state\ M2\ (initial\ M2)$   
**using** *calculation* **by** *auto*  
**ultimately show**  $io-targets\ M2\ (initial\ M2)\ io = \{s\}$   
**using** *assms(3) io-targets-observable-singleton-ex* **by** (*metis singletonD*)  
**qed**

**have**  $vs' \in L\ M1$   
**using** *assms(13) perm-language vs'-def* **by** *blast*  
**then obtain**  $s'$  **where** *s'-def* :  $io-targets\ M1\ (initial\ M1)\ vs' = \{s'\}$

by (meson assms(2) io-targets-observable-singleton-ob)

moreover have  $s' \notin \bigcup (\text{image } (\text{io-targets } M1 \text{ (initial } M1)) (R \ M2 \ s \ vs \ xs))$

proof (rule ccontr)

assume  $\neg s' \notin \bigcup (\text{io-targets } M1 \text{ (initial } M1) \text{ ' } R \ M2 \ s \ vs \ xs)$

then obtain  $xs'$  where  $xs'\text{-def} : vs @ xs' \in R \ M2 \ s \ vs \ xs \wedge s' \in \text{io-targets } M1 \text{ (initial } M1) (vs @ xs')$

proof –

assume  $a1 : \bigwedge xs'. vs @ xs' \in R \ M2 \ s \ vs \ xs \wedge s' \in \text{io-targets } M1 \text{ (initial } M1) (vs @ xs')$   
 $\implies \text{thesis}$

obtain  $pps :: ('a \times 'b) \text{ list set} \Rightarrow (('a \times 'b) \text{ list} \Rightarrow 'c \text{ set}) \Rightarrow 'c \Rightarrow ('a \times 'b) \text{ list}$

where

$\forall x0 \ x1 \ x2. (\exists v3. v3 \in x0 \wedge x2 \in x1 \ v3) = (pps \ x0 \ x1 \ x2 \in x0 \wedge x2 \in x1 \ (pps \ x0 \ x1 \ x2))$

by moura

then have  $f2 : pps (R \ M2 \ s \ vs \ xs) (\text{io-targets } M1 \text{ (initial } M1)) s' \in R \ M2 \ s \ vs \ xs$

$\wedge s' \in \text{io-targets } M1 \text{ (initial } M1) (pps (R \ M2 \ s \ vs \ xs)$

$(\text{io-targets } M1 \text{ (initial } M1)) s')$

using  $\langle \neg s' \notin \bigcup (\text{io-targets } M1 \text{ (initial } M1) \text{ ' } R \ M2 \ s \ vs \ xs) \rangle$  by blast

then have  $\exists ps. pps (R \ M2 \ s \ vs \ xs) (\text{io-targets } M1 \text{ (initial } M1)) s' = vs @ ps$

$\wedge ps \neq [] \wedge \text{prefix } ps \ xs \wedge s \in \text{io-targets } M2 \text{ (initial } M2) (vs @ ps)$

by simp

then show ?thesis

using  $f2 \ a1$  by (metis (no-types))

qed

then obtain  $tr'$  where  $tr'\text{-def} : \text{path } M2 (vs @ xs' || tr') (\text{initial } M2)$

$\wedge \text{length } tr' = \text{length } (vs @ xs')$

by auto

then obtain  $trV' \ trX'$  where  $tr'\text{-split} : trV' = \text{take } (\text{length } vs) \ tr'$

$trX' = \text{drop } (\text{length } vs) \ tr'$

$tr' = trV' @ trX'$

by fastforce

then have  $\text{path } M2 (vs || trV') (\text{initial } M2) \wedge \text{length } trV' = \text{length } vs$

by (metis (no-types) FSM.path-append-elim  $\langle trV' = \text{take } (\text{length } vs) \ tr' \rangle$

$\text{append-eq-conv-conj length-take } tr'\text{-def } \text{zip-append1}$ )

have  $\text{initial } PM = (\text{initial } M2, \text{initial } M1)$

using  $\text{assms}(7)$  by simp

moreover have  $vs \in L \ M2 \ vs \in L \ M1$

using  $\text{assms}(1) \text{ language-state-prefix}$  by auto

ultimately have  $\text{io-targets } M1 \text{ (initial } M1) \ vs = \{q1\}$

$\text{io-targets } M2 \text{ (initial } M2) \ vs = \{q2\}$

using  $\text{productF-path-io-targets}$ [of  $M2 \ M1 \ \text{FAIL} \ PM \ \text{initial } M2 \ \text{initial } M1 \ vs \ q2 \ q1$ ]

by (metis FSM.nodes.initial assms(7) assms(8) assms(2) assms(3) assms(4) assms(5)

$\text{io-targets-observable-singleton-ex singletonD}$ )+

then have  $\text{target } (vs || trV') (\text{initial } M2) = q2$

using  $\langle \text{path } M2 (vs || trV') (\text{initial } M2) \wedge \text{length } trV' = \text{length } vs \rangle$   $\text{io-target-target}$

by metis

then have  $\text{path-}xs' : \text{path } M2 (xs' || trX') \ q2 \wedge \text{length } trX' = \text{length } xs'$

by (metis (no-types) FSM.path-append-elim

$\langle \text{path } M2 (vs || trV') (\text{initial } M2) \wedge \text{length } trV' = \text{length } vs \rangle$

$\langle \text{target } (vs || trV') (\text{initial } M2) = q2 \rangle$   $\text{append-eq-conv-conj length-drop } tr'\text{-def}$

$tr'\text{-split}(1) \ tr'\text{-split}(2) \ \text{zip-append2}$ )

have  $\text{io-targets } M2 \text{ (initial } M2) (vs @ xs') = \{s\}$

using  $\text{state-component-2 } xs'\text{-def}$  by blast

then have  $\text{io-targets } M2 \ q2 \ xs' = \{s\}$

by (meson assms(3) observable-io-targets-split  $\langle \text{io-targets } M2 \text{ (initial } M2) \ vs = \{q2\} \rangle$ )

then have  $\text{target-}xs' : \text{target } (xs' || trX') \ q2 = s$

using  $\text{io-target-target path-}xs'$  by metis

moreover have  $\text{length } xs' > 0$

using  $xs'\text{-def}$  by auto

**ultimately have**  $last (states (xs' || trX') q2) = s$   
**using**  $path-xs'$   $target-in-states$  **by**  $metis$   
**moreover have**  $length (states (xs' || trX') q2) > 0$   
**using**  $\langle 0 < length xs' \rangle$   $path-xs'$  **by**  $auto$   
**ultimately have**  $states-xs' : s \in set (states (xs' || trX') q2)$   
**using**  $last-in-set$  **by**  $blast$

**have**  $vs @ xs \in L M2$   
**using**  $assms$  **by**  $simp$   
**then obtain**  $q'$  **where**  $io-targets M2 (initial M2) (vs@xs) = \{q'\}$   
**using**  $io-targets-observable-singleton-ob$ [of  $M2$   $vs@xs$   $initial M2$ ]  $assms(3)$  **by**  $auto$   
**then have**  $xs \in language-state M2 q2$   
**using**  $assms(3)$   $\langle io-targets M2 (initial M2) vs = \{q2\} \rangle$   
 $observable-io-targets-split$ [of  $M2$   $initial M2$   $vs$   $xs$   $q'$   $q2$ ]  
**by**  $auto$

**moreover have**  $path PM (xs || map fst tr || map snd tr) (q2, q1)$   
 $\wedge length xs = length (map fst tr)$   
**using**  $assms(7)$   $assms(9)$   $assms(10)$   $productF-path-unzip$  **by**  $simp$   
**moreover have**  $xs \in language-state PM (q2, q1)$   
**using**  $assms(9)$   $assms(10)$  **by**  $auto$   
**moreover have**  $q2 \in nodes M2$   
**using**  $\langle io-targets M2 (initial M2) vs = \{q2\} \rangle$   $io-targets-nodes$   
**by**  $(metis FSM.nodes.initial insertI1)$   
**moreover have**  $q1 \in nodes M1$   
**using**  $\langle io-targets M1 (initial M1) vs = \{q1\} \rangle$   $io-targets-nodes$   
**by**  $(metis FSM.nodes.initial insertI1)$   
**ultimately have**  $path-xs : path M2 (xs || map fst tr) q2$   
**using**  $productF-path-reverse-ob-2(1)$ [of  $xs$   $map fst tr$   $map snd tr$   $M2$   $M1$   $FAIL$   $PM$   $q2$   $q1$ ]  
 $assms(2,3,4,5,7)$   
**by**  $simp$

**moreover have**  $prefix xs' xs$   
**using**  $xs'-def$  **by**  $auto$   
**ultimately have**  $trX' = take (length xs') (map fst tr)$   
**using**  $\langle path PM (xs || map fst tr || map snd tr) (q2, q1) \wedge length xs = length (map fst tr) \rangle$   
 $assms(3)$   $path-xs'$   
**by**  $(metis observable-path-prefix)$

**then have**  $states-xs : s \in set (states (xs || map fst tr) q2)$   
**by**  $(metis assms(10) in-set-takeD length-map map-snd-zip path-xs' states-alt-def states-xs')$

**have**  $d-reaches M2 (initial M2) (map fst vs') s$   
**proof** –  
**obtain**  $fV$  **where**  $fV-def : is-det-state-cover-ass M2 fV$   
 $\wedge V = fV ' d-reachable M2 (initial M2)$   
**using**  $assms(12)$  **by**  $auto$   
**moreover have**  $V = image fV (d-reachable M2 (initial M2))$   
**using**  $fV-def$  **by**  $blast$   
**moreover have**  $map fst vs' \in V$   
**using**  $perm-inputs vs'-def$   $assms(13)$  **by**  $metis$   
**ultimately obtain**  $qv$  **where**  $qv-def : fV qv = map fst vs' \wedge qv \in d-reachable M2 (initial M2)$   
**by**  $force$   
**then have**  $d-reaches M2 (initial M2) (map fst vs') qv$   
**by**  $(metis fV-def is-det-state-cover-ass.elims(2))$   
**then have**  $io-targets M2 (initial M2) vs' \subseteq \{qv\}$   
**using**  $d-reaches-io-target$ [of  $M2$   $initial M2$   $map fst vs'$   $qv$   $map snd vs'$ ] **by**  $simp$   
**moreover have**  $io-targets M2 (initial M2) vs' = \{s\}$   
**using**  $vs'-def$  **by**  $(metis (mono-tags, lifting) RP.simps Un-iff insertI1 mem-Collect-eq)$   
**ultimately have**  $qv = s$   
**by**  $simp$   
**then show**  $?thesis$

```

    using ‹d-reaches M2 (initial M2) (map fst vs') qv› by blast
  qed

  then show False by (meson assms(14) assms(13) perm-inputs states-xs vs'-def)
  qed

  moreover have  $\bigcup (image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs)))$ 
    = insert s' ( $\bigcup (image (io-targets M1 (initial M1)) (R M2 s vs xs))$ )
    using s'-def by simp

  moreover have finite ( $\bigcup (image (io-targets M1 (initial M1)) (R M2 s vs xs))$ )
  proof
    show finite (R M2 s vs xs)
      using finite-R by simp
    show  $\bigwedge a. a \in R M2 s vs xs \implies finite (io-targets M1 (initial M1) a)$ 
    proof -
      fix a assume a  $\in R M2 s vs xs$ 
      then have prefix a (vs@xs)
        by auto
      then have a  $\in L M1$ 
        using language-state-prefix by (metis IntD1 assms(1) prefix-def)
      then obtain p where io-targets M1 (initial M1) a = {p}
        using assms(2) io-targets-observable-singleton-ob by metis
      then show finite (io-targets M1 (initial M1) a)
        by simp
    qed
  qed

  ultimately have card ( $\bigcup (image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs)))$ )
    = Suc (card ( $\bigcup (image (io-targets M1 (initial M1)) (R M2 s vs xs))$ ))
    by (metis (no-types) card-insert-disjoint)

  moreover have card ( $\bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))$ )
    = card ( $\bigcup (image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs)))$ )
    using vs'-def by simp

  ultimately have card ( $\bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))$ )
    = Suc (card ( $\bigcup (image (io-targets M1 (initial M1)) (R M2 s vs xs))$ ))
    by linarith

  then have card ( $\bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))$ )
    = Suc (card (R M2 s vs xs))
    using R-count[of vs xs M1 M2 s FAIL PM q2 q1 tr] assms(1,10,11,2-9) by linarith

  moreover have card (RP M2 s vs xs V'') = Suc (card (R M2 s vs xs))
    using vs'-def by (metis card-insert-if finite-R)

  ultimately show ?thesis
    by linarith
  qed
  qed

lemma RP-state-component-2 :
  assumes io  $\in (RP M2 s vs xs V'')$ 
  and observable M2
  shows io-targets M2 (initial M2) io = {s}
  by (metis (mono-tags, lifting) RP.simps R-state-component-2 Un-iff assms mem-Collect-eq)

lemma RP-io-targets-split :
  assumes (vs @ xs)  $\in L M1 \cap L M2$ 
  and observable M1
  and observable M2
  and well-formed M1

```

**and** *well-formed M2*  
**and** *productF M2 M1 FAIL PM*  
**and** *is-det-state-cover M2 V*  
**and**  $V'' \in \text{Perm } V \ M1$   
**and**  $io \in RP \ M2 \ s \ vs \ xs \ V''$   
**shows** *io-targets PM (initial PM) io*  
 $= io-targets \ M2 \ (initial \ M2) \ io \times io-targets \ M1 \ (initial \ M1) \ io$   
**proof** –  
**have** *RP-cases : RP M2 s vs xs V'' = R M2 s vs xs*  
 $\vee (\exists \ vs' \in V'' . \ vs' \notin R \ M2 \ s \ vs \ xs$   
 $\wedge RP \ M2 \ s \ vs \ xs \ V'' = insert \ vs' \ (R \ M2 \ s \ vs \ xs))$   
**using** *RP-from-R assms by metis*  
**show** *?thesis*  
**proof** (*cases io ∈ R M2 s vs xs*)  
**case** *True*  
**then have** *io-prefix : prefix io (vs @ xs)*  
**by** *auto*  
**then have** *io-lang-subst : io ∈ L M1 ∧ io ∈ L M2*  
**using** *assms(1) unfolding prefix-def by (metis IntE language-state language-state-split)*  
**then have** *io-lang-inter : io ∈ L M1 ∩ L M2*  
**by** *simp*  
**then have** *io-lang-pm : io ∈ L PM*  
**using** *productF-language assms by blast*  
**moreover obtain** *p2 p1 where (p2,p1) ∈ io-targets PM (initial PM) io*  
**by** (*metis assms(2) assms(3) assms(6) calculation insert-absorb insert-ident insert-not-empty*  
*io-targets-observable-singleton-ob observable-productF singleton-insert-inj-eq subrelI*)  
**ultimately have** *targets-pm : io-targets PM (initial PM) io = {(p2,p1)}*  
**using** *assms io-targets-observable-singleton-ex singletonD*  
**by** (*metis observable-productF*)  
**then obtain** *trP where trP-def : target (io || trP) (initial PM) = (p2,p1)*  
 $\wedge path \ PM \ (io \ || \ trP) \ (initial \ PM) \wedge length \ io = length \ trP$   
**proof** –  
**assume** *a1 :  $\bigwedge trP. target (io || trP) (initial PM) = (p2, p1)$*   
 $\wedge path \ PM \ (io \ || \ trP) \ (initial \ PM) \wedge length \ io = length \ trP \implies thesis$   
**have**  $\exists ps. target (io || ps) (initial PM) = (p2, p1) \wedge path \ PM \ (io \ || \ ps) \ (initial \ PM)$   
 $\wedge length \ io = length \ ps$   
**using**  $\langle (p2, p1) \in io-targets \ PM \ (initial \ PM) \ io \rangle$  **by** *auto*  
**then show** *?thesis*  
**using** *a1 by blast*  
**qed**  
**then have** *trP-unique : {tr . path PM (io || tr) (initial PM) ∧ length io = length tr} = {trP}*  
**using** *observable-productF observable-path-unique-ex[of PM io initial PM]*  
*io-lang-pm assms(2) assms(3) assms(7)*  
**proof** –  
**obtain** *pps :: ('d × 'c) list where*  
 $f1: \{ps. path \ PM \ (io \ || \ ps) \ (initial \ PM) \wedge length \ io = length \ ps\} = \{pps\}$   
 $\vee \neg observable \ PM$   
**by** (*metis (no-types)  $\langle \bigwedge thesis. \llbracket observable \ PM; io \in L \ PM; \bigwedge tr. \{t. path \ PM \ (io \ || \ t) \ (initial \ PM) \wedge length \ io = length \ t\} = \{tr\} \implies thesis \rrbracket \implies thesis \rangle$*   
*io-lang-pm)*  
**have** *f2: observable PM*  
**by** (*meson  $\langle observable \ M1 \rangle \langle observable \ M2 \rangle \langle productF \ M2 \ M1 \ FAIL \ PM \rangle observable-productF$* )  
**then have**  $trP \in \{pps\}$   
**using** *f1 trP-def by blast*  
**then show** *?thesis*  
**using** *f2 f1 by force*  
**qed**  
  
**obtain** *trIO2 where trIO2-def : {tr . path M2 (io || tr) (initial M2) ∧ length io = length tr}*  
 $= \{trIO2\}$   
**using** *observable-path-unique-ex[of M2 io initial M2] io-lang-subst assms(3) by blast*  
**obtain** *trIO1 where trIO1-def : {tr . path M1 (io || tr) (initial M1) ∧ length io = length tr}*  
 $= \{trIO1\}$   
**using** *observable-path-unique-ex[of M1 io initial M1] io-lang-subst assms(2) by blast*

```

have path PM (io || trIO2 || trIO1) (initial M2, initial M1)
  ∧ length io = length trIO2 ∧ length trIO2 = length trIO1
proof –
  have f1: path M2 (io || trIO2) (initial M2) ∧ length io = length trIO2
    using trIO2-def by auto
  have f2: path M1 (io || trIO1) (initial M1) ∧ length io = length trIO1
    using trIO1-def by auto
  then have length trIO2 = length trIO1
    using f1 by presburger
  then show ?thesis
    using f2 f1 assms(4) assms(5) assms(6) by blast
qed
then have trP-split : path PM (io || trIO2 || trIO1) (initial PM)
  ∧ length io = length trIO2 ∧ length trIO2 = length trIO1
  using assms(6) by auto
then have trP-zip : trIO2 || trIO1 = trP
  using trP-def trP-unique length-zip by fastforce

have target (io || trIO2) (initial M2) = p2
  ∧ path M2 (io || trIO2) (initial M2)
  ∧ length io = length trIO2
  using trP-zip trP-split assms(6) trP-def trIO2-def by auto
then have p2 ∈ io-targets M2 (initial M2) io
  by auto
then have targets-2 : io-targets M2 (initial M2) io = {p2}
  by (meson assms(3) observable-io-target-is-singleton)

have target (io || trIO1) (initial M1) = p1
  ∧ path M1 (io || trIO1) (initial M1)
  ∧ length io = length trIO1
  using trP-zip trP-split assms(6) trP-def trIO1-def by auto
then have p1 ∈ io-targets M1 (initial M1) io
  by auto
then have targets-1 : io-targets M1 (initial M1) io = {p1}
  by (metis io-lang-sub assms(2) io-targets-observable-singleton-ex singletonD)

have io-targets M2 (initial M2) io × io-targets M1 (initial M1) io = {(p2,p1)}
  using targets-2 targets-1 by simp
then show io-targets PM (initial PM) io
  = io-targets M2 (initial M2) io × io-targets M1 (initial M1) io
  using targets-pm by simp

next
case False
then have io ∉ R M2 s vs xs ∧ RP M2 s vs xs V'' = insert io (R M2 s vs xs)
  using RP-cases assms(9) by (metis insertE)

have io ∈ L M1 using assms(8) perm-language assms(9)
  using False by auto
then obtain s' where s'-def : io-targets M1 (initial M1) io = {s'}
  by (meson assms(2) io-targets-observable-singleton-ob)
then obtain tr1 where tr1-def : target (io || tr1) (initial M1) = s'
  ∧ path M1 (io || tr1) (initial M1) ∧ length tr1 = length io
  by (metis io-targets-elim singletonI)

have io-targets M2 (initial M2) io = {s}
  using assms(9) assms(3) RP-state-component-2 by simp
then obtain tr2 where tr2-def : target (io || tr2) (initial M2) = s
  ∧ path M2 (io || tr2) (initial M2) ∧ length tr2 = length io
  by (metis io-targets-elim singletonI)
then have paths : path M2 (io || tr2) (initial M2) ∧ path M1 (io || tr1) (initial M1)
  using tr1-def by simp

have length io = length tr2

```

```

    using tr2-def by simp
  moreover have length tr2 = length tr1
    using tr1-def tr2-def by simp
  ultimately have path PM (io || tr2 || tr1) (initial M2, initial M1)
    using assms(6) assms(5) assms(4) paths
      productF-path-forward[of io tr2 tr1 M2 M1 FAIL PM initial M2 initial M1]
    by blast

  moreover have target (io || tr2 || tr1) (initial M2, initial M1) = (s,s')
    by (simp add: tr1-def tr2-def)
  moreover have length (tr2 || tr2) = length io
    using tr1-def tr2-def by simp
  moreover have (initial M2, initial M1) = initial PM
    using assms(6) by simp
  ultimately have (s,s') ∈ io-targets PM (initial PM) io
    by (metis io-target-from-path length-zip tr1-def tr2-def)
  moreover have observable PM
    using assms(2) assms(3) assms(6) observable-productF by blast
  then have io-targets PM (initial PM) io = {(s,s')}
    by (meson calculation observable-io-target-is-singleton)

  then show ?thesis
    using ‹io-targets M2 (initial M2) io = {s}› ‹io-targets M1 (initial M1) io = {s'}›
    by simp
qed
qed

```

```

lemma RP-io-targets-finite-M1 :
  assumes (vs @ xs) ∈ L M1 ∩ L M2
  and observable M1
  and is-det-state-cover M2 V
  and V'' ∈ Perm V M1
  shows finite (⋃ (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'')))
  proof
    show finite (RP M2 s vs xs V'') using finite-RP assms(3) assms(4) by simp
    show  $\bigwedge a. a \in RP\ M2\ s\ vs\ xs\ V'' \implies finite\ (io-targets\ M1\ (initial\ M1)\ a)$ 
    proof -
      fix a assume a ∈ RP M2 s vs xs V''

      have RP-cases : RP M2 s vs xs V'' = R M2 s vs xs
        ∨ (∃ vs' ∈ V'' . vs' ∉ R M2 s vs xs
          ∧ RP M2 s vs xs V'' = insert vs' (R M2 s vs xs))

      using RP-from-R assms by metis
      have a ∈ L M1
      proof (cases a ∈ R M2 s vs xs)
        case True
          then have prefix a (vs@xs)
            by auto
          then show a ∈ L M1
            using language-state-prefix by (metis IntD1 assms(1) prefix-def)
        case False
          then have a ∈ V'' ∧ RP M2 s vs xs V'' = insert a (R M2 s vs xs)
            using RP-cases ‹a ∈ RP M2 s vs xs V''› by (metis insertE)
          then show a ∈ L M1
            by (meson assms(4) perm-language)
      qed
      then obtain p where io-targets M1 (initial M1) a = {p}
        using assms(2) io-targets-observable-singleton-ob by metis
      then show finite (io-targets M1 (initial M1) a)
        by simp
    qed
  qed

```

qed

**lemma** *RP-io-targets-finite-PM* :

**assumes**  $(vs @ xs) \in L M1 \cap L M2$   
**and** *observable M1*  
**and** *observable M2*  
**and** *well-formed M1*  
**and** *well-formed M2*  
**and** *productF M2 M1 FAIL PM*  
**and** *is-det-state-cover M2 V*  
**and**  $V'' \in Perm V M1$

**shows**  $finite (\bigcup (image (io-targets PM (initial PM)) (RP M2 s vs xs V'')))$

**proof** –

**have**  $\forall io \in RP M2 s vs xs V'' . io-targets PM (initial PM) io$   
 $= \{s\} \times io-targets M1 (initial M1) io$

**proof**

**fix** *io* **assume**  $io \in RP M2 s vs xs V''$

**then have**  $io-targets PM (initial PM) io$   
 $= io-targets M2 (initial M2) io \times io-targets M1 (initial M1) io$

**using** *assms RP-io-targets-split[of vs xs M1 M2 FAIL PM V V'' io s]* **by** *simp*

**moreover have**  $io-targets M2 (initial M2) io = \{s\}$

**using**  $\langle io \in RP M2 s vs xs V'' \rangle$  *assms(3) RP-state-component-2[of io M2 s vs xs V'']*  
**by** *blast*

**ultimately show**  $io-targets PM (initial PM) io = \{s\} \times io-targets M1 (initial M1) io$   
**by** *auto*

qed

**then have**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s vs xs V''))$   
 $= \bigcup (image (\lambda io . \{s\} \times io-targets M1 (initial M1) io) (RP M2 s vs xs V''))$

**by** *simp*

**moreover have**  $\bigcup (image (\lambda io . \{s\} \times io-targets M1 (initial M1) io) (RP M2 s vs xs V''))$   
 $= \{s\} \times \bigcup (image (\lambda io . io-targets M1 (initial M1) io) (RP M2 s vs xs V''))$

**by** *blast*

**ultimately have**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s vs xs V''))$   
 $= \{s\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))$

**by** *auto*

**moreover have**  $finite (\{s\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'')))$   
**using** *assms(1,2,7,8) RP-io-targets-finite-M1[of vs xs M1 M2 V V'' s]* **by** *simp*

**ultimately show** *?thesis*

**by** *simp*

qed

**lemma** *RP-union-card-is-suffix-length* :

**assumes** *OFSM M2*  
**and**  $io@xs \in L M2$   
**and** *is-det-state-cover M2 V*  
**and**  $V'' \in Perm V M1$

**shows**  $\bigwedge q . card (R M2 q io xs) \leq card (RP M2 q io xs V'')$   
 $sum (\lambda q . card (RP M2 q io xs V'')) (nodes M2) \geq length xs$

**proof** –

**have**  $sum (\lambda q . card (R M2 q io xs)) (nodes M2) = length xs$   
**using** *R-union-card-is-suffix-length[OF assms(1,2)]* **by** *assumption*

**show**  $\bigwedge q . card (R M2 q io xs) \leq card (RP M2 q io xs V'')$

**by** *(metis RP-from-R assms(3) assms(4) card-insert-le eq-iff finite-R)*

**show**  $sum (\lambda q . card (RP M2 q io xs V'')) (nodes M2) \geq length xs$

**by** *(metis (no-types, lifting)  $\langle \sum q \in nodes M2 . card (R M2 q io xs) = length xs \rangle$  sum-mono)*  
 $\langle \bigwedge q . card (R M2 q io xs) \leq card (RP M2 q io xs V'') \rangle$  *sum-mono*

qed

**lemma** *RP-state-repetition-distribution-productF* :

**assumes** *OFSM M2*  
**and** *OFSM M1*  
**and**  $(card (nodes M2) * m) \leq length xs$   
**and**  $card (nodes M1) \leq m$   
**and**  $vs@xs \in L M2 \cap L M1$

**and**  $is-det-state-cover\ M2\ V$   
**and**  $V'' \in Perm\ V\ M1$   
**shows**  $\exists q \in nodes\ M2 . card\ (RP\ M2\ q\ vs\ xs\ V'') > m$   
**proof** –  
**have**  $finite\ (nodes\ M1)$   
 $finite\ (nodes\ M2)$   
**using**  $assms(1,2)$  **by**  $auto$   
**then have**  $card(nodes\ M2 \times nodes\ M1) = card\ (nodes\ M2) * card\ (nodes\ M1)$   
**using**  $card-cartesian-product$  **by**  $blast$   
  
**have**  $nodes\ (product\ M2\ M1) \subseteq nodes\ M2 \times nodes\ M1$   
**using**  $product-nodes$  **by**  $auto$   
  
**have**  $card\ (nodes\ (product\ M2\ M1)) \leq card\ (nodes\ M2) * card\ (nodes\ M1)$   
**by**  $(metis\ (no-types)\ \langle card\ (nodes\ M2 \times nodes\ M1) = |M2| * |M1| \rangle\ \langle finite\ (nodes\ M1) \rangle$   
 $\langle finite\ (nodes\ M2) \rangle\ \langle nodes\ (product\ M2\ M1) \subseteq nodes\ M2 \times nodes\ M1 \rangle$   
 $card-mono\ finite-cartesian-product)$   
  
**have**  $(\forall q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) = m) \vee (\exists q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) > m)$   
**proof**  $(rule\ ccontr)$   
**assume**  $\neg ((\forall q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) = m) \vee (\exists q \in nodes\ M2 . m < card\ (R\ M2\ q\ vs\ xs)))$   
  
**then have**  $\forall q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) \leq m$   
**by**  $auto$   
**moreover obtain**  $q'$  **where**  $q' \in nodes\ M2\ card\ (R\ M2\ q'\ vs\ xs) < m$   
**using**  $\langle \neg ((\forall q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) = m) \vee (\exists q \in nodes\ M2 . m < card\ (R\ M2\ q\ vs\ xs))) \rangle$   
 $nat-neq-iff$   
**by**  $blast$   
  
**have**  $sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2)$   
 $= sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2 - \{q'\})$   
 $+ sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ \{q'\}$   
**using**  $\langle q' \in nodes\ M2 \rangle$   
**by**  $(meson\ \langle finite\ (nodes\ M2) \rangle\ empty-subsetI\ insert-subset\ sum.subset-diff)$   
**moreover have**  $sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2 - \{q'\})$   
 $\leq sum\ (\lambda q . m)\ (nodes\ M2 - \{q'\})$   
**using**  $\langle \forall q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs) \leq m \rangle$   
**by**  $(meson\ sum-mono\ DiffD1)$   
**moreover have**  $sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ \{q'\} < m$   
**using**  $\langle card\ (R\ M2\ q'\ vs\ xs) < m \rangle$  **by**  $auto$   
**ultimately have**  $sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2) < sum\ (\lambda q . m)\ (nodes\ M2)$   
**proof** –  
**have**  $\forall C\ c\ f . infinite\ C \vee (c::c) \notin C \vee sum\ f\ C = (f\ c::nat) + sum\ f\ (C - \{c\})$   
**by**  $(meson\ sum.remove)$   
**then have**  $(\sum c \in nodes\ M2 . m) = m + (\sum c \in nodes\ M2 - \{q'\} . m)$   
**using**  $\langle finite\ (nodes\ M2) \rangle\ \langle q' \in nodes\ M2 \rangle$  **by**  $blast$   
**then show**  $?thesis$   
**using**  $\langle (\sum q \in nodes\ M2 - \{q'\} . card\ (R\ M2\ q\ vs\ xs)) \leq (\sum q \in nodes\ M2 - \{q'\} . m) \rangle$   
 $\langle (\sum q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs)) = (\sum q \in nodes\ M2 - \{q'\} . card\ (R\ M2\ q\ vs\ xs))$   
 $+ (\sum q \in \{q'\} . card\ (R\ M2\ q\ vs\ xs)) \rangle$   
 $\langle (\sum q \in \{q'\} . card\ (R\ M2\ q\ vs\ xs)) < m \rangle$   
**by**  $linarith$   
**qed**  
  
**moreover have**  $sum\ (\lambda q . m)\ (nodes\ M2) \leq card\ (nodes\ M2) * m$   
**using**  $assms(2)$  **by**  $auto$   
**ultimately have**  $sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2) < card\ (nodes\ M2) * m$   
**by**  $presburger$   
  
**moreover have**  $Suc\ (card\ (nodes\ M2) * m) \leq sum\ (\lambda q . card\ (R\ M2\ q\ vs\ xs))\ (nodes\ M2)$   
**using**  $R-union-card-is-suffix-length[OF\ assms(1),\ of\ vs\ xs]\ assms(5,3)$   
**by**  $(metis\ Int-iff\ \langle vs\ @\ xs \in L\ M2 \implies (\sum q \in nodes\ M2 . card\ (R\ M2\ q\ vs\ xs)) = length\ xs \rangle$   
 $\langle vs\ @\ xs \in L\ M2 \cap L\ M1 \rangle\ \langle |M2| * m \leq length\ xs \rangle\ calculation\ less-eq-Suc-le\ not-less-eq-eq)$

```

ultimately show False by simp
qed
then show ?thesis
proof
let ?q = initial M2

assume  $\forall q \in \text{nodes } M2. \text{card } (R \ M2 \ q \ \text{vs } \text{xs}) = m$ 
then have  $\text{card } (R \ M2 \ ?q \ \text{vs } \text{xs}) = m$ 
  by auto

have  $\square \in V''$ 
  by (meson assms(6) assms(7) perm-empty)
then have  $\square \in RP \ M2 \ ?q \ \text{vs } \text{xs} \ V''$ 
  by auto
have  $\square \notin R \ M2 \ ?q \ \text{vs } \text{xs}$ 
  by auto
have  $\text{card } (RP \ M2 \ ?q \ \text{vs } \text{xs} \ V'') \geq \text{card } (R \ M2 \ ?q \ \text{vs } \text{xs})$ 
  using finite-R[of M2 ?q vs xs] finite-RP[OF assms(6,7),of ?q vs xs] unfolding RP.simps
  by (simp add: card-mono)

have  $\text{card } (RP \ M2 \ ?q \ \text{vs } \text{xs} \ V'') > \text{card } (R \ M2 \ ?q \ \text{vs } \text{xs})$ 
proof -
  have f1:  $\forall n \ \text{na}. (\neg (n::\text{nat}) \leq \text{na} \vee n = \text{na}) \vee n < \text{na}$ 
    by (meson le-neq-trans)
  have  $RP \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs} \ V'' \neq R \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs}$ 
    using  $\langle \square \in RP \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs} \ V'' \rangle \langle \square \notin R \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs} \rangle$  by blast
  then show ?thesis
    using f1 by (metis (no-types) RP-from-R
       $\langle \text{card } (R \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs}) \leq \text{card } (RP \ M2 \ (\text{initial } M2) \ \text{vs } \text{xs} \ V'') \rangle$ 
      assms(6) assms(7) card-insert-disjoint finite-R le-simps(2))
qed

then show ?thesis
  using  $\langle \text{card } (R \ M2 \ ?q \ \text{vs } \text{xs}) = m \rangle$ 
  by blast
next
assume  $\exists q \in \text{nodes } M2. m < \text{card } (R \ M2 \ q \ \text{vs } \text{xs})$ 
then obtain q where  $q \in \text{nodes } M2 \ m < \text{card } (R \ M2 \ q \ \text{vs } \text{xs})$ 
  by blast
moreover have  $\text{card } (RP \ M2 \ q \ \text{vs } \text{xs} \ V'') \geq \text{card } (R \ M2 \ q \ \text{vs } \text{xs})$ 
  using finite-R[of M2 q vs xs] finite-RP[OF assms(6,7),of q vs xs] unfolding RP.simps
  by (simp add: card-mono)
ultimately have  $m < \text{card } (RP \ M2 \ q \ \text{vs } \text{xs} \ V'')$ 
  by simp

show ?thesis
  using  $\langle q \in \text{nodes } M2 \rangle \langle m < \text{card } (RP \ M2 \ q \ \text{vs } \text{xs} \ V'') \rangle$  by blast
qed
qed

```

## 4.5 Conditions for the result of LB to be a valid lower bound

The following predicates describe the assumptions necessary to show that the value calculated by LB is a lower bound on the number of states of a given FSM.

```

fun Prereq :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  $\Rightarrow$  ('in  $\times$  'out) list
   $\Rightarrow$  ('in  $\times$  'out) list  $\Rightarrow$  'in list set  $\Rightarrow$  'state1 set  $\Rightarrow$  ('in, 'out) ATC set
   $\Rightarrow$  ('in  $\times$  'out) list set  $\Rightarrow$  bool

where
Prereq M2 M1 vs xs T S  $\Omega$  V'' = (
  (finite T)
   $\wedge$  (vs @ xs)  $\in$  L M2  $\cap$  L M1
   $\wedge$  S  $\subseteq$  nodes M2
   $\wedge$  ( $\forall s1 \in S. \forall s2 \in S. s1 \neq s2$ 
     $\longrightarrow$  ( $\forall io1 \in RP \ M2 \ s1 \ \text{vs } \text{xs} \ V''.$ 
       $\forall io2 \in RP \ M2 \ s2 \ \text{vs } \text{xs} \ V''.$ 

```

$B M1 io1 \Omega \neq B M1 io2 \Omega )))$

```

fun Rep-Pre :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  $\Rightarrow$  ('in  $\times$  'out) list
   $\Rightarrow$  ('in  $\times$  'out) list  $\Rightarrow$  bool where
  Rep-Pre M2 M1 vs xs = ( $\exists$  xs1 xs2 . prefix xs1 xs2  $\wedge$  prefix xs2 xs  $\wedge$  xs1  $\neq$  xs2
     $\wedge$  ( $\exists$  s2 . io-targets M2 (initial M2) (vs @ xs1) = {s2}
       $\wedge$  io-targets M2 (initial M2) (vs @ xs2) = {s2})
     $\wedge$  ( $\exists$  s1 . io-targets M1 (initial M1) (vs @ xs1) = {s1}
       $\wedge$  io-targets M1 (initial M1) (vs @ xs2) = {s1}))

```

```

fun Rep-Cov :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  $\Rightarrow$  ('in  $\times$  'out) list set
   $\Rightarrow$  ('in  $\times$  'out) list  $\Rightarrow$  ('in  $\times$  'out) list  $\Rightarrow$  bool where
  Rep-Cov M2 M1 V'' vs xs = ( $\exists$  xs' vs' . xs'  $\neq$  []  $\wedge$  prefix xs' xs  $\wedge$  vs'  $\in$  V''
     $\wedge$  ( $\exists$  s2 . io-targets M2 (initial M2) (vs @ xs') = {s2}
       $\wedge$  io-targets M2 (initial M2) (vs') = {s2})
     $\wedge$  ( $\exists$  s1 . io-targets M1 (initial M1) (vs @ xs') = {s1}
       $\wedge$  io-targets M1 (initial M1) (vs') = {s1}))

```

**lemma** *distinctness-via-Rep-Pre* :

```

assumes  $\neg$  Rep-Pre M2 M1 vs xs
and productF M2 M1 FAIL PM
and observable M1
and observable M2
and io-targets PM (initial PM) vs = {(q2,q1)}
and path PM (xs || tr) (q2,q1)
and length xs = length tr
and (vs @ xs)  $\in$  L M1  $\cap$  L M2
and well-formed M1
and well-formed M2

```

**shows** *distinct* (states (xs || tr) (q2, q1))

**proof** (rule ccontr)

**assume** *assm* :  $\neg$  *distinct* (states (xs || tr) (q2, q1))

**then obtain** *i1 i2* **where** *index-def* :

```

  i1  $\neq$  0
   $\wedge$  i1  $\neq$  i2
   $\wedge$  i1 < length (states (xs || tr) (q2, q1))
   $\wedge$  i2 < length (states (xs || tr) (q2, q1))
   $\wedge$  (states (xs || tr) (q2, q1)) ! i1 = (states (xs || tr) (q2, q1)) ! i2
by (metis distinct-conv-nth)

```

**then have** length xs > 0 **by** auto

```

let ?xs1 = take (Suc i1) xs
let ?xs2 = take (Suc i2) xs
let ?tr1 = take (Suc i1) tr
let ?tr2 = take (Suc i2) tr
let ?st = (states (xs || tr) (q2, q1)) ! i1

```

**have** *obs-PM* : observable PM

**using** observable-productF *assms*(2) *assms*(3) *assms*(4) **by** blast

**have** *initial PM* = (initial M2, initial M1)

**using** *assms*(2) **by** simp

**moreover have** vs  $\in$  L M2 vs  $\in$  L M1

**using** *assms*(8) *language-state-prefix* **by** auto

**ultimately have** io-targets M1 (initial M1) vs = {q1} io-targets M2 (initial M2) vs = {q2}

**using** productF-path-io-targets[of M2 M1 FAIL PM initial M2 initial M1 vs q2 q1]

**by** (metis FSM.nodes.initial *assms*(2) *assms*(3) *assms*(4) *assms*(5) *assms*(9) *assms*(10) io-targets-observable-singleton-ex singletonD)+

— paths for ?xs1

**have** (states (xs || tr) (q2, q1)) ! *i1*  $\in$  io-targets PM (q2, q1) ?xs1

```

  by (metis ‹0 < length xs› assms(6) assms(7) index-def map-snd-zip states-alt-def
    states-index-io-target)
then have io-targets PM (q2, q1) ?xs1 = {?st}
  using obs-PM by (meson observable-io-target-is-singleton)

have path PM (?xs1 || ?tr1) (q2, q1)
  by (metis FSM.path-append-elim append-take-drop-id assms(6) assms(7) length-take zip-append)
then have path PM (?xs1 || map fst ?tr1 || map snd ?tr1) (q2, q1)
  by auto

have vs @ ?xs1 ∈ L M2
  by (metis (no-types) IntD2 append-assoc append-take-drop-id assms(8) language-state-prefix)
then obtain q2' where io-targets M2 (initial M2) (vs @ ?xs1) = {q2'}
  using io-targets-observable-singleton-ob[of M2 vs @ ?xs1 initial M2] assms(4) by auto
then have q2' ∈ io-targets M2 q2 ?xs1
  using assms(4) ‹io-targets M2 (initial M2) vs = {q2'}›
    observable-io-targets-split[of M2 initial M2 vs ?xs1 q2' q2]
  by simp
then have ?xs1 ∈ language-state M2 q2
  by auto
moreover have length ?xs1 = length (map snd ?tr1)
  using assms(7) by auto
moreover have length (map fst ?tr1) = length (map snd ?tr1)
  by auto
moreover have q2 ∈ nodes M2
  using ‹io-targets M2 (initial M2) vs = {q2'}› io-targets-nodes
  by (metis FSM.nodes.initial insertI1)
moreover have q1 ∈ nodes M1
  using ‹io-targets M1 (initial M1) vs = {q1'}› io-targets-nodes
  by (metis FSM.nodes.initial insertI1)
ultimately have
  path M1 (?xs1 || map snd ?tr1) q1
  path M2 (?xs1 || map fst ?tr1) q2
  target (?xs1 || map snd ?tr1) q1 = snd (target (?xs1 || map fst ?tr1 || map snd ?tr1) (q2, q1))
  target (?xs1 || map fst ?tr1) q2 = fst (target (?xs1 || map fst ?tr1 || map snd ?tr1) (q2, q1))
  using assms(2) assms(9) assms(10) ‹path PM (?xs1 || map fst ?tr1 || map snd ?tr1) (q2, q1)›
    assms(4)
    productF-path-reverse-ob-2[of ?xs1 map fst ?tr1 map snd ?tr1 M2 M1 FAIL PM q2 q1]
  by simp+
moreover have target (?xs1 || map fst ?tr1 || map snd ?tr1) (q2, q1) = ?st
  by (metis (no-types) index-def scan-nth take-zip zip-map-fst-snd)
ultimately have
  target (?xs1 || map snd ?tr1) q1 = snd ?st
  target (?xs1 || map fst ?tr1) q2 = fst ?st
  by simp+

— paths for ?xs2

have (states (xs || tr) (q2, q1)) ! i2 ∈ io-targets PM (q2, q1) ?xs2
  by (metis ‹0 < length xs› assms(6) assms(7) index-def map-snd-zip states-alt-def states-index-io-target)
then have io-targets PM (q2, q1) ?xs2 = {?st}
  using obs-PM by (metis index-def observable-io-target-is-singleton)

have path PM (?xs2 || ?tr2) (q2, q1)
  by (metis FSM.path-append-elim append-take-drop-id assms(6) assms(7) length-take zip-append)
then have path PM (?xs2 || map fst ?tr2 || map snd ?tr2) (q2, q1)
  by auto

have vs @ ?xs2 ∈ L M2
  by (metis (no-types) IntD2 append-assoc append-take-drop-id assms(8) language-state-prefix)
then obtain q2'' where io-targets M2 (initial M2) (vs @ ?xs2) = {q2''}
  using io-targets-observable-singleton-ob[of M2 vs @ ?xs2 initial M2] assms(4)
  by auto
then have q2'' ∈ io-targets M2 q2 ?xs2
  using assms(4) ‹io-targets M2 (initial M2) vs = {q2''}›
    observable-io-targets-split[of M2 initial M2 vs ?xs2 q2'' q2]

```

```

by simp
then have ?xs2 ∈ language-state M2 q2
  by auto
moreover have length ?xs2 = length (map snd ?tr2) using assms(7)
  by auto
moreover have length (map fst ?tr2) = length (map snd ?tr2)
  by auto
moreover have q2 ∈ nodes M2
  using ⟨io-targets M2 (initial M2) vs = {q2}⟩ io-targets-nodes
  by (metis FSM.nodes.initial insertI1)
moreover have q1 ∈ nodes M1
  using ⟨io-targets M1 (initial M1) vs = {q1}⟩ io-targets-nodes
  by (metis FSM.nodes.initial insertI1)
ultimately have
  path M1 (?xs2 || map snd ?tr2) q1
  path M2 (?xs2 || map fst ?tr2) q2
  target (?xs2 || map snd ?tr2) q1 = snd(target (?xs2 || map fst ?tr2 || map snd ?tr2) (q2,q1))
  target (?xs2 || map fst ?tr2) q2 = fst(target (?xs2 || map fst ?tr2 || map snd ?tr2) (q2,q1))
  using assms(2) assms(9) assms(10) ⟨path PM (?xs2 || map fst ?tr2 || map snd ?tr2) (q2,q1)⟩
  assms(4)
  productF-path-reverse-ob-2[of ?xs2 map fst ?tr2 map snd ?tr2 M2 M1 FAIL PM q2 q1]
  by simp+
moreover have target (?xs2 || map fst ?tr2 || map snd ?tr2) (q2,q1) = ?st
  by (metis (no-types) index-def scan-nth take-zip zip-map-fst-snd)
ultimately have
  target (?xs2 || map snd ?tr2) q1 = snd ?st
  target (?xs2 || map fst ?tr2) q2 = fst ?st
  by simp+

```

```

have io-targets M1 q1 ?xs1 = {snd ?st}
  using ⟨path M1 (?xs1 || map snd ?tr1) q1⟩ ⟨target (?xs1 || map snd ?tr1) q1 = snd ?st⟩
  ⟨length ?xs1 = length (map snd ?tr1)⟩ assms(3) obs-target-is-io-targets[of M1 ?xs1
  map snd ?tr1 q1]
  by simp
then have tgt-1-1 : io-targets M1 (initial M1) (vs @ ?xs1) = {snd ?st}
  by (meson ⟨io-targets M1 (initial M1) vs = {q1}⟩ assms(3) observable-io-targets-append)

```

```

have io-targets M2 q2 ?xs1 = {fst ?st}
  using ⟨path M2 (?xs1 || map fst ?tr1) q2⟩ ⟨target (?xs1 || map fst ?tr1) q2 = fst ?st⟩
  ⟨length ?xs1 = length (map snd ?tr1)⟩ assms(4)
  obs-target-is-io-targets[of M2 ?xs1 map fst ?tr1 q2]
  by simp
then have tgt-1-2 : io-targets M2 (initial M2) (vs @ ?xs1) = {fst ?st}
  by (meson ⟨io-targets M2 (initial M2) vs = {q2}⟩ assms(4) observable-io-targets-append)

```

```

have io-targets M1 q1 ?xs2 = {snd ?st}
  using ⟨path M1 (?xs2 || map snd ?tr2) q1⟩ ⟨target (?xs2 || map snd ?tr2) q1 = snd ?st⟩
  ⟨length ?xs2 = length (map snd ?tr2)⟩ assms(3)
  obs-target-is-io-targets[of M1 ?xs2 map snd ?tr2 q1]
  by simp
then have tgt-2-1 : io-targets M1 (initial M1) (vs @ ?xs2) = {snd ?st}
  by (meson ⟨io-targets M1 (initial M1) vs = {q1}⟩ assms(3) observable-io-targets-append)

```

```

have io-targets M2 q2 ?xs2 = {fst ?st}
  using ⟨path M2 (?xs2 || map fst ?tr2) q2⟩ ⟨target (?xs2 || map fst ?tr2) q2 = fst ?st⟩
  ⟨length ?xs2 = length (map snd ?tr2)⟩ assms(4)
  obs-target-is-io-targets[of M2 ?xs2 map fst ?tr2 q2]
  by simp
then have tgt-2-2 : io-targets M2 (initial M2) (vs @ ?xs2) = {fst ?st}
  by (meson ⟨io-targets M2 (initial M2) vs = {q2}⟩ assms(4) observable-io-targets-append)

```

```

have ?xs1 ≠ [] using ⟨0 < length xs⟩
  by auto
have prefix ?xs1 xs

```

```

  using take-is-prefix by blast
have prefix ?xs2 xs
  using take-is-prefix by blast
have ?xs1 ≠ ?xs2
proof -
  have f1: ∀ n na. ¬ n < na ∨ Suc n ≤ na
    by presburger
  have f2: Suc i1 ≤ length xs
    using index-def by force
  have Suc i2 ≤ length xs
    using f1 by (metis index-def length-take map-snd-zip-take min-less-iff-conj states-alt-def)
  then show ?thesis
    using f2 by (metis (no-types) index-def length-take min.absorb2 nat.simps(1))
qed
have Rep-Pre M2 M1 vs xs
proof (cases length ?xs1 < length ?xs2)
  case True
  then have prefix ?xs1 ?xs2
    by (meson ⟨prefix (take (Suc i1) xs) xs⟩ ⟨prefix (take (Suc i2) xs) xs⟩ leD prefix-length-le
      prefix-same-cases)
  show ?thesis
    by (meson Rep-Pre.elims(3) ⟨prefix (take (Suc i1) xs) (take (Suc i2) xs)⟩
      ⟨prefix (take (Suc i2) xs) xs⟩ ⟨take (Suc i1) xs ≠ take (Suc i2) xs⟩
      tgt-1-1 tgt-1-2 tgt-2-1 tgt-2-2)
  next
  case False
  moreover have length ?xs1 ≠ length ?xs2
    by (metis (no-types) ⟨take (Suc i1) xs ≠ take (Suc i2) xs⟩ append-eq-conv-conj
      append-take-drop-id)
  ultimately have length ?xs2 < length ?xs1
    by auto
  then have prefix ?xs2 ?xs1
    using ⟨prefix (take (Suc i1) xs) xs⟩ ⟨prefix (take (Suc i2) xs) xs⟩ less-imp-le-nat
      prefix-length-prefix
    by blast
  show ?thesis
    by (metis Rep-Pre.elims(3) ⟨prefix (take (Suc i1) xs) xs⟩
      ⟨prefix (take (Suc i2) xs) (take (Suc i1) xs)⟩ ⟨take (Suc i1) xs ≠ take (Suc i2) xs⟩
      tgt-1-1 tgt-1-2 tgt-2-1 tgt-2-2)
qed

then show False
  using assms(1) by simp
qed

```

```

lemma RP-count-via-Rep-Cov :
  assumes (vs @ xs) ∈ L M1 ∩ L M2
  and observable M1
  and observable M2
  and well-formed M1
  and well-formed M2
  and s ∈ nodes M2
  and productF M2 M1 FAIL PM
  and io-targets PM (initial PM) vs = {(q2,q1)}
  and path PM (xs || tr) (q2,q1)
  and length xs = length tr
  and distinct (states (xs || tr) (q2,q1))
  and is-det-state-cover M2 V
  and V'' ∈ Perm V M1
  and ¬ Rep-Cov M2 M1 V'' vs xs
shows card (⋃ (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))) = card (RP M2 s vs xs V'')
proof -

```

**have**  $RP\text{-cases} : RP\ M2\ s\ vs\ xs\ V'' = R\ M2\ s\ vs\ xs$   
 $\vee (\exists\ vs' \in V'' . vs' \notin R\ M2\ s\ vs\ xs$   
 $\wedge RP\ M2\ s\ vs\ xs\ V'' = insert\ vs' (R\ M2\ s\ vs\ xs))$   
**using**  $RP\text{-from-}R\ assms$  **by**  $metis$   
**show**  $?thesis$   
**proof** ( $cases\ RP\ M2\ s\ vs\ xs\ V'' = R\ M2\ s\ vs\ xs$ )  
**case**  $True$   
**then show**  $?thesis$   
**using**  $R\text{-count}\ assms$  **by**  $metis$   
**next**  
**case**  $False$   
**then obtain**  $vs'$  **where**  $vs'\text{-def} : vs' \in V''$   
 $\wedge vs' \notin R\ M2\ s\ vs\ xs$   
 $\wedge RP\ M2\ s\ vs\ xs\ V'' = insert\ vs' (R\ M2\ s\ vs\ xs)$   
**using**  $RP\text{-cases}$  **by**  $auto$

**have**  $state\text{-component-}2 : \forall\ io \in (R\ M2\ s\ vs\ xs) . io\text{-targets}\ M2\ (initial\ M2)\ io = \{s\}$   
**proof**  
**fix**  $io$  **assume**  $io \in R\ M2\ s\ vs\ xs$   
**then have**  $s \in io\text{-targets}\ M2\ (initial\ M2)\ io$   
**by**  $auto$   
**moreover have**  $io \in language\text{-state}\ M2\ (initial\ M2)$   
**using**  $calculation$  **by**  $auto$   
**ultimately show**  $io\text{-targets}\ M2\ (initial\ M2)\ io = \{s\}$   
**using**  $assms(3)\ io\text{-targets-observable-singleton-ex}$  **by** ( $metis\ singletonD$ )  
**qed**

**have**  $vs' \in L\ M1$   
**using**  $assms(13)\ perm\text{-language}\ vs'\text{-def}$  **by**  $blast$   
**then obtain**  $s'$  **where**  $s'\text{-def} : io\text{-targets}\ M1\ (initial\ M1)\ vs' = \{s'\}$   
**by** ( $meson\ assms(2)\ io\text{-targets-observable-singleton-ob}$ )

**moreover have**  $s' \notin \bigcup (image\ (io\text{-targets}\ M1\ (initial\ M1))\ (R\ M2\ s\ vs\ xs))$   
**proof** ( $rule\ ccontr$ )  
**assume**  $\neg s' \notin \bigcup (io\text{-targets}\ M1\ (initial\ M1)\ 'R\ M2\ s\ vs\ xs)$   
**then obtain**  $xs'$  **where**  $xs'\text{-def} : vs\ @\ xs' \in R\ M2\ s\ vs\ xs$   
 $\wedge s' \in io\text{-targets}\ M1\ (initial\ M1)\ (vs\ @\ xs')$   
**proof** –  
**assume**  $a1 : \bigwedge xs' . vs\ @\ xs' \in R\ M2\ s\ vs\ xs$   
 $\wedge s' \in io\text{-targets}\ M1\ (initial\ M1)\ (vs\ @\ xs') \implies thesis$   
**obtain**  $pps :: ('a \times 'b)\ list\ set \Rightarrow (('a \times 'b)\ list \Rightarrow 'c\ set) \Rightarrow 'c \Rightarrow ('a \times 'b)\ list$   
**where**  
 $\forall x0\ x1\ x2. (\exists v3. v3 \in x0 \wedge x2 \in x1\ v3) = (pps\ x0\ x1\ x2 \in x0 \wedge x2 \in x1\ (pps\ x0\ x1\ x2))$   
**by**  $moura$   
**then have**  $f2 : pps\ (R\ M2\ s\ vs\ xs)\ (io\text{-targets}\ M1\ (initial\ M1))\ s' \in R\ M2\ s\ vs\ xs$   
 $\wedge s' \in io\text{-targets}\ M1\ (initial\ M1)$   
 $(pps\ (R\ M2\ s\ vs\ xs)\ (io\text{-targets}\ M1\ (initial\ M1))\ s')$   
**using**  $\langle \neg s' \notin \bigcup (io\text{-targets}\ M1\ (initial\ M1)\ 'R\ M2\ s\ vs\ xs) \rangle$  **by**  $blast$   
**then have**  $\exists ps. pps\ (R\ M2\ s\ vs\ xs)\ (io\text{-targets}\ M1\ (initial\ M1))\ s' = vs\ @\ ps \wedge ps \neq []$   
 $\wedge prefix\ ps\ xs \wedge s \in io\text{-targets}\ M2\ (initial\ M2)\ (vs\ @\ ps)$   
**by**  $simp$   
**then show**  $?thesis$   
**using**  $f2\ a1$  **by** ( $metis\ (no\text{-types})$ )  
**qed**

**have**  $vs\ @\ xs' \in L\ M1$   
**using**  $xs'\text{-def}$  **by**  $blast$   
**then have**  $io\text{-targets}\ M1\ (initial\ M1)\ (vs@xs') = \{s'\}$   
**by** ( $metis\ assms(2)\ io\text{-targets-observable-singleton-ob}\ singletonD\ xs'\text{-def}$ )  
**moreover have**  $io\text{-targets}\ M1\ (initial\ M1)\ (vs') = \{s'\}$   
**using**  $s'\text{-def}$  **by**  $blast$   
**moreover have**  $io\text{-targets}\ M2\ (initial\ M2)\ (vs\ @\ xs') = \{s\}$   
**using**  $state\text{-component-}2\ xs'\text{-def}$  **by**  $blast$   
**moreover have**  $io\text{-targets}\ M2\ (initial\ M2)\ (vs') = \{s\}$   
**by** ( $metis\ (mono\text{-tags},\ lifting)\ RP.simps\ Un\text{-iff}\ insertI1\ mem\text{-Collect-eq}\ vs'\text{-def}$ )  
**moreover have**  $xs' \neq []$

```

    using  $xs'$ -def by simp
  moreover have prefix  $xs' xs$ 
    using  $xs'$ -def by simp
  moreover have  $vs' \in V''$ 
    using  $vs'$ -def by simp
  ultimately have Rep-Cov  $M2 M1 V'' vs xs$ 
    by auto

  then show False
    using assms(14) by simp
qed

moreover have  $\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (\text{insert } vs' (R M2 s vs xs)))$ 
  =  $\text{insert } s' (\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (R M2 s vs xs)))$ 
  using  $s'$ -def by simp

moreover have finite  $(\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (R M2 s vs xs)))$ 
proof
  show finite  $(R M2 s vs xs)$ 
    using finite-R by simp
  show  $\bigwedge a. a \in R M2 s vs xs \implies \text{finite } (io\text{-targets } M1 (\text{initial } M1) a)$ 
  proof -
    fix a assume  $a \in R M2 s vs xs$ 
    then have prefix a  $(vs@xs)$ 
      by auto
    then have  $a \in L M1$ 
      using language-state-prefix by (metis IntD1 assms(1) prefix-def)
    then obtain p where  $io\text{-targets } M1 (\text{initial } M1) a = \{p\}$ 
      using assms(2) io-targets-observable-singleton-ob by metis
    then show finite  $(io\text{-targets } M1 (\text{initial } M1) a)$ 
      by simp
  qed
qed

ultimately have card  $(\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (\text{insert } vs' (R M2 s vs xs))))$ 
  =  $\text{Suc } (\text{card } (\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (R M2 s vs xs))))$ 
  by (metis (no-types) card-insert-disjoint)

moreover have card  $(\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (RP M2 s vs xs V'')))$ 
  =  $\text{card } (\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (\text{insert } vs' (R M2 s vs xs))))$ 
  using  $vs'$ -def by simp

ultimately have card  $(\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (RP M2 s vs xs V'')))$ 
  =  $\text{Suc } (\text{card } (\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (R M2 s vs xs))))$ 
  by linarith

then have card  $(\bigcup (\text{image } (io\text{-targets } M1 (\text{initial } M1)) (RP M2 s vs xs V'')))$ 
  =  $\text{Suc } (\text{card } (R M2 s vs xs))$ 
  using R-count[of vs xs M1 M2 s FAIL PM q2 q1 tr] using assms(1,10,11,2-9)
  by linarith

moreover have card  $(RP M2 s vs xs V'') = \text{Suc } (\text{card } (R M2 s vs xs))$ 
  using  $vs'$ -def by (metis card-insert-if finite-R)

ultimately show ?thesis
  by linarith
qed
qed

lemma RP-count-alt-def :
  assumes  $(vs @ xs) \in L M1 \cap L M2$ 
  and observable M1
  and observable M2
  and well-formed M1

```

```

and well-formed M2
and s ∈ nodes M2
and productF M2 M1 FAIL PM
and io-targets PM (initial PM) vs = {(q2,q1)}
and path PM (xs || tr) (q2,q1)
and length xs = length tr
and ¬ Rep-Pre M2 M1 vs xs
and is-det-state-cover M2 V
and V'' ∈ Perm V M1
and ¬ Rep-Cov M2 M1 V'' vs xs
shows card (⋃ (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))) = card (RP M2 s vs xs V'')
proof -
  have distinct (states (xs || tr) (q2,q1))
    using distinctness-via-Rep-Pre[of M2 M1 vs xs FAIL PM q2 q1 tr] assms by simp
  then show ?thesis
    using RP-count-via-Rep-Cov[of vs xs M1 M2 s FAIL PM q2 q1 tr V V'']
    using assms(1,10,12-14,2-9) by blast
qed

```

## 4.6 Function LB

LB adds together the number of elements in sets calculated via RP for a given set of states and the number of ATC-reaction known to exist but not produced by a state reached by any of the above elements.

```

fun LB :: ('in, 'out, 'state1) FSM ⇒ ('in, 'out, 'state2) FSM
  ⇒ ('in × 'out) list ⇒ ('in × 'out) list ⇒ 'in list set
  ⇒ 'state1 set ⇒ ('in, 'out) ATC set
  ⇒ ('in × 'out) list set ⇒ nat
where
  LB M2 M1 vs xs T S Ω V'' =
    (sum (λ s . card (RP M2 s vs xs V'')) S)
    + card ((D M1 T Ω) -
      {B M1 xs' Ω | xs' s' . s' ∈ S ∧ xs' ∈ RP M2 s' vs xs V''})

```

**lemma** *LB-count-helper-RP-disjoint-and-cards* :

```

assumes (vs @ xs) ∈ L M1 ∩ L M2
and observable M1
and observable M2
and well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
and is-det-state-cover M2 V
and V'' ∈ Perm V M1
and s1 ≠ s2
shows ⋃ (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V''))
  ∩ ⋃ (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'')) = {}
  card (⋃ (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V'')))
  = card (⋃ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V'')))
  card (⋃ (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'')))
  = card (⋃ (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V'')))
proof -
  have ∀ io ∈ RP M2 s1 vs xs V'' . io-targets PM (initial PM) io
    = {s1} × io-targets M1 (initial M1) io
  proof
    fix io assume io ∈ RP M2 s1 vs xs V''
    then have io-targets PM (initial PM) io
      = io-targets M2 (initial M2) io × io-targets M1 (initial M1) io
      using assms RP-io-targets-split[of vs xs M1 M2 FAIL PM V V'' io s1] by simp
    moreover have io-targets M2 (initial M2) io = {s1}
      using ⟨io ∈ RP M2 s1 vs xs V''⟩ assms(3) RP-state-component-2[of io M2 s1 vs xs V'']
      by blast
    ultimately show io-targets PM (initial PM) io = {s1} × io-targets M1 (initial M1) io
      by auto
  qed
qed

```

**then have**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V''))$   
 $= \bigcup (image (\lambda io . \{s1\} \times io-targets M1 (initial M1) io) (RP M2 s1 vs xs V''))$   
**by simp**  
**moreover have**  $\bigcup (image (\lambda io . \{s1\} \times io-targets M1 (initial M1) io) (RP M2 s1 vs xs V''))$   
 $= \{s1\} \times \bigcup (image (\lambda io . io-targets M1 (initial M1) io) (RP M2 s1 vs xs V''))$   
**by blast**  
**ultimately have image-split-1 :**  
 $\bigcup (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V''))$   
 $= \{s1\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V''))$   
**by simp**  
**then show card**  $(\bigcup (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V'')))$   
 $= card (\bigcup (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V'')))$   
**by (metis (no-types) card-cartesian-product-singleton)**

**have**  $\forall io \in RP M2 s2 vs xs V'' . io-targets PM (initial PM) io$   
 $= \{s2\} \times io-targets M1 (initial M1) io$

**proof**

**fix io assume**  $io \in RP M2 s2 vs xs V''$   
**then have**  $io-targets PM (initial PM) io$   
 $= io-targets M2 (initial M2) io \times io-targets M1 (initial M1) io$   
**using** *assms RP-io-targets-split[of vs xs M1 M2 FAIL PM V V'' io s2]* **by simp**  
**moreover have**  $io-targets M2 (initial M2) io = \{s2\}$   
**using**  $\langle io \in RP M2 s2 vs xs V'' \rangle$  *assms(3) RP-state-component-2[of io M2 s2 vs xs V'']*  
**by blast**  
**ultimately show**  $io-targets PM (initial PM) io = \{s2\} \times io-targets M1 (initial M1) io$   
**by auto**

**qed**

**then have**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V''))$   
 $= \bigcup (image (\lambda io . \{s2\} \times io-targets M1 (initial M1) io) (RP M2 s2 vs xs V''))$   
**by simp**  
**moreover have**  $\bigcup (image (\lambda io . \{s2\} \times io-targets M1 (initial M1) io) (RP M2 s2 vs xs V''))$   
 $= \{s2\} \times \bigcup (image (\lambda io . io-targets M1 (initial M1) io) (RP M2 s2 vs xs V''))$   
**by blast**  
**ultimately have image-split-2 :**  
 $\bigcup (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V''))$   
 $= \{s2\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V''))$  **by simp**  
**then show card**  $(\bigcup (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'')))$   
 $= card (\bigcup (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V'')))$   
**by (metis (no-types) card-cartesian-product-singleton)**

**have**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V''))$   
 $\cap \bigcup (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V''))$   
 $= \{s1\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V''))$   
 $\cap \{s2\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V''))$   
**using image-split-1 image-split-2 by blast**  
**moreover have**  $\{s1\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V''))$   
 $\cap \{s2\} \times \bigcup (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V'')) = \{\}$   
**using assms(9) by auto**  
**ultimately show**  $\bigcup (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V''))$   
 $\cap \bigcup (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'')) = \{\}$   
**by presburger**

**qed**

**lemma LB-count-helper-RP-disjoint-card-M1 :**

**assumes**  $(vs @ xs) \in L M1 \cap L M2$   
**and observable**  $M1$   
**and observable**  $M2$   
**and well-formed**  $M1$   
**and well-formed**  $M2$   
**and productF**  $M2 M1 FAIL PM$   
**and is-det-state-cover**  $M2 V$

**and**  $V'' \in \text{Perm } V \ M1$   
**and**  $s1 \neq s2$   
**shows**  $\text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } PM \left( \text{initial } PM \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right) \right.$   
 $\quad \cup \bigcup \left( \text{image} \left( \text{io-targets } PM \left( \text{initial } PM \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right)$   
 $= \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right) \right.$   
 $\quad \left. + \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) \right)$   
**proof** –  
**have**  $\text{finite} \left( \bigcup \left( \text{image} \left( \text{io-targets } PM \left( \text{initial } PM \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right) \right)$   
**using**  $RP\text{-io-targets-finite-PM}[OF \ \text{assms}(1-8)]$  **by**  $\text{simp}$   
**moreover have**  $\text{finite} \left( \bigcup \left( \text{image} \left( \text{io-targets } PM \left( \text{initial } PM \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) \right)$   
**using**  $RP\text{-io-targets-finite-PM}[OF \ \text{assms}(1-8)]$  **by**  $\text{simp}$   
**ultimately show**  $?thesis$   
**using**  $LB\text{-count-helper-}RP\text{-disjoint-and-cards}[OF \ \text{assms}]$   
**by**  $(metis \ (no-types) \ \text{card-Un-disjoint})$   
**qed**

**lemma**  $LB\text{-count-helper-}RP\text{-disjoint-}M1\text{-pair} :$

**assumes**  $(vs \ @ \ xs) \in L \ M1 \ \cap \ L \ M2$   
**and**  $\text{observable } M1$   
**and**  $\text{observable } M2$   
**and**  $\text{well-formed } M1$   
**and**  $\text{well-formed } M2$   
**and**  $\text{productF } M2 \ M1 \ \text{FAIL } PM$   
**and**  $\text{io-targets } PM \left( \text{initial } PM \right) \ \text{vs} = \{(q2, q1)\}$   
**and**  $\text{path } PM \ (xs \ || \ tr) \ (q2, q1)$   
**and**  $\text{length } xs = \text{length } tr$   
**and**  $\neg \text{Rep-Pre } M2 \ M1 \ \text{vs } xs$   
**and**  $\text{is-det-state-cover } M2 \ V$   
**and**  $V'' \in \text{Perm } V \ M1$   
**and**  $\neg \text{Rep-Cov } M2 \ M1 \ V'' \ \text{vs } xs$   
**and**  $\text{Prereq } M2 \ M1 \ \text{vs } xs \ T \ S \ \Omega \ V''$   
**and**  $s1 \neq s2$   
**and**  $s1 \in S$   
**and**  $s2 \in S$   
**and**  $\text{applicable-set } M1 \ \Omega$   
**and**  $\text{completely-specified } M1$   
**shows**  $\text{card} \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) + \text{card} \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right)$   
 $= \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right) \right)$   
 $+ \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) \right)$   
 $\bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right)$   
 $\cap \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right)$   
 $= \{\}$   
**proof** –  
**have**  $s1 \in \text{nodes } M2$   
**using**  $\text{assms}(14, 16)$  **unfolding**  $\text{Prereq.simps}$  **by**  $\text{blast}$   
**have**  $s2 \in \text{nodes } M2$   
**using**  $\text{assms}(14, 17)$  **unfolding**  $\text{Prereq.simps}$  **by**  $\text{blast}$   
**have**  $\text{card} \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right)$   
 $= \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right) \right)$   
**using**  $RP\text{-count-alt-def}[OF \ \text{assms}(1-5) \ \langle s1 \in \text{nodes } M2 \rangle \ \text{assms}(6-13)]$   
**by**  $\text{linarith}$   
**moreover have**  $\text{card} \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right)$   
 $= \text{card} \left( \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) \right)$   
**using**  $RP\text{-count-alt-def}[OF \ \text{assms}(1-5) \ \langle s2 \in \text{nodes } M2 \rangle \ \text{assms}(6-13)]$   
**by**  $\text{linarith}$   
**moreover show**  $\bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right)$   
 $\cap \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) = \{\}$   
**proof**  $(rule \ ccontr)$   
**assume**  $\bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s1 \ \text{vs } xs \ V'' \right) \right)$   
 $\cap \bigcup \left( \text{image} \left( \text{io-targets } M1 \left( \text{initial } M1 \right) \right) \left( RP \ M2 \ s2 \ \text{vs } xs \ V'' \right) \right) \neq \{\}$   
**then obtain**  $io1 \ io2 \ t$  **where**  $\text{shared-elem-def} :$   
 $io1 \in (RP \ M2 \ s1 \ \text{vs } xs \ V'')$   
 $io2 \in (RP \ M2 \ s2 \ \text{vs } xs \ V'')$   
 $t \in \text{io-targets } M1 \left( \text{initial } M1 \right) \ io1$   
 $t \in \text{io-targets } M1 \left( \text{initial } M1 \right) \ io2$

by *blast*

have *dist-prop*:  $(\forall s1 \in S . \forall s2 \in S . s1 \neq s2$   
 $\longrightarrow (\forall io1 \in RP\ M2\ s1\ vs\ xs\ V'' .$   
 $\quad \forall io2 \in RP\ M2\ s2\ vs\ xs\ V'' .$   
 $\quad B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega ))$

using *assms(14)* by *simp*

have *io-targets M1 (initial M1) io1*  $\cap$  *io-targets M1 (initial M1) io2* = {}

proof (rule *ccontr*)

assume *io-targets M1 (initial M1) io1*  $\cap$  *io-targets M1 (initial M1) io2*  $\neq$  {}

then have *io-targets M1 (initial M1) io1*  $\neq$  {} *io-targets M1 (initial M1) io2*  $\neq$  {}

by *blast+*

then obtain *s1 s2* where *s1*  $\in$  *io-targets M1 (initial M1) io1*  
*s2*  $\in$  *io-targets M1 (initial M1) io2*

by *blast*

then have *io-targets M1 (initial M1) io1* = {*s1*}  
*io-targets M1 (initial M1) io2* = {*s2*}

by (*meson assms(2) observable-io-target-is-singleton*)+

then have *s1* = *s2*

using  $\langle io-targets\ M1\ (initial\ M1)\ io1\ \cap\ io-targets\ M1\ (initial\ M1)\ io2\ \neq\ \{\} \rangle$

by *auto*

then have *B M1 io1*  $\Omega$  = *B M1 io2*  $\Omega$

using  $\langle io-targets\ M1\ (initial\ M1)\ io1 = \{s1\} \rangle$   $\langle io-targets\ M1\ (initial\ M1)\ io2 = \{s2\} \rangle$

by *auto*

then show *False*

using *assms(15–17) dist-prop shared-elem-def(1,2)* by *blast*

qed

then show *False*

using *shared-elem-def(3,4)* by *blast*

qed

ultimately show *card (RP M2 s1 vs xs V'') + card (RP M2 s2 vs xs V'')*  
 $= card (\bigcup (image\ (io-targets\ M1\ (initial\ M1))\ (RP\ M2\ s1\ vs\ xs\ V'')))$   
 $+ card (\bigcup (image\ (io-targets\ M1\ (initial\ M1))\ (RP\ M2\ s2\ vs\ xs\ V'')))$

by *linarith*

qed

lemma *LB-count-helper-RP-card-union* :

assumes *observable M2*

and *s1*  $\neq$  *s2*

shows *RP M2 s1 vs xs V''*  $\cap$  *RP M2 s2 vs xs V''* = {}

proof (rule *ccontr*)

assume *RP M2 s1 vs xs V''*  $\cap$  *RP M2 s2 vs xs V''*  $\neq$  {}

then obtain *io* where *io*  $\in$  *RP M2 s1 vs xs V''*  $\wedge$  *io*  $\in$  *RP M2 s2 vs xs V''*

by *blast*

then have *s1*  $\in$  *io-targets M2 (initial M2) io*  
*s2*  $\in$  *io-targets M2 (initial M2) io*

by *auto*

then have *s1* = *s2*

using *assms(1)* by (*metis observable-io-target-is-singleton singletonD*)

then show *False*

using *assms(2)* by *simp*

qed

**lemma** *LB-count-helper-RP-inj* :

**obtains**  $f$

**where**  $\forall q \in (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S) .$   
 $f\ q \in \text{nodes } M1$   
 $\text{inj-on } f (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S)$

**proof** –

**let**  $?f =$   
 $\lambda q . \text{if } (q \in (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S))$   
 $\text{then } q$   
 $\text{else } (\text{initial } M1)$

**have**  $(\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S) \subseteq \text{nodes } M1$   
**by** *blast*

**then have**  $\forall q \in (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S) .$   
 $?f\ q \in \text{nodes } M1$   
**by** (*metis Un-iff sup.order-iff*)

**moreover have**  $\text{inj-on } ?f (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S)$

**proof**

**fix**  $x$  **assume**  $x \in (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S)$   
**then have**  $?f\ x = x$   
**by** *presburger*

**fix**  $y$  **assume**  $y \in (\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S)$   
**then have**  $?f\ y = y$   
**by** *presburger*

**assume**  $?f\ x = ?f\ y$   
**then show**  $x = y$  **using**  $\langle ?f\ x = x \rangle \langle ?f\ y = y \rangle$   
**by** *presburger*

**qed**

**ultimately show** *?thesis*  
**using that** **by** *presburger*

**qed**

**abbreviation** (*input*) *UNION* ::  $'a\ \text{set} \Rightarrow ('a \Rightarrow 'b\ \text{set}) \Rightarrow 'b\ \text{set}$   
**where**  $UNION\ A\ f \equiv \bigcup (f\ 'A)$

**lemma** *LB-count-helper-RP-card-union-sum* :

**assumes**  $(vs\ @\ xs) \in L\ M2 \cap L\ M1$   
**and** *OFSM*  $M1$   
**and** *OFSM*  $M2$   
**and** *asc-fault-domain*  $M2\ M1\ m$   
**and** *test-tools*  $M2\ M1\ FAIL\ PM\ V\ \Omega$   
**and**  $V'' \in \text{Perm } V\ M1$   
**and** *Prereq*  $M2\ M1\ vs\ xs\ T\ S\ \Omega\ V''$   
**and**  $\neg \text{Rep-Pre } M2\ M1\ vs\ xs$   
**and**  $\neg \text{Rep-Cov } M2\ M1\ V''\ vs\ xs$

**shows**  $\text{sum } (\lambda s . \text{card } (RP\ M2\ s\ vs\ xs\ V'')) S$   
 $= \text{sum } (\lambda s . \text{card } (\bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S$

**using** *assms* **proof** –

**have** *finite* (*nodes*  $M2$ )  
**using** *assms*(3) **by** *auto*

**moreover have**  $S \subseteq \text{nodes } M2$   
**using** *assms*(7) **by** *simp*

**ultimately have** *finite*  $S$   
**using** *infinite-super* **by** *blast*

**then have**  $\text{sum } (\lambda s . \text{card } (RP\ M2\ s\ vs\ xs\ V'')) S$   
 $= \text{sum } (\lambda s . \text{card } (\bigcup (\text{image } (\text{io-targets } M1 (\text{initial } M1)) (RP\ M2\ s\ vs\ xs\ V'')) S)) S$

**using** *assms* **proof** (*induction*  $S$ )

```

case empty
show ?case by simp
next
case (insert s S)

have (insert s S)  $\subseteq$  nodes M2
  using insert.premis(7) by simp
then have s  $\in$  nodes M2
  by simp

have Prereq M2 M1 vs xs T S  $\Omega$  V''
  using  $\langle$ Prereq M2 M1 vs xs T (insert s S)  $\Omega$  V'' $\rangle$  by simp
then have  $(\sum s \in S. \text{card } (RP\ M2\ s\ vs\ xs\ V''))$ 
  =  $(\sum s \in S. \text{card } (\bigcup a \in RP\ M2\ s\ vs\ xs\ V''. \text{io-targets } M1\ (\text{initial } M1)\ a))$ 
  using insert.IH[OF insert.premis(1-6) - assms(8,9)] by metis
moreover have  $(\sum s' \in (\text{insert } s\ S). \text{card } (RP\ M2\ s'\ vs\ xs\ V''))$ 
  =  $(\sum s' \in S. \text{card } (RP\ M2\ s'\ vs\ xs\ V'')) + \text{card } (RP\ M2\ s\ vs\ xs\ V'')$ 
  by (simp add: add.commute insert.hyps(1) insert.hyps(2))
ultimately have S-prop :  $(\sum s' \in (\text{insert } s\ S). \text{card } (RP\ M2\ s'\ vs\ xs\ V''))$ 
  =  $(\sum s \in S. \text{card } (\bigcup a \in RP\ M2\ s\ vs\ xs\ V''. \text{io-targets } M1\ (\text{initial } M1)\ a))$ 
  +  $\text{card } (RP\ M2\ s\ vs\ xs\ V'')$ 
  by presburger

have vs@xs  $\in$  L M1  $\cap$  L M2
  using insert.premis(1) by simp

obtain q2 q1 tr where suffix-path : io-targets PM (initial PM) vs =  $\{(q2, q1)\}$ 
  path PM (xs || tr) (q2, q1)
  length xs = length tr
  using productF-language-state-intermediate[OF insert.premis(1)
  test-tools-props(1)[OF insert.premis(5,4)] OFSM-props(2,1)[OF insert.premis(3)]
  OFSM-props(2,1)[OF insert.premis(2)]
  by blast

have  $\text{card } (RP\ M2\ s\ vs\ xs\ V'')$ 
  =  $\text{card } (\bigcup (\text{image } (\text{io-targets } M1\ (\text{initial } M1))\ (RP\ M2\ s\ vs\ xs\ V'')))$ 
  using OFSM-props(2,1)[OF insert.premis(3)] OFSM-props(2,1)[OF insert.premis(2)]
  RP-count-alt-def[OF  $\langle vs@xs \in L\ M1 \cap L\ M2 \rangle$  - - -
   $\langle s \in \text{nodes } M2 \rangle$  test-tools-props(1)[OF insert.premis(5,4)]
  suffix-path insert.premis(8)
  test-tools-props(2)[OF insert.premis(5,4)] assms(6) insert.premis(9)]
  by linarith

show  $(\sum s \in \text{insert } s\ S. \text{card } (RP\ M2\ s\ vs\ xs\ V'')) =$ 
   $(\sum s \in \text{insert } s\ S. \text{card } (\text{UNION } (RP\ M2\ s\ vs\ xs\ V'')\ (\text{io-targets } M1\ (\text{initial } M1))))$ 
proof -
  have  $(\sum c \in \text{insert } s\ S. \text{card } (\text{UNION } (RP\ M2\ c\ vs\ xs\ V'')\ (\text{io-targets } M1\ (\text{initial } M1))))$ 
  =  $\text{card } (\text{UNION } (RP\ M2\ s\ vs\ xs\ V'')\ (\text{io-targets } M1\ (\text{initial } M1)))$ 
  +  $(\sum c \in S. \text{card } (\text{UNION } (RP\ M2\ c\ vs\ xs\ V'')\ (\text{io-targets } M1\ (\text{initial } M1))))$ 
  by (meson insert.hyps(1) insert.hyps(2) sum.insert)
then show ?thesis
  using  $\langle$  $(\sum s' \in \text{insert } s\ S. \text{card } (RP\ M2\ s'\ vs\ xs\ V''))$ 
  =  $(\sum s \in S. \text{card } (\bigcup a \in RP\ M2\ s\ vs\ xs\ V''. \text{io-targets } M1\ (\text{initial } M1)\ a))$ 
  +  $\text{card } (RP\ M2\ s\ vs\ xs\ V'')$ 
   $\langle$  $\text{card } (RP\ M2\ s\ vs\ xs\ V'')$ 
  =  $\text{card } (\text{UNION } (RP\ M2\ s\ vs\ xs\ V'')\ (\text{io-targets } M1\ (\text{initial } M1)))\rangle$ 
  by presburger
qed
qed

then show ?thesis
  using assms by blast
qed

```

```

lemma finite-insert-card :
  assumes finite ( $\bigcup SS$ )
  and finite S
  and  $S \cap (\bigcup SS) = \{\}$ 
shows card ( $\bigcup (\text{insert } S \text{ } SS)$ ) = card ( $\bigcup SS$ ) + card S
  by (simp add: assms(1) assms(2) assms(3) card-Un-disjoint)

lemma LB-count-helper-RP-disjoint-M1-union :
  assumes (vs @ xs)  $\in L \text{ } M2 \cap L \text{ } M1$ 
  and OFSM M1
  and OFSM M2
  and asc-fault-domain M2 M1 m
  and test-tools M2 M1 FAIL PM V  $\Omega$ 
  and  $V'' \in \text{Perm } V \text{ } M1$ 
  and Prereq M2 M1 vs xs T S  $\Omega \text{ } V''$ 
  and  $\neg \text{Rep-Pre } M2 \text{ } M1 \text{ } vs \text{ } xs$ 
  and  $\neg \text{Rep-Cov } M2 \text{ } M1 \text{ } V'' \text{ } vs \text{ } xs$ 
shows sum ( $\lambda s . \text{card } (RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V'')$ ) S
  = card ( $\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 \text{ } (\text{initial } M1)) (RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V'')) S))$ )
using assms proof -
  have finite (nodes M2)
    using assms(3) by auto
  moreover have  $S \subseteq \text{nodes } M2$ 
    using assms(7) by simp
  ultimately have finite S
    using infinite-super by blast

then show sum ( $\lambda s . \text{card } (RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V'')$ ) S
  = card ( $\bigcup (\text{image } (\lambda s . \bigcup (\text{image } (\text{io-targets } M1 \text{ } (\text{initial } M1)) (RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V'')) S))$ )
using assms proof (induction S)
  case empty
  show ?case by simp
next
  case (insert s S)

  have (insert s S)  $\subseteq \text{nodes } M2$ 
    using insert.prem(7) by simp
  then have  $s \in \text{nodes } M2$ 
    by simp

  have Prereq M2 M1 vs xs T S  $\Omega \text{ } V''$ 
    using  $\langle \text{Prereq } M2 \text{ } M1 \text{ } vs \text{ } xs \text{ } T \text{ } (\text{insert } s \text{ } S) \text{ } \Omega \text{ } V'' \rangle$  by simp
  then have applied-IH : ( $\sum s \in S. \text{card } (RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V'')$ )
    = card ( $\bigcup s \in S. \bigcup a \in RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V''. \text{io-targets } M1 \text{ } (\text{initial } M1) \text{ } a$ )
    using insert.IH[OF insert.prem(1-6) - insert.prem(8,9)] by metis

  obtain q2 q1 tr where suffix-path : io-targets PM (initial PM) vs = {(q2,q1)}
    path PM (xs || tr) (q2,q1)
    length xs = length tr
  using productF-language-state-intermediate
    [OF insert.prem(1) test-tools-props(1)[OF insert.prem(5,4)]
    OFSM-props(2,1)[OF insert.prem(3)] OFSM-props(2,1)[OF insert.prem(2)]]
  by blast

  have  $s \in \text{insert } s \text{ } S$ 
    by simp

  have vs@xs  $\in L \text{ } M1 \cap L \text{ } M2$ 
    using insert.prem(1) by simp

  have  $\forall s' \in S . (\bigcup a \in RP \text{ } M2 \text{ } s \text{ } vs \text{ } xs \text{ } V''. \text{io-targets } M1 \text{ } (\text{initial } M1) \text{ } a)$ 
     $\cap (\bigcup a \in RP \text{ } M2 \text{ } s' \text{ } vs \text{ } xs \text{ } V''. \text{io-targets } M1 \text{ } (\text{initial } M1) \text{ } a) = \{\}$ 
  proof
    fix s' assume s'  $\in S$ 

```

```

have s ≠ s'
  using insert.hyps(2) ⟨s' ∈ S⟩ by blast
have s' ∈ insert s S
  using ⟨s' ∈ S⟩ by simp

show (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  ∩ (⋃ a∈RP M2 s' vs xs V''. io-targets M1 (initial M1) a) = {}
  using OFSM-props(2,1)[OF assms(3)] OFSM-props(2,1,3)[OF assms(2)]
  LB-count-helper-RP-disjoint-M1-pair(2)
  [OF ⟨vs@xs ∈ L M1 ∩ L M2⟩ - - - test-tools-props(1)[OF insert.prem(5,4)]
  suffix-path insert.prem(8) test-tools-props(2)[OF insert.prem(5,4)]
  insert.prem(6,9,7) ⟨s ≠ s'⟩ ⟨s ∈ insert s S⟩ ⟨s' ∈ insert s S⟩
  test-tools-props(4)[OF insert.prem(5,4)]]
  by linarith
qed
then have disj-insert : (⋃ s∈S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  ∩ (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a) = {}
  by blast
have finite-S : finite (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  using RP-io-targets-finite-M1[OF insert.prem(1)]
  by (meson RP-io-targets-finite-M1 ⟨vs @ xs ∈ L M1 ∩ L M2⟩ assms(2) assms(5) insert.prem(6))
have finite-s : finite (⋃ s∈S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  by (meson RP-io-targets-finite-M1 ⟨vs @ xs ∈ L M1 ∩ L M2⟩ assms(2) assms(5)
  finite-UN-I insert.hyps(1) insert.prem(6))

have card (⋃ s∈insert s S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  = card (⋃ s∈S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  + card (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
proof -
  have f1: insert (UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1)))
    ((λc. UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1))) ' S)
    = (λc. UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1))) ' insert s S
  by blast
  have ∀ c. c ∈ S → UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1))
    ∩ UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1)) = {}
  by (meson ⟨∀ s'∈S. (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
    ∩ (⋃ a∈RP M2 s' vs xs V''. io-targets M1 (initial M1) a) = {}⟩)
  then have UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1))
    ∩ (⋃ c∈S. UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1))) = {}
  by blast
  then show ?thesis
    using f1 by (metis finite-S finite-insert-card finite-s)
qed

have card (RP M2 s vs xs V'')
  = card (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
  using assms(2) assms(3)
  RP-count-alt-def[OF ⟨vs@xs ∈ L M1 ∩ L M2⟩ - - - ⟨s ∈ nodes M2⟩
  test-tools-props(1)[OF insert.prem(5,4)] suffix-path
  insert.prem(8) test-tools-props(2)[OF insert.prem(5,4)]
  insert.prem(6,9)]
  by metis

show ?case
proof -
  have (∑ c∈insert s S. card (RP M2 c vs xs V''))
    = card (RP M2 s vs xs V'') + (∑ c∈S. card (RP M2 c vs xs V''))
  by (meson insert.hyps(1) insert.hyps(2) sum.insert)
  then show ?thesis
    using ⟨card (RP M2 s vs xs V'')
    = card (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)⟩
    ⟨card (⋃ s∈insert s S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
    = card (⋃ s∈S. ⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)
    + card (⋃ a∈RP M2 s vs xs V''. io-targets M1 (initial M1) a)⟩ applied-IH
  by presburger

```

qed  
 qed  
 qed

**lemma** *LB-count-helper-LB1* :

**assumes**  $(vs @ xs) \in L M2 \cap L M1$   
**and** *OFSM*  $M1$   
**and** *OFSM*  $M2$   
**and** *asc-fault-domain*  $M2 M1 m$   
**and** *test-tools*  $M2 M1 FAIL PM V \Omega$   
**and**  $V'' \in Perm V M1$   
**and** *Prereq*  $M2 M1 vs xs T S \Omega V''$   
**and**  $\neg Rep-Pre M2 M1 vs xs$   
**and**  $\neg Rep-Cov M2 M1 V'' vs xs$

**shows**  $(sum (\lambda s . card (RP M2 s vs xs V'')) S) \leq card (nodes M1)$

**proof** –

**have**  $(\bigcup_{s \in S}. UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1))) \subseteq nodes M1$   
**by** *blast*

**moreover have** *finite*  $(nodes M1)$

**using** *assms(2) OFSM-props(1) unfolding well-formed.simps finite-FSM.simps by simp*

**ultimately have**  $card (\bigcup_{s \in S}. UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1))) \leq card (nodes M1)$

**by** *(meson card-mono)*

**moreover have**  $(\sum_{s \in S}. card (RP M2 s vs xs V''))$

$= card (\bigcup_{s \in S}. UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1)))$

**using** *LB-count-helper-RP-disjoint-M1-union[OF assms]*

**by** *linarith*

**ultimately show** *?thesis*

**by** *linarith*

qed

**lemma** *LB-count-helper-D-states* :

**assumes** *observable*  $M$   
**and**  $RS \in (D M T \Omega)$

**obtains**  $q$

**where**  $q \in nodes M \wedge RS = IO-set M q \Omega$

**proof** –

**have**  $RS \in image (\lambda io . B M io \Omega) (LS_{in} M (initial M) T)$

**using** *assms by simp*

**then obtain**  $io$  **where**  $RS = B M io \Omega$   $io \in LS_{in} M (initial M) T$

**by** *blast*

**then have**  $io \in language-state M (initial M)$

**using** *language-state-for-inputs-in-language-state[of M initial M T] by blast*

**then obtain**  $q$  **where**  $\{q\} = io-targets M (initial M) io$

**by** *(metis assms(1) io-targets-observable-singleton-ob)*

**then have**  $B M io \Omega = \bigcup (image (\lambda s . IO-set M s \Omega) \{q\})$

**by** *simp*

**then have**  $B M io \Omega = IO-set M q \Omega$

**by** *simp*

**then have**  $RS = IO-set M q \Omega$  **using**  $\langle RS = B M io \Omega \rangle$

**by** *simp*

**moreover have**  $q \in nodes M$  **using**  $\langle \{q\} = io-targets M (initial M) io \rangle$

**by** *(metis FSM.nodes.insertI1 io-targets-nodes)*

**ultimately show** *?thesis*

**using** *that by simp*

qed

**lemma** *LB-count-helper-LB2* :

**assumes** *observable M1*  
**and**  $IO\text{-set } M1 \ q \ \Omega \in (D \ M1 \ T \ \Omega) - \{B \ M1 \ xs' \ \Omega \mid xs' \ s' . s' \in S \wedge xs' \in RP \ M2 \ s' \ vs \ xs \ V''\}$   
**shows**  $q \notin \left( \bigcup (image \ (\lambda \ s . \bigcup (image \ (io\text{-targets } M1 \ (initial \ M1)) \ (RP \ M2 \ s \ vs \ xs \ V'')) \ S) \right)$   
**proof**  
**assume**  $q \in \left( \bigcup_{s \in S} UNION \ (RP \ M2 \ s \ vs \ xs \ V'') \ (io\text{-targets } M1 \ (initial \ M1)) \right)$   
**then obtain**  $s' \text{ where } s' \in S \ q \in \left( \bigcup (image \ (io\text{-targets } M1 \ (initial \ M1)) \ (RP \ M2 \ s' \ vs \ xs \ V'')) \right)$   
**by** *blast*  
**then obtain**  $xs' \text{ where } q \in io\text{-targets } M1 \ (initial \ M1) \ xs' \ xs' \in RP \ M2 \ s' \ vs \ xs \ V''$   
**by** *blast*  
**then have**  $\{q\} = io\text{-targets } M1 \ (initial \ M1) \ xs'$   
**by** *(metis assms(1) observable-io-target-is-singleton)*  
**then have**  $B \ M1 \ xs' \ \Omega = \bigcup (image \ (\lambda \ s . IO\text{-set } M1 \ s \ \Omega) \ \{q\})$   
**by** *simp*  
**then have**  $B \ M1 \ xs' \ \Omega = IO\text{-set } M1 \ q \ \Omega$   
**by** *simp*  
**moreover have**  $B \ M1 \ xs' \ \Omega \in \{B \ M1 \ xs' \ \Omega \mid xs' \ s' . s' \in S \wedge xs' \in RP \ M2 \ s' \ vs \ xs \ V''\}$   
**using**  $\langle s' \in S \rangle \langle xs' \in RP \ M2 \ s' \ vs \ xs \ V'' \rangle$  **by** *blast*  
**ultimately have**  $IO\text{-set } M1 \ q \ \Omega \in \{B \ M1 \ xs' \ \Omega \mid xs' \ s' . s' \in S \wedge xs' \in RP \ M2 \ s' \ vs \ xs \ V''\}$   
**by** *blast*  
**moreover have**  $IO\text{-set } M1 \ q \ \Omega \notin \{B \ M1 \ xs' \ \Omega \mid xs' \ s' . s' \in S \wedge xs' \in RP \ M2 \ s' \ vs \ xs \ V''\}$   
**using** *assms(2)* **by** *blast*  
**ultimately show** *False*  
**by** *simp*  
**qed**

## 4.7 Validity of the result of LB constituting a lower bound

**lemma** *LB-count* :

**assumes**  $(vs \ @ \ xs) \in L \ M1$   
**and** *OFSM M1*  
**and** *OFSM M2*  
**and** *asc-fault-domain M2 M1 m*  
**and** *test-tools M2 M1 FAIL PM V Ω*  
**and**  $V'' \in Perm \ V \ M1$   
**and** *Prereq M2 M1 vs xs T S Ω V''*  
**and**  $\neg Rep\text{-Pre } M2 \ M1 \ vs \ xs$   
**and**  $\neg Rep\text{-Cov } M2 \ M1 \ V'' \ vs \ xs$   
**shows**  $LB \ M2 \ M1 \ vs \ xs \ T \ S \ \Omega \ V'' \leq |M1|$   
**proof** –

**let**  $?D = D \ M1 \ T \ \Omega$   
**let**  $?B = \{B \ M1 \ xs' \ \Omega \mid xs' \ s' . s' \in S \wedge xs' \in RP \ M2 \ s' \ vs \ xs \ V''\}$   
**let**  $?DB = ?D - ?B$   
**let**  $?RP = \bigcup_{s \in S} \bigcup_{a \in RP \ M2 \ s \ vs \ xs \ V''} io\text{-targets } M1 \ (initial \ M1) \ a$

**have** *finite (nodes M1)*  
**using** *OFSM-props[OF assms(2)] unfolding well-formed.simps finite-FSM.simps by simp*  
**then have** *finite ?D*  
**using** *OFSM-props[OF assms(2)] assms(7) D-bound[of M1 T Ω] unfolding Prereq.simps by linarith*  
**then have** *finite ?DB*  
**by** *simp*

— Proof sketch: Construct a function  $f$  (via induction) that maps each response set in  $?DB$  to some state that produces that response set. This is then used to show that each response sets in  $?DB$  indicates the existence of a distinct state in  $M1$  not reached via the  $RP$ -sequences.

**have**  $states\text{-}f : \bigwedge DB' . DB' \subseteq ?DB \implies \exists f . inj\text{-on } f \ DB'$   
 $\wedge image \ f \ DB' \subseteq (nodes \ M1) - ?RP$   
 $\wedge (\forall RS \in DB' . IO\text{-set } M1 \ (f \ RS) \ \Omega = RS)$

**proof** –  
**fix**  $DB' \text{ assume } DB' \subseteq ?DB$   
**have** *finite DB'*  
**proof** *(rule ccontr)*  
**assume** *infinite DB'*

```

have infinite ?DB
  using infinite-super[OF ‹DB' ⊆ ?DB› ‹infinite DB'›] by simp
then show False
  using ‹finite ?DB› by simp
qed
then show ∃ f . inj-on f DB' ∧ image f DB' ⊆ (nodes M1) - ?RP
  ∧ (∀ RS ∈ DB' . IO-set M1 (f RS) Ω = RS)
using assms ‹DB' ⊆ ?DB› proof (induction DB')
  case empty
  show ?case by simp
next
  case (insert RS DB')

  have DB' ⊆ ?DB
    using insert.prem(10) by blast
  obtain f' where inj-on f' DB'
    image f' DB' ⊆ (nodes M1) - ?RP
    ∀ RS ∈ DB' . IO-set M1 (f' RS) Ω = RS
  using insert.IH[OF insert.prem(1-9) ‹DB' ⊆ ?DB›]
  by blast

  have RS ∈ D M1 T Ω
    using insert.prem(10) by blast
  obtain q where q ∈ nodes M1 RS = IO-set M1 q Ω
    using insert.prem(2) LB-count-helper-D-states[OF - ‹RS ∈ D M1 T Ω›]
    by blast
  then have IO-set M1 q Ω ∈ ?DB
    using insert.prem(10) by blast

  have q ∉ ?RP
    using insert.prem(2) LB-count-helper-LB2[OF - ‹IO-set M1 q Ω ∈ ?DB›]
    by blast

  let ?f = f'(RS := q)
  have inj-on ?f (insert RS DB')
  proof
    have ?f RS ∉ ?f '(DB' - {RS})
    proof
      assume ?f RS ∈ ?f '(DB' - {RS})
      then have q ∈ ?f '(DB' - {RS}) by auto
      have RS ∈ DB'
      proof -
        have ∀ P c f. ∃ Pa. ((c::'c) ∉ f' P ∨ (Pa::('a × 'b) list set) ∈ P)
          ∧ (c ∉ f' P ∨ f Pa = c)
          by auto
        moreover
        { assume q ∉ f' ' DB'
          moreover
          { assume q ∉ f'(RS := q) ' DB'
            then have ?thesis
              using ‹q ∈ f'(RS := q) ' (DB' - {RS})› by blast }
          ultimately have ?thesis
            by (metis fun-upd-image) }
          ultimately show ?thesis
            by (metis (no-types) ‹RS = IO-set M1 q Ω› ‹∀ RS ∈ DB'. IO-set M1 (f' RS) Ω = RS›)
        }
      qed
    }
    then show False using insert.hyps(2) by simp
  qed
  then show inj-on ?f DB' ∧ ?f RS ∉ ?f '(DB' - {RS})
    using ‹inj-on f' DB'› inj-on-fun-updI by fastforce
  qed
  moreover have image ?f (insert RS DB') ⊆ (nodes M1) - ?RP
  proof -
    have image ?f {RS} = {q} by simp
    then have image ?f {RS} ⊆ (nodes M1) - ?RP
      using ‹q ∈ nodes M1› ‹q ∉ ?RP› by auto
  
```

```

moreover have  $image\ ?f\ (insert\ RS\ DB') = image\ ?f\ \{RS\} \cup image\ ?f\ DB'$ 
  by auto
ultimately show ?thesis
  by  $(metis\ (no-types,\ lifting)\ \langle image\ f'\ DB' \subseteq (nodes\ M1) - ?RP \rangle\ fun-upd-other\ image-cong\ image-insert\ insert.hyps(2)\ insert-subset)$ 
qed
moreover have  $\forall\ RS \in (insert\ RS\ DB') . IO-set\ M1\ (?f\ RS)\ \Omega = RS$ 
  using  $\langle RS = IO-set\ M1\ q\ \Omega \rangle\ \langle \forall\ RS \in DB' . IO-set\ M1\ (f'\ RS)\ \Omega = RS \rangle$  by auto

  ultimately show ?case
    by blast
  qed
qed

have  $?DB \subseteq ?DB$ 
  by simp
obtain  $f$  where  $inj-on\ f\ ?DB\ image\ f\ ?DB \subseteq (nodes\ M1) - ?RP$ 
  using  $states-f[OF\ \langle ?DB \subseteq ?DB \rangle]$  by blast
have  $finite\ (nodes\ M1 - ?RP)$ 
  using  $\langle finite\ (nodes\ M1) \rangle$  by simp
have  $card\ ?DB \leq card\ (nodes\ M1 - ?RP)$ 
  using  $card-inj-on-le[OF\ \langle inj-on\ f\ ?DB \rangle\ \langle image\ f\ ?DB \subseteq (nodes\ M1) - ?RP \rangle\ \langle finite\ (nodes\ M1 - ?RP) \rangle]$ 
  by assumption

have  $?RP \subseteq nodes\ M1$ 
  by blast
then have  $card\ (nodes\ M1 - ?RP) = card\ (nodes\ M1) - card\ ?RP$ 
  by  $(meson\ \langle finite\ (nodes\ M1) \rangle\ card-Diff-subset\ infinite-subset)$ 
then have  $card\ ?DB \leq card\ (nodes\ M1) - card\ ?RP$ 
  using  $\langle card\ ?DB \leq card\ (nodes\ M1 - ?RP) \rangle$  by linarith

have  $vs\ @\ xs \in L\ M2 \cap L\ M1$ 
  using  $assms(7)$  by simp
have  $(sum\ (\lambda\ s . card\ (RP\ M2\ s\ vs\ xs\ V''))\ S) = card\ ?RP$ 
  using  $LB-count-helper-RP-disjoint-M1-union[OF\ \langle vs\ @\ xs \in L\ M2 \cap L\ M1 \rangle\ assms(2-9)]$  by simp
moreover have  $card\ ?RP \leq card\ (nodes\ M1)$ 
  using  $card-mono[OF\ \langle finite\ (nodes\ M1) \rangle\ \langle ?RP \subseteq nodes\ M1 \rangle]$  by assumption
ultimately show ?thesis
  unfolding  $LB.simps$  using  $\langle card\ ?DB \leq card\ (nodes\ M1) - card\ ?RP \rangle$ 
  by linarith
qed

lemma contradiction-via-LB :
assumes  $(vs\ @\ xs) \in L\ M1$ 
  and  $OFSM\ M1$ 
  and  $OFSM\ M2$ 
  and  $asc-fault-domain\ M2\ M1\ m$ 
  and  $test-tools\ M2\ M1\ FAIL\ PM\ V\ \Omega$ 
  and  $V'' \in Perm\ V\ M1$ 
  and  $Prereq\ M2\ M1\ vs\ xs\ T\ S\ \Omega\ V''$ 
  and  $\neg\ Rep-Pre\ M2\ M1\ vs\ xs$ 
  and  $\neg\ Rep-Cov\ M2\ M1\ V''\ vs\ xs$ 
  and  $LB\ M2\ M1\ vs\ xs\ T\ S\ \Omega\ V'' > m$ 
shows False
proof -
  have  $LB\ M2\ M1\ vs\ xs\ T\ S\ \Omega\ V'' \leq card\ (nodes\ M1)$ 
  using  $LB-count[OF\ assms(1-9)]$  by assumption
  moreover have  $card\ (nodes\ M1) \leq m$ 
  using  $assms(4)$  by auto
  ultimately show False
  using  $assms(10)$  by linarith
qed

```

```

end
theory ASC-Suite
imports ASC-LB
begin

```

## 5 Test suite generated by the Adaptive State Counting Algorithm

### 5.1 Maximum length contained prefix

```

fun mcp :: 'a list ⇒ 'a list set ⇒ 'a list ⇒ bool where
  mcp z W p = (prefix p z ∧ p ∈ W ∧
    (∀ p'. (prefix p' z ∧ p' ∈ W) ⟶ length p' ≤ length p))

lemma mcp-ex :
  assumes [] ∈ W
  and finite W
  obtains p
  where mcp z W p
  proof -
    let ?P = {p . prefix p z ∧ p ∈ W}
    let ?maxP = arg-max length (λ p . p ∈ ?P)

    have finite {p . prefix p z}
    proof -
      have {p . prefix p z} ⊆ image (λ i . take i z) (set [0 ..< Suc (length z)])
      proof
        fix p assume p ∈ {p . prefix p z}
        then obtain i where i ≤ length z ∧ p = take i z
        by (metis append-eq-conv-conj mem-Collect-eq prefix-def prefix-length-le)
        then have i < Suc (length z) ∧ p = take i z
        by simp
        then show p ∈ image (λ i . take i z) (set [0 ..< Suc (length z)])
        using atLeast-upt by blast
      qed
      then show ?thesis
      using finite-surj by blast
    qed
    then have finite ?P
    by simp

    have ?P ≠ {}
    using Nil-prefix assms(1) by blast

    have ∃ maxP ∈ ?P . ∀ p ∈ ?P . length p ≤ length maxP
    proof (rule ccontr)
      assume ¬(∃ maxP ∈ ?P . ∀ p ∈ ?P . length p ≤ length maxP)
      then have ∀ p ∈ ?P . ∃ p' ∈ ?P . length p < length p'
      by (meson not-less)
      then have ∀ l ∈ (image length ?P) . ∃ l' ∈ (image length ?P) . l < l'
      by auto

      then have infinite (image length ?P)
      by (metis (no-types, lifting) ‹?P ≠ {}› image-is-empty infinite-growing)
      then have infinite ?P
      by blast
      then show False
      using ‹finite ?P› by simp
    qed

    then obtain maxP where maxP ∈ ?P ∧ ∀ p ∈ ?P . length p ≤ length maxP
    by blast

    then have mcp z W maxP
    unfolding mcp.simps by blast

```

then show *?thesis*  
 using *that* by *auto*  
 qed

lemma *mcp-unique* :  
 assumes *mcp z W p*  
 and *mcp z W p'*  
 shows  $p = p'$   
 proof –  
 have  $\text{length } p' \leq \text{length } p$   
 using *assms(1) assms(2)* by *auto*  
 moreover have  $\text{length } p \leq \text{length } p'$   
 using *assms(1) assms(2)* by *auto*  
 ultimately have  $\text{length } p' = \text{length } p$   
 by *simp*

moreover have *prefix p z*  
 using *assms(1)* by *auto*  
 moreover have *prefix p' z*  
 using *assms(2)* by *auto*  
 ultimately show *?thesis*  
 by (*metis append-eq-conv-conj prefixE*)  
 qed

fun *mcp'* :: 'a list  $\Rightarrow$  'a list set  $\Rightarrow$  'a list **where**  
*mcp' z W = (THE p . mcp z W p)*

lemma *mcp'-intro* :  
 assumes *mcp z W p*  
 shows *mcp' z W = p*  
 using *assms mcp-unique* by (*metis mcp'.elim theI-unique*)

lemma *mcp-prefix-of-suffix* :  
 assumes *mcp (vs@xs) V vs*  
 and *prefix xs' xs*  
 shows *mcp (vs@xs') V vs*  
 proof (rule *ccontr*)  
 assume  $\neg \text{mcp } (vs @ xs') V vs$   
 then have  $\neg (\text{prefix } vs (vs @ xs') \wedge vs \in V \wedge$   
 $(\forall p' . (\text{prefix } p' (vs @ xs') \wedge p' \in V) \longrightarrow \text{length } p' \leq \text{length } vs))$   
 by *auto*  
 then have  $\neg (\forall p' . (\text{prefix } p' (vs @ xs') \wedge p' \in V) \longrightarrow \text{length } p' \leq \text{length } vs)$   
 using *assms(1)* by *auto*  
 then obtain *vs' where*  $vs' \in V \wedge \text{prefix } vs' (vs@xs) \wedge \text{length } vs < \text{length } vs'$   
 by (*meson assms(2) leI prefix-append prefix-order.dual-order.trans*)  
 then have  $\neg (\text{mcp } (vs@xs) V vs)$   
 by *auto*  
 then show *False*  
 using *assms(1)* by *auto*  
 qed

lemma *minimal-sequence-to-failure-extending-mcp* :  
 assumes *OFSM M1*  
 and *OFSM M2*  
 and *is-det-state-cover M2 V*  
 and *minimal-sequence-to-failure-extending V M1 M2 vs xs*  
 shows *mcp (map fst (vs@xs)) V (map fst vs)*  
 proof (rule *ccontr*)  
 assume  $\neg \text{mcp } (\text{map fst } (vs @ xs)) V (\text{map fst } vs)$   
 moreover have *prefix (map fst vs) (map fst (vs @ xs))*  
 by *auto*  
 moreover have  $(\text{map fst } vs) \in V$   
 using *mstfe-prefix-input-in-V assms(4)* by *auto*  
 ultimately obtain *v' where* *prefix v' (map fst (vs @ xs))*  
 $v' \in V$

```

      length v' > length (map fst vs)
using leI by auto

then obtain x' where (map fst (vs@xs)) = v'@x'
  using prefixE by blast

have vs@xs ∈ L M1 - L M2
  using assms(4) unfolding minimal-sequence-to-failure-extending.simps sequence-to-failure.simps
  by blast
then have vs@xs ∈ Lin M1 {map fst (vs@xs)}
  by (meson DiffE insertI1 language-state-for-inputs-map-fst)
have vs@xs ∈ Lin M1 {v'@x'}
  using ⟨map fst (vs @ xs) = v' @ x'⟩ ⟨vs @ xs ∈ Lin M1 {map fst (vs @ xs)}⟩
  by presburger

let ?vs' = take (length v') (vs@xs)
let ?xs' = drop (length v') (vs@xs)

have vs@xs = ?vs'@?xs'
  by (metis append-take-drop-id)

have ?vs' ∈ Lin M1 V
  by (metis (no-types) DiffE ⟨map fst (vs @ xs) = v' @ x'⟩ ⟨v' ∈ V⟩ ⟨vs @ xs ∈ L M1 - L M2⟩
    append-eq-conv-conj append-take-drop-id language-state-for-inputs-map-fst
    language-state-prefix take-map)

have sequence-to-failure M1 M2 (?vs' @ ?xs')
  by (metis (full-types) ⟨vs @ xs = take (length v') (vs @ xs) @ drop (length v') (vs @ xs)⟩
    assms(4) minimal-sequence-to-failure-extending.simps)

have length ?xs' < length xs
  using ⟨length (map fst vs) < length v'⟩ ⟨prefix v' (map fst (vs @ xs))⟩
    ⟨vs @ xs = take (length v') (vs @ xs) @ drop (length v') (vs @ xs)⟩ prefix-length-le
  by fastforce

show False
  by (meson ⟨length (drop (length v') (vs @ xs)) < length xs⟩
    ⟨sequence-to-failure M1 M2 (take (length v') (vs @ xs) @ drop (length v') (vs @ xs))⟩
    ⟨take (length v') (vs @ xs) ∈ Lin M1 V⟩ assms(4)
    minimal-sequence-to-failure-extending.elims(2))

```

qed

## 5.2 Function N

Function N narrows the sets of reaction to the deterministic state cover considered by the adaptive state counting algorithm to contain only relevant sequences. It is the main refinement of the original formulation of the algorithm as given in [2]. An example for the necessity for this refinement is given in [3].

```

fun N :: ('in × 'out) list ⇒ ('in, 'out, 'state) FSM ⇒ 'in list set ⇒ ('in × 'out) list set set
  where
    N io M V = { V'' ∈ Perm V M . (map fst (mcp' io V'')) = (mcp' (map fst io) V) }

```

**lemma** *N-nonempty* :

```

  assumes is-det-state-cover M2 V
  and OFSM M1
  and OFSM M2
  and asc-fault-domain M2 M1 m
  and io ∈ L M1

```

**shows**  $N \text{ io } M1 \ V \neq \{\}$

**proof** –

```

  have [] ∈ V
  using assms(1) det-state-cover-empty by blast

```

```

  have inputs M1 = inputs M2

```

```

using assms(4) by auto

have is-det-state-cover M2 V
  using assms by auto
moreover have finite (nodes M2)
  using assms(3) by auto
moreover have d-reachable M2 (initial M2)  $\subseteq$  nodes M2
  by auto
ultimately have finite V
  using det-state-cover-card[of M2 V]
  by (metis finite-if-finite-subsets-card-bdd infinite-subset is-det-state-cover.elims(2)
    surj-card-le)

obtain ioV where mcp (map fst io) V ioV
  using mcp-ex[OF  $\langle [] \in V \rangle \langle \text{finite } V \rangle$ ] by blast
then have ioV  $\in$  V
  by auto

```

— Proof sketch: - *ioV* uses only inputs of *M2* - *ioV* uses only inputs of *M1* - as *M1* completely spec.: ex. reaction of *M1* to *ioV* - this reaction is in some *V*”

```

obtain q2 where d-reaches M2 (initial M2) ioV q2
  using det-state-cover-d-reachable[OF assms(1)  $\langle \text{ioV} \in V \rangle$ ] by blast
then obtain ioV' ioP where io-path : length ioV = length ioV'
   $\wedge$  length ioV = length ioP
   $\wedge$  (path M2 ((ioV || ioV') || ioP) (initial M2))
   $\wedge$  target ((ioV || ioV') || ioP) (initial M2) = q2

  by auto

```

```

have well-formed M2
  using assms by auto

```

```

have map fst (map fst ((ioV || ioV') || ioP)) = ioV

```

```

proof –
  have length (ioV || ioV') = length ioP
    using io-path by simp
  then show ?thesis
    using io-path by auto

```

```

qed
moreover have set (map fst (map fst ((ioV || ioV') || ioP)))  $\subseteq$  inputs M2
  using path-input-containment[OF  $\langle \text{well-formed } M2 \rangle$ , of (ioV || ioV') || ioP initial M2 ]
  io-path
  by linarith
ultimately have set ioV  $\subseteq$  inputs M2
  by presburger

```

```

then have set ioV  $\subseteq$  inputs M1
  using assms by auto

```

```

then have Lin M1 {ioV}  $\neq$  {}
  using assms(2) language-state-for-inputs-nonempty by (metis FSM.nodes.initial)

```

```

have prefix ioV (map fst io)
  using  $\langle \text{mcp} (\text{map fst } io) V \text{ioV} \rangle$  mcp.simps by blast
then have length ioV  $\leq$  length (map fst io)
  using prefix-length-le by blast
then have length ioV  $\leq$  length io
  by auto

```

```

have (map fst io || map snd io)  $\in$  L M1
  using assms(5) by auto
moreover have length (map fst io) = length (map snd io)
  by auto
ultimately have (map fst io || map snd io)

```

$\in \text{language-state-for-input } M1 \text{ (initial } M1) \text{ (map fst io)}$

**unfolding** *language-state-def*  
**by** *(metis (mono-tags, lifting) <map fst io || map snd io ∈ L M1>*  
*language-state-for-input.simps mem-Collect-eq)*

**have**  $ioV = \text{take } (\text{length } ioV) \text{ (map fst io)}$   
**by** *(metis (no-types) <prefix ioV (map fst io)> append-eq-conv-conj prefixE)*

**then have**  $\text{take } (\text{length } ioV) io \in \text{language-state-for-input } M1 \text{ (initial } M1) ioV$   
**using** *language-state-for-input-take*  
**by** *(metis <map fst io || map snd io ∈ language-state-for-input M1 (initial M1) (map fst io)>*  
*zip-map-fst-snd)*

**then obtain**  $V''$  **where**  $V'' \in \text{Perm } V M1 \text{ take } (\text{length } ioV) io \in V''$   
**using** *perm-elim[OF assms(1-3) <inputs M1 = inputs M2> <ioV ∈ V>]* **by** *blast*

**have**  $ioV = \text{mcp}' (\text{map fst io}) V$   
**using** *<mcp (map fst io) V ioV> mcp'-intro* **by** *blast*

**have**  $\text{map fst } (\text{take } (\text{length } ioV) io) = ioV$   
**by** *(metis <ioV = take (length ioV) (map fst io)> take-map)*

**obtain**  $\text{mcp} V''$  **where**  $\text{mcp } io V'' \text{mcp} V''$   
**by** *(meson <V'' ∈ Perm V M1> <well-formed M2> assms(1) mcp-ex perm-elim-finite perm-empty)*

**have**  $\text{map fst } \text{mcp} V'' \in V$  **using** *perm-inputs*  
**using** *<V'' ∈ Perm V M1> <mcp io V'' mcp V''> mcp.simps* **by** *blast*

**have**  $\text{map fst } \text{mcp} V'' = ioV$   
**by** *(metis (no-types) <map fst (take (length ioV) io) = ioV> <map fst mcp V'' ∈ V>*  
*<mcp (map fst io) V ioV> <mcp io V'' mcp V''> <take (length ioV) io ∈ V''>*  
*map-mono-prefix mcp.elims(2) prefix-length-prefix prefix-order.dual-order.antisym*  
*take-is-prefix)*

**have**  $\text{map fst } (\text{mcp}' io V'') = \text{mcp}' (\text{map fst io}) V$   
**using** *<ioV = mcp' (map fst io) V> <map fst mcp V'' = ioV> <mcp io V'' mcp V''>* *mcp'-intro*  
**by** *blast*

**then show** *?thesis*  
**using** *<V'' ∈ Perm V M1>* **by** *fastforce*

**qed**

**lemma** *N-mcp-prefix* :

**assumes**  $\text{map fst } vs = \text{mcp}' (\text{map fst } (vs@xs)) V$   
**and**  $V'' \in N (vs@xs) M1 V$   
**and** *is-det-state-cover M2 V*  
**and** *well-formed M2*  
**and** *finite V*

**shows**  $vs \in V'' vs = \text{mcp}' (vs@xs) V''$

**proof** –

**have**  $\text{map fst } (\text{mcp}' (vs@xs) V'') = \text{mcp}' (\text{map fst } (vs@xs)) V$   
**using** *assms(2)* **by** *auto*  
**then have**  $\text{map fst } (\text{mcp}' (vs@xs) V'') = \text{map fst } vs$   
**using** *assms(1)* **by** *presburger*  
**then have**  $\text{length } (\text{mcp}' (vs@xs) V'') = \text{length } vs$   
**by** *(metis length-map)*

**have**  $\square \in V''$

**using** *perm-empty[OF assms(3)] N.simps assms(2)* **by** *blast*  
**moreover have** *finite V''*  
**using** *perm-elim-finite[OF assms(3,4)] N.simps assms(2)* **by** *blast*  
**ultimately obtain**  $p$  **where**  $\text{mcp } (vs@xs) V'' p$   
**using** *mcp-ex* **by** *auto*  
**then have**  $\text{mcp}' (vs@xs) V'' = p$

using *mcp'-intro* by *simp*

then have *prefix* (*mcp'* (*vs@xs*) *V''*) (*vs@xs*)  
 unfolding *mcp'.simps* *mcp.simps*  
 using  $\langle \text{mcp } (vs @ xs) V'' p \rangle$  *mcp.elims(2)* by *blast*  
 then show *vs = mcp'* (*vs@xs*) *V''*  
 by (*metis*  $\langle \text{length } (mcp' (vs @ xs) V'') = \text{length } vs \rangle$  *append-eq-append-conv* *prefix-def*)  
 show *vs*  $\in V''$   
 using  $\langle \text{mcp } (vs @ xs) V'' p \rangle$   $\langle \text{mcp}' (vs @ xs) V'' = p \rangle$   $\langle vs = \text{mcp}' (vs @ xs) V'' \rangle$   
 by *auto*  
 qed

### 5.3 Functions TS, C, RM

Function TTS defines the calculation of the test suite used by the adaptive state counting algorithm in an iterative way. It is defined using the three functions TS, C and RM where TS represents the test suite calculated up to some iteration, C contains the sequences considered for extension in some iteration, and RM contains the sequences of the corresponding C result that are not to be extended, which we also call removed sequences.

**abbreviation** *append-set* :: 'a list set  $\Rightarrow$  'a set  $\Rightarrow$  'a list set **where**  
*append-set* *T X*  $\equiv \{xs @ [x] \mid xs \ x . xs \in T \wedge x \in X\}$

**abbreviation** *append-sets* :: 'a list set  $\Rightarrow$  'a list set  $\Rightarrow$  'a list set **where**  
*append-sets* *T X*  $\equiv \{xs @ xs' \mid xs \ xs' . xs \in T \wedge xs' \in X\}$

**fun** *TS* :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  
 $\Rightarrow$  ('in, 'out) ATC set  $\Rightarrow$  'in list set  $\Rightarrow$  nat  $\Rightarrow$  nat  
 $\Rightarrow$  'in list set

**and** *C* :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  
 $\Rightarrow$  ('in, 'out) ATC set  $\Rightarrow$  'in list set  $\Rightarrow$  nat  $\Rightarrow$  nat  
 $\Rightarrow$  'in list set

**and** *RM* :: ('in, 'out, 'state1) FSM  $\Rightarrow$  ('in, 'out, 'state2) FSM  
 $\Rightarrow$  ('in, 'out) ATC set  $\Rightarrow$  'in list set  $\Rightarrow$  nat  $\Rightarrow$  nat  
 $\Rightarrow$  'in list set

**where**

*RM M2 M1*  $\Omega$  *V m 0* = {} |  
*TS M2 M1*  $\Omega$  *V m 0* = {} |  
*TS M2 M1*  $\Omega$  *V m (Suc 0)* = *V* |  
*C M2 M1*  $\Omega$  *V m 0* = {} |  
*C M2 M1*  $\Omega$  *V m (Suc 0)* = *V* |  
*RM M2 M1*  $\Omega$  *V m (Suc n)* =  
 {*xs'*  $\in$  *C M2 M1*  $\Omega$  *V m (Suc n)* .  
 ( $\neg (L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\})$ )  
 $\vee (\forall io \in L_{in} M1 \{xs'\} .$   
 $\exists V'' \in N \ io \ M1 \ V .$   
 $\exists S1 .$   
 $\exists \ vs \ xs .$   
 $io = (vs @ xs)$   
 $\wedge \text{mcp } (vs @ xs) V'' \ vs$   
 $\wedge S1 \subseteq \text{nodes } M2$   
 $\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$   
 $s1 \neq s2 \rightarrow$   
 $(\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V'' .$   
 $\forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V'' .$   
 $B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega ))$   
 $\wedge m < LB \ M2 \ M1 \ vs \ xs \ (TS \ M2 \ M1 \ \Omega \ V \ m \ n \cup V) \ S1 \ \Omega \ V'' \}$  |  
*C M2 M1*  $\Omega$  *V m (Suc n)* =  
 (*append-set* ((*C M2 M1*  $\Omega$  *V m n*) - (*RM M2 M1*  $\Omega$  *V m n*)) (*inputs M2*))  
 - (*TS M2 M1*  $\Omega$  *V m n*) |  
*TS M2 M1*  $\Omega$  *V m (Suc n)* =  
 (*TS M2 M1*  $\Omega$  *V m n*)  $\cup$  (*C M2 M1*  $\Omega$  *V m (Suc n)*)

**abbreviation** *lists-of-length* :: 'a set  $\Rightarrow$  nat  $\Rightarrow$  'a list set **where**  
*lists-of-length* X n  $\equiv$  {xs . length xs = n  $\wedge$  set xs  $\subseteq$  X}

**lemma** *append-lists-of-length-alt-def* :

*append-sets* T (*lists-of-length* X (Suc n)) = *append-set* (*append-sets* T (*lists-of-length* X n)) X

**proof**

**show** *append-sets* T (*lists-of-length* X (Suc n))  
 $\subseteq$  *append-set* (*append-sets* T (*lists-of-length* X n)) X

**proof**

**fix** tx **assume** tx  $\in$  *append-sets* T (*lists-of-length* X (Suc n))

**then obtain** t x **where** t@x = tx t  $\in$  T length x = Suc n set x  $\subseteq$  X

**by** blast

**then have** x  $\neq$  [] length (butlast x) = n

**by** auto

**moreover have** set (butlast x)  $\subseteq$  X

**using** <set x  $\subseteq$  X> **by** (meson dual-order.trans prefixeq-butlast set-mono-prefix)

**ultimately have** butlast x  $\in$  *lists-of-length* X n

**by** auto

**then have** t@(butlast x)  $\in$  *append-sets* T (*lists-of-length* X n)

**using** <t  $\in$  T> **by** blast

**moreover have** last x  $\in$  X

**using** <set x  $\subseteq$  X> <x  $\neq$  []> **by** auto

**ultimately have** t@(butlast x)@[last x]  $\in$  *append-set* (*append-sets* T (*lists-of-length* X n)) X

**by** auto

**then show** tx  $\in$  *append-set* (*append-sets* T (*lists-of-length* X n)) X

**using** <t@x = tx> **by** (simp add: <x  $\neq$  []>)

**qed**

**show** *append-set* (*append-sets* T (*lists-of-length* X n)) X

$\subseteq$  *append-sets* T (*lists-of-length* X (Suc n))

**proof**

**fix** tx **assume** tx  $\in$  *append-set* (*append-sets* T (*lists-of-length* X n)) X

**then obtain** tx' x **where** tx = tx' @ [x] tx'  $\in$  *append-sets* T (*lists-of-length* X n) x  $\in$  X

**by** blast

**then obtain** tx'' x' **where** tx''@x' = tx' tx''  $\in$  T length x' = n set x'  $\subseteq$  X

**by** blast

**then have** tx''@x'@[x] = tx

**by** (simp add: <tx = tx' @ [x]>)

**moreover have** tx''  $\in$  T

**by** (meson <tx''  $\in$  T>)

**moreover have** length (x'@[x]) = Suc n

**by** (simp add: <length x' = n>)

**moreover have** set (x'@[x])  $\subseteq$  X

**by** (simp add: <set x'  $\subseteq$  X> <x  $\in$  X>)

**ultimately show** tx  $\in$  *append-sets* T (*lists-of-length* X (Suc n))

**by** blast

**qed**

**qed**

## 5.4 Basic properties of the test suite calculation functions

**lemma** *C-step* :

**assumes** n > 0

**shows** C M2 M1  $\Omega$  V m (Suc n)  $\subseteq$  (*append-set* (C M2 M1  $\Omega$  V m n) (inputs M2)) - C M2 M1  $\Omega$  V m n

**proof** -

**let** ?TS =  $\lambda$  n . TS M2 M1  $\Omega$  V m n

**let** ?C =  $\lambda$  n . C M2 M1  $\Omega$  V m n

**let** ?RM =  $\lambda$  n . RM M2 M1  $\Omega$  V m n

**obtain** k **where** n-def[*simp*] : n = Suc k

**using** *assms not0-implies-Suc* **by** blast

**have** ?C (Suc n) = (*append-set* (?C n - ?RM n) (inputs M2)) - ?TS n

**using** n-def C.simps(3) **by** blast

**moreover have** ?C n  $\subseteq$  ?TS n

**using** n-def **by** (metis C.simps(2) TS.elims UnCI *assms neq0-conv subsetI*)

**ultimately show** ?C (Suc n)  $\subseteq$  *append-set* (?C n) (inputs M2) - ?C n

by blast  
qed

lemma *C-extension* :

$C\ M2\ M1\ \Omega\ V\ m\ (Suc\ n) \subseteq \text{append-sets}\ V\ (\text{lists-of-length}\ (\text{inputs}\ M2)\ n)$

proof (induction n)

case 0

then show ?case by auto

next

case (Suc k)

let ?TS =  $\lambda\ n.\ TS\ M2\ M1\ \Omega\ V\ m\ n$

let ?C =  $\lambda\ n.\ C\ M2\ M1\ \Omega\ V\ m\ n$

let ?RM =  $\lambda\ n.\ RM\ M2\ M1\ \Omega\ V\ m\ n$

have  $0 < Suc\ k$  by simp

have ?C (Suc (Suc k))  $\subseteq$  (append-set (?C (Suc k)) (inputs M2)) - ?C (Suc k)

using C-step[OF ‹0 < Suc k›] by blast

then have ?C (Suc (Suc k))  $\subseteq$  append-set (?C (Suc k)) (inputs M2)

by blast

moreover have append-set (?C (Suc k)) (inputs M2)

$\subseteq$  append-set (append-sets V (lists-of-length (inputs M2) k)) (inputs M2)

using Suc.IH by auto

ultimately have I-Step :

?C (Suc (Suc k))  $\subseteq$  append-set (append-sets V (lists-of-length (inputs M2) k)) (inputs M2)

by (meson order-trans)

show ?case

using append-lists-of-length-alt-def[symmetric, of V k inputs M2] I-Step

by presburger

qed

lemma *TS-union* :

shows  $TS\ M2\ M1\ \Omega\ V\ m\ i = (\bigcup j \in (\text{set } [0..<Suc\ i]) . C\ M2\ M1\ \Omega\ V\ m\ j)$

proof (induction i)

case 0

then show ?case by auto

next

case (Suc i)

let ?TS =  $\lambda\ n.\ TS\ M2\ M1\ \Omega\ V\ m\ n$

let ?C =  $\lambda\ n.\ C\ M2\ M1\ \Omega\ V\ m\ n$

let ?RM =  $\lambda\ n.\ RM\ M2\ M1\ \Omega\ V\ m\ n$

have ?TS (Suc i) = ?TS i  $\cup$  ?C (Suc i)

by (metis (no-types) C.simps(2) TS.simps(1) TS.simps(2) TS.simps(3) not0-implies-Suc  
sup-bot.right-neutral sup-commute)

then have ?TS (Suc i) =  $(\bigcup j \in (\text{set } [0..<Suc\ i]) . ?C\ j) \cup ?C\ (Suc\ i)$

using Suc.IH by simp

then show ?case

by auto

qed

lemma *C-disj-le-gz* :

assumes  $i \leq j$

and  $0 < i$

shows  $C\ M2\ M1\ \Omega\ V\ m\ i \cap C\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) = \{\}$

proof -

let ?TS =  $\lambda\ n.\ TS\ M2\ M1\ \Omega\ V\ m\ n$

let ?C =  $\lambda\ n.\ C\ M2\ M1\ \Omega\ V\ m\ n$

let ?RM =  $\lambda\ n.\ RM\ M2\ M1\ \Omega\ V\ m\ n$

```

have  $Suc\ 0 < Suc\ j$ 
  using  $assms(1-2)$  by auto
then obtain  $k$  where  $Suc\ j = Suc\ (Suc\ k)$ 
  using  $not0\ implies\ Suc$  by blast
then have  $?C\ (Suc\ j) = (append\ set\ (?C\ j - ?RM\ j)\ (inputs\ M2)) - ?TS\ j$ 
  using  $C.simps(3)$  by blast
then have  $?C\ (Suc\ j) \cap ?TS\ j = \{\}$ 
  by blast
moreover have  $?C\ i \subseteq ?TS\ j$ 
  using  $assms(1)\ TS\ union[of\ M2\ M1\ \Omega\ V\ m\ j]$  by fastforce
ultimately show  $?thesis$ 
  by blast
qed

```

```

lemma  $C\ disj\ lt$  :
  assumes  $i < j$ 
shows  $C\ M2\ M1\ \Omega\ V\ m\ i \cap C\ M2\ M1\ \Omega\ V\ m\ j = \{\}$ 
proof (cases  $i$ )
  case 0
  then show  $?thesis$  by auto
next
  case (Suc  $k$ )
  then show  $?thesis$ 
    using  $C\ disj\ le\ gz$ 
    by (metis  $assms\ gr\ implies\ not0\ less\ Suc\ eq\ le\ old.nat.exhaust\ zero\ less\ Suc$ )
qed

```

```

lemma  $C\ disj$  :
  assumes  $i \neq j$ 
shows  $C\ M2\ M1\ \Omega\ V\ m\ i \cap C\ M2\ M1\ \Omega\ V\ m\ j = \{\}$ 
  by (metis  $C\ disj\ lt\ Int\ commute\ antisym\ conv3\ assms$ )

```

```

lemma  $RM\ subset$  :  $RM\ M2\ M1\ \Omega\ V\ m\ i \subseteq C\ M2\ M1\ \Omega\ V\ m\ i$ 
proof (cases  $i$ )
  case 0
  then show  $?thesis$  by auto
next
  case (Suc  $n$ )
  then show  $?thesis$ 
    using  $RM.simps(2)$  by blast
qed

```

```

lemma  $RM\ disj$  :
  assumes  $i \leq j$ 
  and  $0 < i$ 
shows  $RM\ M2\ M1\ \Omega\ V\ m\ i \cap RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) = \{\}$ 
proof -
  let  $?TS = \lambda\ n.\ TS\ M2\ M1\ \Omega\ V\ m\ n$ 
  let  $?C = \lambda\ n.\ C\ M2\ M1\ \Omega\ V\ m\ n$ 
  let  $?RM = \lambda\ n.\ RM\ M2\ M1\ \Omega\ V\ m\ n$ 

  have  $?RM\ i \subseteq ?C\ i\ ?RM\ (Suc\ j) \subseteq ?C\ (Suc\ j)$ 
    using  $RM\ subset$  by blast+
  moreover have  $?C\ i \cap ?C\ (Suc\ j) = \{\}$ 
    using  $C\ disj\ le\ gz[OF\ assms]$  by assumption
  ultimately show  $?thesis$ 
    by blast
qed

```

**lemma** *T-extension* :  
**assumes**  $n > 0$   
**shows**  $TS\ M2\ M1\ \Omega\ V\ m\ (Suc\ n) - TS\ M2\ M1\ \Omega\ V\ m\ n$   
 $\subseteq (append\text{-}set\ (TS\ M2\ M1\ \Omega\ V\ m\ n)\ (inputs\ M2)) - TS\ M2\ M1\ \Omega\ V\ m\ n$   
**proof** –  
**let**  $?TS = \lambda\ n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?C = \lambda\ n . C\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?RM = \lambda\ n . RM\ M2\ M1\ \Omega\ V\ m\ n$   
  
**obtain**  $k$  **where**  $n\text{-}def[simp] : n = Suc\ k$   
**using** *assms not0-implies-Suc*  
**by** *blast*  
  
**have**  $?C\ (Suc\ n) = (append\text{-}set\ (?C\ n - ?RM\ n)\ (inputs\ M2)) - ?TS\ n$   
**using** *n-def* **using** *C.simps(3)* **by** *blast*  
**then have**  $?C\ (Suc\ n) \subseteq append\text{-}set\ (?C\ n)\ (inputs\ M2) - ?TS\ n$   
**by** *blast*  
**moreover have**  $?C\ n \subseteq ?TS\ n$  **using** *TS-union[of M2 M1 Ω V m n]*  
**by** *fastforce*  
**ultimately have**  $?C\ (Suc\ n) \subseteq append\text{-}set\ (?TS\ n)\ (inputs\ M2) - ?TS\ n$   
**by** *blast*  
**moreover have**  $?TS\ (Suc\ n) - ?TS\ n \subseteq ?C\ (Suc\ n)$   
**using** *TS.simps(3)[of M2 M1 Ω V m k]* **using** *n-def* **by** *blast*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *append-set-prefix* :  
**assumes**  $xs \in append\text{-}set\ T\ X$   
**shows**  $butlast\ xs \in T$   
**using** *assms* **by** *auto*

**lemma** *C-subset* :  $C\ M2\ M1\ \Omega\ V\ m\ i \subseteq TS\ M2\ M1\ \Omega\ V\ m\ i$   
**by** (*simp add: TS-union*)

**lemma** *TS-subset* :  
**assumes**  $i \leq j$   
**shows**  $TS\ M2\ M1\ \Omega\ V\ m\ i \subseteq TS\ M2\ M1\ \Omega\ V\ m\ j$   
**proof** –  
**have**  $TS\ M2\ M1\ \Omega\ V\ m\ i = (\bigcup\ k \in (set\ [0..<Suc\ i]) . C\ M2\ M1\ \Omega\ V\ m\ k)$   
 $TS\ M2\ M1\ \Omega\ V\ m\ j = (\bigcup\ k \in (set\ [0..<Suc\ j]) . C\ M2\ M1\ \Omega\ V\ m\ k)$   
**using** *TS-union* **by** *assumption+*  
**moreover have**  $set\ [0..<Suc\ i] \subseteq set\ [0..<Suc\ j]$   
**using** *assms* **by** *auto*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *C-immediate-prefix-containment* :  
**assumes**  $vs@xs \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ (Suc\ i))$   
**and**  $xs \neq []$   
**shows**  $vs@(butlast\ xs) \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ i) - RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ i)$   
**proof** (*rule ccontr*)  
**let**  $?TS = \lambda\ n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?C = \lambda\ n . C\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?RM = \lambda\ n . RM\ M2\ M1\ \Omega\ V\ m\ n$

**assume**  $vs\ @\ butlast\ xs \notin C\ M2\ M1\ \Omega\ V\ m\ (Suc\ i) - RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ i)$

**have**  $?C\ (Suc\ (Suc\ i)) \subseteq append\text{-}set\ (?C\ (Suc\ i) - ?RM\ (Suc\ i))\ (inputs\ M2)$   
**using** *C.simps(3)* **by** *blast*  
**then have**  $?C\ (Suc\ (Suc\ i)) \subseteq append\text{-}set\ (?C\ (Suc\ i) - ?RM\ (Suc\ i))\ UNIV$

by *blast*  
**moreover have**  $vs @ xs \notin \text{append-set } (?C (Suc i) - ?RM (Suc i)) UNIV$   
**proof** –  
 have  $\forall as a. vs @ xs \neq as @ [a]$   
      $\vee as \notin C M2 M1 \Omega V m (Suc i) - RM M2 M1 \Omega V m (Suc i)$   
      $\vee a \notin UNIV$   
 by  $(metis \langle vs @ butlast xs \notin C M2 M1 \Omega V m (Suc i) - RM M2 M1 \Omega V m (Suc i) \rangle$   
      $assms(2) butlast-append butlast-snoc)$   
**then show** *?thesis*  
 by *blast*  
**qed**  
**ultimately have**  $vs @ xs \notin ?C (Suc (Suc i))$   
 by *blast*  
**then show** *False*  
 using *assms(1)* by *blast*  
**qed**

**lemma** *TS-immediate-prefix-containment* :

**assumes**  $vs@xs \in TS M2 M1 \Omega V m i$   
**and**  $mcp (vs@xs) V vs$   
**and**  $0 < i$

**shows**  $vs@(butlast xs) \in TS M2 M1 \Omega V m i$

**proof** –

**let**  $?TS = \lambda n. TS M2 M1 \Omega V m n$   
**let**  $?C = \lambda n. C M2 M1 \Omega V m n$   
**let**  $?RM = \lambda n. RM M2 M1 \Omega V m n$

**obtain**  $j$  **where**  $j\text{-def} : j \leq i \wedge vs@xs \in ?C j$

using *assms(1)* *TS-union[where i=i]*

**proof** –

**assume**  $a1: \bigwedge j. j \leq i \wedge vs @ xs \in C M2 M1 \Omega V m j \implies thesis$

**obtain**  $nn :: nat \text{ set} \implies (nat \implies 'a \text{ list set}) \implies 'a \text{ list} \implies nat$  **where**

$f2: \forall x0 x1 x2. (\exists v3. v3 \in x0 \wedge x2 \in x1 v3) = (nn x0 x1 x2 \in x0 \wedge x2 \in x1 (nn x0 x1 x2))$

by *moura*

**have**  $vs @ xs \in UNION (set [0..<Suc i]) (C M2 M1 \Omega V m)$

by  $(metis \langle \bigwedge \Omega V T S M2 M1. TS M2 M1 \Omega V m i = (\bigcup_{j \in set [0..<Suc i]}. C M2 M1 \Omega V m j) \rangle$   
 $\langle vs @ xs \in TS M2 M1 \Omega V m i \rangle)$

**then have**  $nn (set [0..<Suc i]) (C M2 M1 \Omega V m) (vs @ xs) \in set [0..<Suc i]$

$\wedge vs @ xs \in C M2 M1 \Omega V m (nn (set [0..<Suc i]) (C M2 M1 \Omega V m) (vs @ xs))$

using *f2* by *blast*

**then show** *?thesis*

using *a1* by  $(metis (no-types) atLeastLessThan-iff leD not-less-eq-eq set-upt)$

**qed**

**show** *?thesis*

**proof**  $(cases j)$

**case**  $0$

**then have**  $?C j = \{\}$

by *auto*

**moreover have**  $vs@xs \in \{\}$

using *j-def 0* by *auto*

**ultimately show** *?thesis*

by *auto*

**next**

**case**  $(Suc k)$

**then show** *?thesis*

**proof**  $(cases k)$

**case**  $0$

**then have**  $?C j = V$

using *Suc* by *auto*

**then have**  $vs@xs \in V$

using *j-def* by *auto*

**then have**  $mcp (vs@xs) V (vs@xs)$

using *assms(2)* by *auto*

```

then have  $vs@xs = vs$ 
  using  $assms(2)$  mcp-unique by auto
then have  $butlast\ xs = []$ 
  by auto
then show ?thesis
  using  $\langle vs @ xs = vs \rangle$   $assms(1)$  by auto
next
case  $(Suc\ n)$ 
assume  $j-assms : j = Suc\ k$ 
   $k = Suc\ n$ 
then have  $?C\ (Suc\ (Suc\ n)) = append-set\ (?C\ (Suc\ n) - ?RM\ (Suc\ n))\ (inputs\ M2) - ?TS\ (Suc\ n)$ 
  using  $C.simps(3)$  by blast
then have  $?C\ (Suc\ (Suc\ n)) \subseteq append-set\ (?C\ (Suc\ n))\ (inputs\ M2)$ 
  by blast

have  $vs@xs \in ?C\ (Suc\ (Suc\ n))$ 
  using  $j-assms\ j-def$  by blast

have  $butlast\ (vs@xs) \in ?C\ (Suc\ n)$ 
proof -
  show ?thesis
  by  $(meson\ \langle ?C\ (Suc\ (Suc\ n)) \subseteq append-set\ (?C\ (Suc\ n))\ (inputs\ M2) \rangle$ 
     $\langle vs @ xs \in ?C\ (Suc\ (Suc\ n)) \rangle$  append-set-prefix subsetCE)
qed

moreover have  $xs \neq []$ 
proof -
  have  $1 \leq k$ 
  using  $j-assms$  by auto
then have  $?C\ j \cap ?C\ 1 = \{\}$ 
  using  $C-disj-le-gz[of\ 1\ k]$   $j-assms(1)$  less-numeral-extra(1) by blast
then have  $?C\ j \cap V = \{\}$ 
  by auto
then have  $vs@xs \notin V$ 
  using  $j-def$  by auto
then show ?thesis
  using  $assms(2)$  by auto
qed

ultimately have  $vs@(butlast\ xs) \in ?C\ (Suc\ n)$ 
  by  $(simp\ add:\ butlast-append)$ 

have  $Suc\ n < Suc\ j$ 
  using  $j-assms$  by auto
have  $?C\ (Suc\ n) \subseteq ?TS\ j$ 
  using  $TS-union[of\ M2\ M1\ \Omega\ V\ m\ j]$   $\langle Suc\ n < Suc\ j \rangle$ 
  by  $(metis\ UN-upper\ atLeast-upt\ lessThan-iff)$ 

have  $vs @ butlast\ xs \in TS\ M2\ M1\ \Omega\ V\ m\ j$ 
  using  $\langle vs@(butlast\ xs) \in ?C\ (Suc\ n) \rangle$   $\langle ?C\ (Suc\ n) \subseteq ?TS\ j \rangle$   $j-def$ 
  by auto
then show ?thesis
  using  $j-def\ TS-subset[of\ j\ i]$ 
  by blast
qed
qed
qed

lemma TS-prefix-containment :
assumes  $vs@xs \in TS\ M2\ M1\ \Omega\ V\ m\ i$ 
and  $mcp\ (vs@xs)\ V\ vs$ 
and  $prefix\ xs'\ xs$ 
shows  $vs@xs' \in TS\ M2\ M1\ \Omega\ V\ m\ i$ 

```

— Proof sketch: Perform induction on length difference, as from each prefix it is possible to deduce the desired property for the prefix one element smaller than it via above results

```

using assms proof (induction length xs - length xs' arbitrary: xs')
  case 0
  then have  $xs = xs'$ 
    by (metis append-Nil2 append-eq-conv-conj gr-implies-not0 length-drop length-greater-0-conv prefixE)
  then show ?case
    using 0 by auto
next
case (Suc k)
have  $0 < i$ 
  using assms(1) using Suc.hyps(2) append-eq-append-conv assms(2) by auto

show ?case
proof (cases xs')
  case Nil
  then show ?thesis
    by (metis (no-types, opaque-lifting) <0 < i> TS.simps(2) TS-subset append-Nil2 assms(2)
      contra-subsetD leD mcp.elims(2) not-less-eq-eq)
next
case (Cons a list)
  then show ?thesis
proof (cases xs = xs')
  case True
  then show ?thesis
    using assms(1) by simp
next
case False
  then obtain  $xs''$  where  $xs = xs' @ xs''$ 
    using Suc.prem(3) prefixE by blast
  then have  $xs'' \neq []$ 
    using False by auto
  then have  $k = \text{length } xs - \text{length } (xs' @ [hd \ xs''])$ 
    using  $\langle xs = xs' @ xs'' \rangle$  Suc.hyps(2) by auto
  moreover have prefix  $(xs' @ [hd \ xs'']) \ xs$ 
    using  $\langle xs = xs' @ xs'' \rangle$   $\langle xs'' \neq [] \rangle$ 
    by (metis Cons-prefix-Cons list.exhaust-sel prefix-code(1) same-prefix-prefix)
  ultimately have  $vs @ (xs' @ [hd \ xs'']) \in TS \ M2 \ M1 \ \Omega \ V \ m \ i$ 
    using Suc.hyps(1)[OF - Suc.prem(1,2)] by simp

have mcp  $(vs @ xs' @ [hd \ xs'']) \ V \ vs$ 
  using  $\langle xs = xs' @ xs'' \rangle$   $\langle xs'' \neq [] \rangle$  assms(2)
proof -
  obtain aas :: 'a list  $\Rightarrow$  'a list set  $\Rightarrow$  'a list  $\Rightarrow$  'a list where
     $\forall x0 \ x1 \ x2. (\exists v3. (\text{prefix } v3 \ x2 \wedge v3 \in x1) \wedge \neg \text{length } v3 \leq \text{length } x0)$ 
     $= ((\text{prefix } (aas \ x0 \ x1 \ x2) \ x2 \wedge aas \ x0 \ x1 \ x2 \in x1)$ 
     $\wedge \neg \text{length } (aas \ x0 \ x1 \ x2) \leq \text{length } x0)$ 
  by moura
  then have f1:  $\forall as \ A \ asa. (\neg \text{mcp } as \ A \ asa$ 
     $\vee \text{prefix } asa \ as \wedge asa \in A \wedge (\forall asb. (\neg \text{prefix } asb \ as \vee asb \notin A)$ 
     $\vee \text{length } asb \leq \text{length } asa))$ 
     $\wedge (\text{mcp } as \ A \ asa$ 
     $\vee \neg \text{prefix } asa \ as$ 
     $\vee asa \notin A$ 
     $\vee (\text{prefix } (aas \ asa \ A \ as) \ as \wedge aas \ asa \ A \ as \in A)$ 
     $\wedge \neg \text{length } (aas \ asa \ A \ as) \leq \text{length } asa)$ 
  by auto
  obtain aasa :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  'a list where
    f2:  $\forall x0 \ x1. (\exists v2. x0 = x1 @ v2) = (x0 = x1 @ aasa \ x0 \ x1)$ 
  by moura
  then have f3:  $([] @ [hd \ xs'']) @ aasa \ (xs' @ xs'') \ (xs' @ [hd \ xs''])$ 
     $= ([] @ [hd \ xs'']) @ aasa \ (([] @ [hd \ xs''])$ 
     $@ aasa \ (xs' @ xs'') \ (xs' @ [hd \ xs''])) \ ([] @ [hd \ xs''])$ 
  by (meson prefixE prefixI)
  have  $xs' @ xs'' = (xs' @ [hd \ xs'']) @ aasa \ (xs' @ xs'') \ (xs' @ [hd \ xs''])$ 

```

```

    using f2 by (metis (no-types) ⟨prefix (xs' @ [hd xs']) xs⟩ ⟨xs = xs' @ xs'⟩ prefixE)
  then have (vs @ (a # list) @ [hd xs']) @ aasa (([] @ [hd xs'])
    @ aasa (xs' @ xs') (xs' @ [hd xs'])) ([] @ [hd xs'])
    = vs @ xs
    using f3 by (simp add: ⟨xs = xs' @ xs'⟩ local.Cons)
  then have ¬ prefix (aas vs V (vs @ xs' @ [hd xs'])) (vs @ xs' @ [hd xs'])
    ∨ aas vs V (vs @ xs' @ [hd xs']) ∉ V
    ∨ length (aas vs V (vs @ xs' @ [hd xs'])) ≤ length vs
    using f1 by (metis (no-types) ⟨mcp (vs @ xs) V vs⟩ local.Cons prefix-append)
  then show ?thesis
    using f1 by (meson ⟨mcp (vs @ xs) V vs⟩ prefixI)
qed

then have vs @ butlast (xs' @ [hd xs']) ∈ TS M2 M1 Ω V m i
  using TS-immediate-prefix-containment
    [OF ⟨vs @ (xs' @ [hd xs']) ∈ TS M2 M1 Ω V m i⟩ - ⟨0 < i⟩]
  by simp

moreover have xs' = butlast (xs' @ [hd xs'])
  using ⟨xs'' ≠ []⟩ by simp

ultimately show ?thesis
  by simp
qed
qed
qed

```

```

lemma C-index :
  assumes vs @ xs ∈ C M2 M1 Ω V m i
  and mcp (vs@xs) V vs
  shows Suc (length xs) = i
  using assms proof (induction xs arbitrary: i rule: rev-induct)
  case Nil
  then have vs @ [] ∈ C M2 M1 Ω V m 1
    by auto
  then have vs @ [] ∈ C M2 M1 Ω V m (Suc (length []))
    by simp

  show ?case
  proof (rule ccontr)
  assume Suc (length []) ≠ i
  moreover have vs @ [] ∈ C M2 M1 Ω V m i ∩ C M2 M1 Ω V m (Suc (length []))
    using Nil.premis(1) ⟨vs @ [] ∈ C M2 M1 Ω V m (Suc (length []))⟩ by auto
  ultimately show False
    using C-disj by blast
  qed
next
  case (snoc x xs')

  let ?TS = λ n . TS M2 M1 Ω V m n
  let ?C = λ n . C M2 M1 Ω V m n
  let ?RM = λ n . RM M2 M1 Ω V m n

  have vs @ xs' @ [x] ∉ V
    using snoc.premis(2) by auto
  then have vs @ xs' @ [x] ∉ ?C 1
    by auto
  moreover have vs @ xs' @ [x] ∉ ?C 0
    by auto

```

ultimately have  $1 < i$   
 using *snoc.premis(1)* by (*metis less-one linorder-neqE-nat*)

then have  $vs @ butlast (xs' @ [x]) \in C M2 M1 \Omega V m (i-1)$   
 proof –  
 have  $Suc\ 0 < i$   
 using  $\langle 1 < i \rangle$  by *auto*  
 then have  $f1: Suc (i - Suc (Suc\ 0)) = i - Suc\ 0$   
 using *Suc-diff-Suc* by *presburger*  
 have  $0 < i$   
 by (*metis (no-types) One-nat-def Suc-lessD \langle 1 < i \rangle*)  
 then show *?thesis*  
 using  $f1$  by (*metis C-immediate-prefix-containment DiffD1 One-nat-def Suc-pred' snoc.premis(1) snoc-eq-iff-butlast*)

qed

moreover have  $mcp (vs @ butlast (xs' @ [x])) V vs$   
 by (*meson mcp-prefix-of-suffix prefixeq-butlast snoc.premis(2)*)

ultimately have  $Suc (length\ xs') = i-1$   
 using *snoc.IH* by *simp*

then show *?case*  
 by *auto*

qed

lemma *TS-index* :

assumes  $vs @ xs \in TS\ M2\ M1\ \Omega\ V\ m\ i$   
 and  $mcp (vs@xs) V vs$

shows  $Suc (length\ xs) \leq i$   $vs@xs \in C\ M2\ M1\ \Omega\ V\ m\ (Suc (length\ xs))$

proof –

let  $?TS = \lambda n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
 let  $?C = \lambda n . C\ M2\ M1\ \Omega\ V\ m\ n$   
 let  $?RM = \lambda n . RM\ M2\ M1\ \Omega\ V\ m\ n$

obtain  $j$  where  $j < Suc\ i$   $vs@xs \in ?C\ j$   
 using *TS-union[of M2 M1 Ω V m i]*  
 by (*metis (full-types) UN-iff assms(1) atLeastLessThan-iff set-upt*)

then have  $Suc (length\ xs) = j$   
 using *C-index assms(2)* by *blast*

then show  $Suc (length\ xs) \leq i$   
 using  $\langle j < Suc\ i \rangle$  by *auto*

show  $vs@xs \in C\ M2\ M1\ \Omega\ V\ m\ (Suc (length\ xs))$   
 using  $\langle vs@xs \in ?C\ j \rangle \langle Suc (length\ xs) = j \rangle$  by *auto*

qed

lemma *C-extension-options* :

assumes  $vs @ xs \in C\ M2\ M1\ \Omega\ V\ m\ i$   
 and  $mcp (vs @ xs @ [x]) V vs$   
 and  $x \in inputs\ M2$   
 and  $0 < i$

shows  $vs@xs@[x] \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ i) \vee vs@xs \in RM\ M2\ M1\ \Omega\ V\ m\ i$

proof (*cases vs@xs ∈ RM M2 M1 Ω V m i*)

case *True*  
 then show *?thesis* by *auto*

next

case *False*

let  $?TS = \lambda n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
 let  $?C = \lambda n . C\ M2\ M1\ \Omega\ V\ m\ n$   
 let  $?RM = \lambda n . RM\ M2\ M1\ \Omega\ V\ m\ n$

obtain  $k$  where  $i = Suc\ k$   
 using *assms(4) gr0-implies-Suc* by *blast*  
 then have  $?C (Suc\ i) = append-set (?C\ i - ?RM\ i) (inputs\ M2) - ?TS\ i$

**using**  $C.simps(3)$  **by** *blast*  
**moreover have**  $vs@xs \in ?C\ i - ?RM\ i$   
**using**  $assms(1)$  **False** **by** *blast*  
**ultimately have**  $vs@xs@[x] \in \text{append-set } (?C\ i - ?RM\ i)$  (*inputs*  $M2$ )  
**by** (*simp add: assms(3)*)  
**moreover have**  $vs@xs@[x] \notin ?TS\ i$   
**proof** (*rule ccontr*)  
**assume**  $\neg vs\ @\ xs\ @\ [x] \notin ?TS\ i$   
**then obtain**  $j$  **where**  $j < Suc\ i$   $vs@xs@[x] \in ?C\ j$   
**using**  $TS\text{-union}[of\ M2\ M1\ \Omega\ V\ m\ i]$  **by** *fastforce*  
**then have**  $Suc\ (\text{length } (xs@[x])) = j$   
**using**  $C\text{-index } assms(2)$  **by** *blast*  
  
**then have**  $Suc\ (\text{length } (xs@[x])) < Suc\ i$   
**using**  $\langle j < Suc\ i \rangle$  **by** *auto*  
**moreover have**  $Suc\ (\text{length } xs) = i$   
**using**  $C\text{-index}$   
**by** (*metis assms(1) assms(2) mcp-prefix-of-suffix prefixI*)  
**ultimately have**  $Suc\ (\text{length } (xs@[x])) < Suc\ (Suc\ (\text{length } xs))$   
**by** *auto*  
**then show** *False*  
**by** *auto*  
**qed**  
  
**ultimately show** *?thesis*  
**by** (*simp add: \langle ?C (Suc i) = append-set (?C i - ?RM i) (inputs M2) - ?TS i \rangle*)  
**qed**

**lemma** *TS-non-containment-causes* :

**assumes**  $vs@xs \notin TS\ M2\ M1\ \Omega\ V\ m\ i$   
**and**  $mcp\ (vs@xs)\ V\ vs$   
**and**  $set\ xs \subseteq \text{inputs } M2$   
**and**  $0 < i$   
**shows**  $(\exists\ xr\ j . xr \neq xs \wedge \text{prefix } xr\ xs \wedge j \leq i \wedge vs@xr \in RM\ M2\ M1\ \Omega\ V\ m\ j)$   
 $\vee (\exists\ xc . xc \neq xs \wedge \text{prefix } xc\ xs \wedge vs@xc \in (C\ M2\ M1\ \Omega\ V\ m\ i) - (RM\ M2\ M1\ \Omega\ V\ m\ i))$   
**(is**  $?PrefPreviouslyRemoved \vee ?PrefJustContained$   
 $\neg ((\exists\ xr\ j . xr \neq xs \wedge \text{prefix } xr\ xs \wedge j \leq i \wedge vs@xr \in RM\ M2\ M1\ \Omega\ V\ m\ j)$   
 $\wedge (\exists\ xc . xc \neq xs \wedge \text{prefix } xc\ xs \wedge vs@xc \in (C\ M2\ M1\ \Omega\ V\ m\ i) - (RM\ M2\ M1\ \Omega\ V\ m\ i)))$   
— If a sequence is not contained in TS up to (incl.) iteration  $i$ , then either a prefix of it has been removed or a prefix of it is contained in the C set for iteration  $i$   
**proof** –

**let**  $?TS = \lambda\ n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?C = \lambda\ n . C\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?RM = \lambda\ n . RM\ M2\ M1\ \Omega\ V\ m\ n$

**show**  $?PrefPreviouslyRemoved \vee ?PrefJustContained$

**proof** (*rule ccontr*)  
**assume**  $\neg (?PrefPreviouslyRemoved \vee ?PrefJustContained)$   
**then have**  $\neg ?PrefPreviouslyRemoved \neg ?PrefJustContained$  **by** *auto*

**have**  $\neg (\exists\ xr\ j . \text{prefix } xr\ xs \wedge j \leq i \wedge vs\ @\ xr \in ?RM\ j)$

**proof**

**assume**  $\exists\ xr\ j . \text{prefix } xr\ xs \wedge j \leq i \wedge vs\ @\ xr \in RM\ M2\ M1\ \Omega\ V\ m\ j$

**then obtain**  $xr\ j$  **where**  $\text{prefix } xr\ xs \wedge j \leq i \wedge vs\ @\ xr \in ?RM\ j$

**by** *blast*

**then show** *False*

**proof** (*cases*  $xr = xs$ )

```

case True
then have  $vs @ xs \in ?RM\ j$  using  $\langle vs @ xr \in ?RM\ j \rangle$  by auto
then have  $vs @ xs \in ?TS\ j$ 
  using C-subset RM-subset  $\langle vs @ xr \in ?RM\ j \rangle$  by blast
then have  $vs @ xs \in ?TS\ i$ 
  using TS-subset  $\langle j \leq i \rangle$  by blast
then show ?thesis using assms(1) by blast
next
case False
then show ?thesis
  using  $\langle \neg ?PrefPreviouslyRemoved \rangle \langle prefix\ xr\ xs \rangle \langle j \leq i \rangle \langle vs @ xr \in ?RM\ j \rangle$ 
  by blast
qed
qed

have  $vs \in V$  using assms(2) by auto
then have  $vs \in ?C\ 1$  by auto

have  $\bigwedge k . (1 \leq Suc\ k \wedge Suc\ k \leq i) \longrightarrow vs @ (take\ k\ xs) \in ?C\ (Suc\ k) - ?RM\ (Suc\ k)$ 
proof
  fix k assume  $1 \leq Suc\ k \wedge Suc\ k \leq i$ 
  then show  $vs @ (take\ k\ xs) \in ?C\ (Suc\ k) - ?RM\ (Suc\ k)$ 
  proof (induction k)
    case 0
    show ?case using  $\langle vs \in ?C\ 1 \rangle$ 
    by (metis 0.premys DiffI One-nat-def
       $\langle \neg (\exists\ xr\ j. prefix\ xr\ xs \wedge j \leq i \wedge vs @ xr \in RM\ M2\ M1\ \Omega\ V\ m\ j) \rangle$ 
      append-Nil2 take-0 take-is-prefix)
    next
    case (Suc k)

    have  $1 \leq Suc\ k \wedge Suc\ k \leq i$ 
    using Suc.premys by auto
    then have  $vs @ take\ k\ xs \in ?C\ (Suc\ k)$ 
    using Suc.IH by simp

    moreover have  $vs @ take\ k\ xs \notin ?RM\ (Suc\ k)$ 
    using  $\langle 1 \leq Suc\ k \wedge Suc\ k \leq i \rangle \langle \neg ?PrefPreviouslyRemoved \rangle$  take-is-prefix Suc.IH
    by blast

    ultimately have  $vs @ take\ k\ xs \in (?C\ (Suc\ k)) - (?RM\ (Suc\ k))$ 
    by blast

    have  $k < length\ xs$ 
    proof (rule ccontr)
      assume  $\neg k < length\ xs$ 
      then have  $vs @ xs \in ?C\ (Suc\ k)$  using  $\langle vs @ take\ k\ xs \in ?C\ (Suc\ k) \rangle$ 
      by simp
      have  $vs @ xs \in ?TS\ i$ 
      by (metis C-subset TS-subset  $\langle 1 \leq Suc\ k \wedge Suc\ k \leq i \rangle \langle vs @ xs \in ?C\ (Suc\ k) \rangle$ 
        contra-subsetD)
      then show False
      using assms(1) by simp
    qed

    moreover have  $set\ xs \subseteq inputs\ M2$ 
    using assms(3) by auto
    ultimately have  $last\ (take\ (Suc\ k)\ xs) \in inputs\ M2$ 
    by (simp add: subset-eq take-Suc-conv-app-nth)

    have  $vs @ take\ (Suc\ k)\ xs \in append\ set\ ((?C\ (Suc\ k)) - (?RM\ (Suc\ k)))\ (inputs\ M2)$ 
    proof -
      have f1:  $xs ! k \in inputs\ M2$ 
      by (meson  $\langle k < length\ xs \rangle \langle set\ xs \subseteq inputs\ M2 \rangle$  nth-mem subset-iff)
      have  $vs @ take\ (Suc\ k)\ xs = (vs @ take\ k\ xs) @ [xs ! k]$ 
      by (simp add:  $\langle k < length\ xs \rangle$  take-Suc-conv-app-nth)
  
```

**then show** *?thesis*  
**using** *f1*  $\langle vs @ take\ k\ xs \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k) - RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ k) \rangle$  **by** *blast*  
**qed**

**moreover have**  $vs @ take\ (Suc\ k)\ xs \notin ?TS\ (Suc\ k)$

**proof**

**assume**  $vs @ take\ (Suc\ k)\ xs \in ?TS\ (Suc\ k)$

**then have**  $Suc\ (length\ (take\ (Suc\ k)\ xs)) \leq Suc\ k$

**using** *TS-index(1) assms(2) mcp-prefix-of-suffix take-is-prefix* **by** *blast*

**moreover have**  $Suc\ (length\ (take\ k\ xs)) = Suc\ k$  **using** *C-index*  $\langle vs @ take\ k\ xs \in ?C\ (Suc\ k) \rangle$

**by** *(metis assms(2) mcp-prefix-of-suffix take-is-prefix)*

**ultimately show** *False* **using**  $\langle k < length\ xs \rangle$

**by** *simp*

**qed**

**show**  $vs @ take\ (Suc\ k)\ xs \in ?C\ (Suc\ (Suc\ k)) - ?RM\ (Suc\ (Suc\ k))$

**using** *C.simps(3)[of M2 M1  $\Omega$  V m k]*

**by** *(metis (no-types, lifting) DiffI Suc.prem)*

$\langle \neg (\exists\ xr\ j.\ prefix\ xr\ xs \wedge j \leq i \wedge vs @ xr \in RM\ M2\ M1\ \Omega\ V\ m\ j) \rangle$

$\langle vs @ take\ (Suc\ k)\ xs \notin TS\ M2\ M1\ \Omega\ V\ m\ (Suc\ k) \rangle$  *calculation take-is-prefix*

**qed**

**qed**

**then have**  $vs @ take\ (i-1)\ xs \in C\ M2\ M1\ \Omega\ V\ m\ i - RM\ M2\ M1\ \Omega\ V\ m\ i$

**using** *assms(4)*

**by** *(metis One-nat-def Suc-diff-1 Suc-leI le-less)*

**then have** *?PrefJustContained*

**by** *(metis C-subset DiffD1 assms(1) subsetCE take-is-prefix)*

**then show** *False*

**using**  $\langle \neg ?PrefJustContained \rangle$  **by** *simp*

**qed**

**show**  $\neg (?PrefPreviouslyRemoved \wedge ?PrefJustContained)$

**proof**

**assume**  $?PrefPreviouslyRemoved \wedge ?PrefJustContained$

**then have**  $?PrefPreviouslyRemoved$

$?PrefJustContained$

**by** *auto*

**obtain**  $xr\ j$  **where**  $prefix\ xr\ xs\ j \leq i$   $vs @ xr \in ?RM\ j$

**using**  $\langle ?PrefPreviouslyRemoved \rangle$  **by** *blast*

**obtain**  $xc$  **where**  $prefix\ xc\ xs\ vs @ xc \in ?C\ i - ?RM\ i$

**using**  $\langle ?PrefJustContained \rangle$  **by** *blast*

**then have**  $Suc\ (length\ xc) = i$

**using** *C-index*

**by** *(metis Diff-iff assms(2) mcp-prefix-of-suffix)*

**moreover have**  $length\ xc \leq length\ xs$

**using**  $\langle prefix\ xc\ xs \rangle$  **by** *(simp add: prefix-length-le)*

**moreover have**  $xc \neq xs$

**proof**

**assume**  $xc = xs$

**then have**  $vs @ xc \in ?C\ i$

**using**  $\langle vs @ xc \in ?C\ i - ?RM\ i \rangle$  **by** *auto*

**then have**  $vs @ xc \in ?TS\ i$

**using** *C-subset* **by** *blast*

**then show** *False*

**using** *assms(1)* **by** *blast*

**qed**

**ultimately have**  $i \leq length\ xs$

**using**  $\langle prefix\ xc\ xs \rangle$  *not-less-eq-eq prefix-length-prefix prefix-order.antisym*

**by** *blast*

```

have  $\bigwedge n . (n < i) \implies vs@(take\ n\ xs) \in ?C\ (Suc\ n)$ 
proof -
  fix n assume n < i
  show vs @ take n xs  $\in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ n)$ 
  proof -
    have n  $\leq$  length xc
      using  $\langle n < i \rangle \langle Suc\ (length\ xc) = i \rangle$  less-Suc-eq-le
      by blast
    then have prefix (vs @ (take n xs)) (vs @ xc)
    proof -
      have n  $\leq$  length xs
        using  $\langle length\ xc \leq length\ xs \rangle \langle n \leq length\ xc \rangle$  order-trans
        by blast
      then have prefix (take n xs) xc
        by (metis (no-types)  $\langle n \leq length\ xc \rangle \langle prefix\ xc\ xs \rangle$  length-take min.absorb2
            prefix-length-prefix take-is-prefix)
      then show ?thesis
        by simp
    qed
  then have vs @ take n xs  $\in ?TS\ i$ 
    by (meson C-subset DiffD1 TS-prefix-containment  $\langle prefix\ xc\ xs \rangle$ 
         $\langle vs\ @\ xc \in C\ M2\ M1\ \Omega\ V\ m\ i - RM\ M2\ M1\ \Omega\ V\ m\ i \rangle$  assms(2) contra-subsetD
        mcp-prefix-of-suffix same-prefix-prefix)
  then obtain jn where jn < Suc i vs@(take n xs)  $\in ?C\ jn$ 
    using TS-union[of M2 M1  $\Omega$  V m i]
    by (metis UN-iff atLeast-upt lessThan-iff)
  moreover have mcp (vs @ take n xs) V vs
    by (meson assms(2) mcp-prefix-of-suffix take-is-prefix)
  ultimately have jn = Suc (length (take n xs))
    using C-index[of vs take n xs M2 M1  $\Omega$  V m jn] by auto
  then have jn = Suc n
    using  $\langle length\ xc \leq length\ xs \rangle \langle n \leq length\ xc \rangle$  by auto
  then show vs@(take n xs)  $\in ?C\ (Suc\ n)$ 
    using  $\langle vs@(take\ n\ xs) \in ?C\ jn \rangle$  by auto
  qed
qed

```

```

have  $\bigwedge n . (n < i) \implies vs@(take\ n\ xs) \notin ?RM\ (Suc\ n)$ 
proof -
  fix n assume n < i
  show vs @ take n xs  $\notin RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ n)$ 
  proof (cases n = length xc)
    case True
      then show ?thesis
        using  $\langle vs@xc \in ?C\ i - ?RM\ i \rangle$ 
        by (metis DiffD2  $\langle Suc\ (length\ xc) = i \rangle \langle prefix\ xc\ xs \rangle$  append-eq-conv-conj prefixE)
    next
      case False
        then have n < length xc
          using  $\langle n < i \rangle \langle Suc\ (length\ xc) = i \rangle$  by linarith

        show ?thesis
        proof (cases Suc n < length xc)
          case True
            then have Suc n < i
              using  $\langle Suc\ (length\ xc) = i \rangle \langle n < length\ xc \rangle$  by blast
            then have vs @ (take (Suc n) xs)  $\in ?C\ (Suc\ (Suc\ n))$ 
              using  $\langle \bigwedge n . (n < i) \implies vs@(take\ n\ xs) \in ?C\ (Suc\ n) \rangle$  by blast
            then have vs @ butlast (take (Suc n) xs)  $\in ?C\ (Suc\ n) - ?RM\ (Suc\ n)$ 
              using True C-immediate-prefix-containment[of vs take (Suc n) xs M2 M1  $\Omega$  V m n]
              by (metis Suc-neq-Zero  $\langle prefix\ xc\ xs \rangle \langle xc \neq xs \rangle$  prefix-Nil take-eq-Nil)
            then show ?thesis
              by (metis DiffD2 Suc-lessD True  $\langle length\ xc \leq length\ xs \rangle$  butlast-snoc less-le-trans)
          case False
            then show ?thesis
              by (metis DiffD2 Suc-lessD True  $\langle length\ xc \leq length\ xs \rangle$  butlast-snoc less-le-trans)
        qed
      qed
  qed

```

```

      take-Suc-conv-app-nth)
next
  case False
  then have Suc n = length xc
    using Suc-lessI ⟨n < length xc⟩ by blast
  then have vs @ (take (Suc n) xs) ∈ ?C (Suc (Suc n))
    using ⟨Suc (length xc) = i⟩ ⟨∧n. n < i ⟹ vs @ take n xs ∈ C M2 M1 Ω V m (Suc n)⟩
    by auto
  then have vs @ butlast (take (Suc n) xs) ∈ ?C (Suc n) - ?RM (Suc n)
    using False C-immediate-prefix-containment[of vs take (Suc n) xs M2 M1 Ω V m n]
    by (metis Suc-neq-Zero ⟨prefix xc xs⟩ ⟨xc ≠ xs⟩ prefix-Nil take-eq-Nil)
  then show ?thesis
    by (metis Diff-iff ⟨Suc n = length xc⟩ ⟨length xc ≤ length xs⟩ butlast-take diff-Suc-1)
  qed
qed
qed

have xr = take j xs
proof -
  have vs@xr ∈ ?C j
    using ⟨vs@xr ∈ ?RM j⟩ RM-subset by blast
  then show ?thesis
    using C-index
    by (metis Suc-le-lessD ⟨∧n. n < i ⟹ vs @ take n xs ∉ RM M2 M1 Ω V m (Suc n)⟩ ⟨j ≤ i⟩
      ⟨prefix xr xs⟩ ⟨vs @ xr ∈ RM M2 M1 Ω V m j⟩ append-eq-conv-conj assms(2)
      mcp-prefix-of-suffix prefix-def)
  qed

have vs@xr ∉ ?RM j
  by (metis (no-types) C-index RM-subset ⟨i ≤ length xs⟩ ⟨j ≤ i⟩ ⟨prefix xr xs⟩
    ⟨xr = take j xs⟩ assms(2) contra-subsetD dual-order.trans length-take lessI less-irrefl
    mcp-prefix-of-suffix min.absorb2)

then show False
  using ⟨vs@xr ∈ ?RM j⟩ by simp
qed
qed

```

```

lemma TS-non-containment-causes-rev :
  assumes mcp (vs@xs) V vs
  and (∃ xr j . xr ≠ xs ∧ prefix xr xs ∧ j ≤ i ∧ vs@xr ∈ RM M2 M1 Ω V m j)
    ∨ (∃ xc . xc ≠ xs ∧ prefix xc xs ∧ vs@xc ∈ (C M2 M1 Ω V m i) - (RM M2 M1 Ω V m i))
    (is ?PrefPreviouslyRemoved ∨ ?PrefJustContained)
  shows vs@xs ∉ TS M2 M1 Ω V m i
proof
  let ?TS = λ n . TS M2 M1 Ω V m n
  let ?C = λ n . C M2 M1 Ω V m n
  let ?RM = λ n . RM M2 M1 Ω V m n

  assume vs @ xs ∈ TS M2 M1 Ω V m i

  have ?PrefPreviouslyRemoved ⟹ False
  proof -
    assume ?PrefPreviouslyRemoved
    then obtain xr j where xr ≠ xs ∧ prefix xr xs ∧ j ≤ i ∧ vs@xr ∈ ?RM j
      by blast
    then have vs@xr ∉ ?C j - ?RM j
      by blast
  qed

  have vs@(take (Suc (length xr)) xs) ∉ ?C (Suc j)

```

**proof** –

**have**  $vs@(take\ (length\ xr)\ xs) \notin ?C\ j - ?RM\ j$   
**by**  $(metis\ \langle prefix\ xr\ xs \rangle\ \langle vs\ @\ xr \notin C\ M2\ M1\ \Omega\ V\ m\ j - RM\ M2\ M1\ \Omega\ V\ m\ j \rangle$   
 $append-eq-conv-conj\ prefix-def)$

**show**  $?thesis$

**proof**  $(cases\ j)$

**case**  $0$

**then show**  $?thesis$

**using**  $RM.simps(1)\ \langle vs\ @\ xr \in RM\ M2\ M1\ \Omega\ V\ m\ j \rangle$  **by**  $blast$

**next**

**case**  $(Suc\ j')$

**then have**  $?C\ (Suc\ j) \subseteq append-set\ (?C\ j - ?RM\ j)\ (inputs\ M2)$

**using**  $C.simps(3)\ Suc$  **by**  $blast$

**obtain**  $x$  **where**  $vs@(take\ (Suc\ (length\ xr))\ xs) = vs@(take\ (length\ xr)\ xs) @ [x]$

**by**  $(metis\ \langle prefix\ xr\ xs \rangle\ \langle xr \neq xs \rangle\ append-eq-conv-conj\ not-le\ prefix-def$   
 $take-Suc-conv-app-nth\ take-all)$

**have**  $vs@(take\ (length\ xr)\ xs) @ [x] \notin append-set\ (?C\ j - ?RM\ j)\ (inputs\ M2)$

**using**  $\langle vs@(take\ (length\ xr)\ xs) \notin ?C\ j - ?RM\ j \rangle$  **by**  $simp$

**then have**  $vs@(take\ (length\ xr)\ xs) @ [x] \notin ?C\ (Suc\ j)$

**using**  $\langle ?C\ (Suc\ j) \subseteq append-set\ (?C\ j - ?RM\ j)\ (inputs\ M2) \rangle$  **by**  $blast$

**then show**  $?thesis$

**using**  $\langle vs@(take\ (Suc\ (length\ xr))\ xs) = vs@(take\ (length\ xr)\ xs) @ [x] \rangle$  **by**  $auto$

**qed**

**qed**

**have**  $prefix\ (take\ (Suc\ (length\ xr))\ xs)\ xs$

**by**  $(simp\ add:\ take-is-prefix)$

**then have**  $vs@(take\ (Suc\ (length\ xr))\ xs) \in ?TS\ i$

**using**  $TS-prefix-containment[OF\ \langle vs\ @\ xs \in TS\ M2\ M1\ \Omega\ V\ m\ i \rangle\ assms(1)]$  **by**  $simp$

**then obtain**  $j'$  **where**  $j' < Suc\ i \wedge vs@(take\ (Suc\ (length\ xr))\ xs) \in ?C\ j'$

**using**  $TS-union[of\ M2\ M1\ \Omega\ V\ m\ i]$  **by**  $fastforce$

**then have**  $Suc\ (Suc\ (length\ xr)) = j'$

**using**  $C-index[of\ vs\ take\ (Suc\ (length\ xr))\ xs]$

**proof** –

**have**  $\neg length\ xs \leq length\ xr$

**by**  $(metis\ (no-types)\ \langle prefix\ xr\ xs \rangle\ \langle xr \neq xs \rangle\ append-Nil2\ append-eq-conv-conj\ leD$   
 $nat-less-le\ prefix-def\ prefix-length-le)$

**then show**  $?thesis$

**by**  $(metis\ (no-types)\ \langle \bigwedge i\ \Omega\ V\ T\ S\ M2\ M1.\ \llbracket vs\ @\ take\ (Suc\ (length\ xr))\ xs \in C\ M2\ M1\ \Omega\ V\ m\ i;$   
 $mcp\ (vs\ @\ take\ (Suc\ (length\ xr))\ xs)\ V\ vs \rrbracket$

$\implies Suc\ (length\ (take\ (Suc\ (length\ xr))\ xs)) = i \rangle$

$\langle j' < Suc\ i \wedge vs\ @\ take\ (Suc\ (length\ xr))\ xs \in C\ M2\ M1\ \Omega\ V\ m\ j' \rangle$

$append-eq-conv-conj\ assms(1)\ length-take\ mcp-prefix-of-suffix\ min.absorb2$

$not-less-eq-eq\ prefix-def)$

**qed**

**moreover have**  $Suc\ (length\ xr) = j$

**using**  $\langle vs@xr \in ?RM\ j \rangle\ RM-subset\ C-index$

**by**  $(metis\ \langle prefix\ xr\ xs \rangle\ assms(1)\ mcp-prefix-of-suffix\ subsetCE)$

**ultimately have**  $j' = Suc\ j$

**by**  $auto$

**then have**  $vs@(take\ (Suc\ (length\ xr))\ xs) \in ?C\ (Suc\ j)$

**using**  $\langle j' < Suc\ i \wedge vs@(take\ (Suc\ (length\ xr))\ xs) \in ?C\ j' \rangle$  **by**  $auto$

**then show**  $False$

**using**  $\langle vs@(take\ (Suc\ (length\ xr))\ xs) \notin ?C\ (Suc\ j) \rangle$  **by**  $blast$

**qed**

**moreover have**  $?PrefJustContained \implies False$

**proof** –

**assume**  $?PrefJustContained$

**then obtain**  $xc$  **where**  $xc \neq xs$

$prefix\ xc\ xs$

$vs\ @\ xc \in ?C\ i - ?RM\ i$

**by**  $blast$

— only possible if  $xc = xs$   
**then show** *False*  
 by (*metis C-index DiffD1 Suc-less-eq TS-index(1) ‹vs @ xs ∈ ?TS i› assms(1) leD le-neq-trans*  
*mcp-prefix-of-suffix prefix-length-le prefix-length-prefix*  
*prefix-order.dual-order.antisym prefix-order.order-refl*)  
**qed**

**ultimately show** *False*  
 using *assms(2)* by *auto*  
**qed**

**lemma** *TS-finite* :  
 assumes *finite V*  
 and *finite (inputs M2)*  
 shows *finite (TS M2 M1 Ω V m n)*  
 using *assms* **proof** (*induction n*)  
 case 0  
 then show *?case* by *auto*  
**next**  
 case (*Suc n*)

let *?TS = λ n . TS M2 M1 Ω V m n*  
 let *?C = λ n . C M2 M1 Ω V m n*  
 let *?RM = λ n . RM M2 M1 Ω V m n*

show *?case*  
**proof** (*cases n=0*)  
 case *True*  
 then have *?TS (Suc n) = V*  
 by *auto*  
 then show *?thesis*  
 using *‹finite V›* by *auto*

**next**  
 case *False*  
 then have *?TS (Suc n) = ?TS n ∪ ?C (Suc n)*  
 by (*metis TS.simps(3) gr0-implies-Suc neq0-conv*)  
 moreover have *finite (?TS n)*  
 using *Suc.IH[OF Suc.prem]* by *assumption*  
 moreover have *finite (?C (Suc n))*  
**proof** –  
 have *?C (Suc n) ⊆ append-set (?C n) (inputs M2)*  
 using *C-step False* by *blast*  
 moreover have *?C n ⊆ ?TS n*  
 by (*simp add: C-subset*)  
 ultimately have *?C (Suc n) ⊆ append-set (?TS n) (inputs M2)*  
 by *blast*  
 moreover have *finite (append-set (?TS n) (inputs M2))*  
 by (*simp add: ‹finite (TS M2 M1 Ω V m n)› assms(2) finite-image-set2*)  
 ultimately show *?thesis*  
 using *infinite-subset* by *auto*  
**qed**  
 ultimately show *?thesis*  
 by *auto*  
**qed**  
**qed**

**lemma** *C-finite* :  
 assumes *finite V*  
 and *finite (inputs M2)*  
 shows *finite (C M2 M1 Ω V m n)*  
**proof** –  
 have *C M2 M1 Ω V m n ⊆ TS M2 M1 Ω V m n*  
 by (*simp add: C-subset*)

```

then show ?thesis using TS-finite[OF assms]
using Finite-Set.finite-subset by blast
qed

```

## 5.5 Final iteration

The result of calculating TS for some iteration is final if the result does not change for the next iteration.

Such a final iteration exists and is at most equal to the number of states of FSM M2 multiplied by an upper bound on the number of states of FSM M1.

Furthermore, for any sequence not contained in the final iteration of the test suite, a prefix of this sequence must be contained in the latter.

**abbreviation** *final-iteration*  $M2\ M1\ \Omega\ V\ m\ i \equiv TS\ M2\ M1\ \Omega\ V\ m\ i = TS\ M2\ M1\ \Omega\ V\ m\ (Suc\ i)$

**lemma** *final-iteration-ex* :

```

assumes OFSM M1
and OFSM M2
and asc-fault-domain M2 M1 m
and test-tools M2 M1 FAIL PM V  $\Omega$ 
shows final-iteration M2 M1  $\Omega$  V m (Suc ( |M2| * m ))

```

**proof** –

```

let ?i = Suc ( |M2| * m )

```

```

let ?TS =  $\lambda\ n.\ TS\ M2\ M1\ \Omega\ V\ m\ n$ 
let ?C =  $\lambda\ n.\ C\ M2\ M1\ \Omega\ V\ m\ n$ 
let ?RM =  $\lambda\ n.\ RM\ M2\ M1\ \Omega\ V\ m\ n$ 

```

```

have is-det-state-cover M2 V
using assms by auto
moreover have finite (nodes M2)
using assms(2) by auto
moreover have d-reachable M2 (initial M2)  $\subseteq$  nodes M2
by auto
ultimately have finite V
using det-state-cover-card[of M2 V]
by (metis finite-if-finite-subsets-card-bdd infinite-subset is-det-state-cover.elims(2)
surj-card-le)

```

```

have  $\forall\ seq \in ?C\ ?i.\ seq \in ?RM\ ?i$ 

```

**proof**

```

fix seq assume seq  $\in ?C\ ?i$ 
show seq  $\in ?RM\ ?i$ 
proof –

```

```

have []  $\in V$ 
using <is-det-state-cover M2 V> det-state-cover-empty
by blast
then obtain vs where mcp seq V vs
using mcp-ex[OF - <finite V>]
by blast
then obtain xs where seq = vs@xs
using prefixE by auto

```

```

then have Suc (length xs) = ?i using C-index
using <mcp seq V vs> <seq  $\in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ (|M2| * m))$ > by blast
then have length xs = ( |M2| * m ) by auto

```

```

have RM-def : ?RM ?i = {xs'  $\in C\ M2\ M1\ \Omega\ V\ m\ ?i$  .
  ( $\neg (L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\})$ )
   $\vee (\forall\ io \in L_{in}\ M1\ \{xs'\} .$ 
    ( $\exists\ V'' \in N\ io\ M1\ V .$ 
      ( $\exists\ S1 .$ 
        ( $\exists\ vs\ xs .$ 

```

$io = (vs@xs)$   
 $\wedge mcp (vs@xs) V'' vs$   
 $\wedge S1 \subseteq nodes M2$   
 $\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP M2 s1 vs xs V'' .$   
 $\forall io2 \in RP M2 s2 vs xs V'' .$   
 $B M1 io1 \Omega \neq B M1 io2 \Omega ))$   
 $\wedge m < LB M2 M1 vs xs (?TS ((|M2| * m) \cup V) S1 \Omega V'' ))\}$   
**using** *RM.simps(2)*[of *M2 M1 \Omega V m ((card (nodes M2))\*m)*] **by** *assumption*

**have**  $(\neg (L_{in} M1 \{seq\} \subseteq L_{in} M2 \{seq\}))$   
 $\vee (\forall io \in L_{in} M1 \{seq\} .$   
 $(\exists V'' \in N io M1 V .$   
 $(\exists S1 .$   
 $(\exists vs xs .$   
 $io = (vs@xs)$   
 $\wedge mcp (vs@xs) V'' vs$   
 $\wedge S1 \subseteq nodes M2$   
 $\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP M2 s1 vs xs V'' .$   
 $\forall io2 \in RP M2 s2 vs xs V'' .$   
 $B M1 io1 \Omega \neq B M1 io2 \Omega ))$   
 $\wedge m < LB M2 M1 vs xs (?TS ((|M2| * m) \cup V) S1 \Omega V'' ))\}))$

**proof** (*cases*  $(\neg (L_{in} M1 \{seq\} \subseteq L_{in} M2 \{seq\}))$ )

**case** *True*

**then show** *?thesis*

**using** *RM-def* **by** *blast*

**next**

**case** *False*

**have**  $(\forall io \in L_{in} M1 \{seq\} .$

$(\exists V'' \in N io M1 V .$

$(\exists S1 .$

$(\exists vs xs .$

$io = (vs@xs)$

$\wedge mcp (vs@xs) V'' vs$

$\wedge S1 \subseteq nodes M2$

$\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP M2 s1 vs xs V'' .$

$\forall io2 \in RP M2 s2 vs xs V'' .$

$B M1 io1 \Omega \neq B M1 io2 \Omega ))$

$\wedge m < LB M2 M1 vs xs (?TS ((|M2| * m) \cup V) S1 \Omega V'' ))\}))$

**proof**

**fix** *io* **assume**  $io \in L_{in} M1 \{seq\}$

**then have**  $io \in L M1$

**by** *auto*

**moreover have** *is-det-state-cover M2 V*

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

**ultimately obtain**  $V''$  **where**  $V'' \in N io M1 V$

**using** *N-nonempty[OF - assms(1-3), of V io]* **by** *blast*

**have**  $io \in L M2$

**using**  $\langle io \in L_{in} M1 \{seq\} \rangle$  *False* **by** *auto*

**have**  $V'' \in Perm V M1$

**using**  $\langle V'' \in N io M1 V \rangle$  **by** *auto*

**have**  $\square \in V''$

**using**  $\langle V'' \in Perm V M1 \rangle$  *assms(4)* *perm-empty* **by** *blast*

**have** *finite V''*

**using**  $\langle V'' \in Perm V M1 \rangle$  *assms(2)* *assms(4)* *perm-elem-finite* **by** *blast*

**obtain** *vs* **where** *mcp io V'' vs*

**using**  $mcp\text{-ex}[OF \langle [] \in V'' \rangle \langle finite\ V'' \rangle]$  **by** *blast*

**obtain**  $xs$  **where**  $io = (vs @ xs)$   
**using**  $\langle mcp\ io\ V''\ vs \rangle$  *prefixE* **by** *auto*

**then have**  $vs @ xs \in L\ M1\ vs @ xs \in L\ M2$   
**using**  $\langle io \in L\ M1 \rangle \langle io \in L\ M2 \rangle$  **by** *auto*

**have**  $io \in L\ M1\ map\ fst\ io \in \{seq\}$   
**using**  $\langle io \in L_{in}\ M1\ \{seq\} \rangle$  **by** *auto*  
**then have**  $map\ fst\ io = seq$   
**by** *auto*

**then have**  $map\ fst\ io \in ?C\ ?i$   
**using**  $\langle seq \in ?C\ ?i \rangle$  **by** *blast*

**then have**  $(map\ fst\ vs) @ (map\ fst\ xs) \in ?C\ ?i$   
**using**  $\langle io = (vs @ xs) \rangle$  **by** *(metis map-append)*

**have**  $mcp'\ io\ V'' = vs$   
**using**  $\langle mcp\ io\ V''\ vs \rangle$  *mcp'-intro* **by** *blast*

**have**  $mcp'\ (map\ fst\ io)\ V = (map\ fst\ vs)$   
**using**  $\langle V'' \in N\ io\ M1\ V \rangle \langle mcp'\ io\ V'' = vs \rangle$  **by** *auto*

**then have**  $mcp\ (map\ fst\ io)\ V\ (map\ fst\ vs)$   
**by** *(metis \langle \bigwedge thesis. (\bigwedge vs. mcp\ seq\ V\ vs \implies thesis) \implies thesis \rangle \langle map\ fst\ io = seq \rangle mcp'-intro)*

**then have**  $mcp\ (map\ fst\ vs @ map\ fst\ xs)\ V\ (map\ fst\ vs)$   
**by** *(simp add: \langle io = vs @ xs \rangle)*

**then have**  $Suc\ (length\ xs) = ?i$  **using**  $C\text{-index}[OF \langle (map\ fst\ vs) @ (map\ fst\ xs) \in ?C\ ?i \rangle]$   
**by** *simp*

**then have**  $(|M2| * m) \leq length\ xs$   
**by** *simp*

**have**  $|M1| \leq m$   
**using** *assms(3)* **by** *auto*

**have**  $vs @ xs \in L\ M2 \cap L\ M1$   
**using**  $\langle vs @ xs \in L\ M1 \rangle \langle vs @ xs \in L\ M2 \rangle$  **by** *blast*

**obtain**  $q$  **where**  $q \in nodes\ M2\ m < card\ (RP\ M2\ q\ vs\ xs\ V'')$   
**using** *RP-state-repetition-distribution-productF*  
 $[OF\ assms(2,1) \langle (|M2| * m) \leq length\ xs \rangle \langle |M1| \leq m \rangle \langle vs @ xs \in L\ M2 \cap L\ M1 \rangle$   
 $\langle is\ det\ state\ cover\ M2\ V \rangle \langle V'' \in Perm\ V\ M1 \rangle]$   
**by** *blast*

**have**  $m < LB\ M2\ M1\ vs\ xs\ (?TS\ ((|M2| * m) \cup V)\ \{q\}\ \Omega\ V'')$   
**proof** –

**have**  $m < (sum\ (\lambda\ s.\ card\ (RP\ M2\ s\ vs\ xs\ V''))\ \{q\})$   
**using**  $\langle m < card\ (RP\ M2\ q\ vs\ xs\ V'') \rangle$   
**by** *auto*

**moreover have**  $(sum\ (\lambda\ s.\ card\ (RP\ M2\ s\ vs\ xs\ V''))\ \{q\})$   
 $\leq LB\ M2\ M1\ vs\ xs\ (?TS\ ((|M2| * m) \cup V)\ \{q\}\ \Omega\ V'')$   
**by** *auto*

**ultimately show** *?thesis*  
**by** *linarith*

**qed**

**show**  $\exists V'' \in N\ io\ M1\ V.$   
 $\exists S1\ vs\ xs.$   
 $io = vs @ xs \wedge$   
 $mcp\ (vs @ xs)\ V''\ vs \wedge$



**then show** *?thesis*  
**using** *RM-def* **by** *blast*  
**qed**  
**qed**

**then have**  $?C \ ?i - ?RM \ ?i = \{\}$   
**by** *blast*

**have**  $?C \ (Suc \ ?i) = append-set \ (?C \ ?i - ?RM \ ?i) \ (inputs \ M2) - ?TS \ ?i$   
**using** *C.simps(3)* **by** *blast*

**then have**  $?C \ (Suc \ ?i) = \{\}$  **using**  $\langle ?C \ ?i - ?RM \ ?i = \{\} \rangle$   
**by** *blast*

**then have**  $?TS \ (Suc \ ?i) = ?TS \ ?i$   
**using** *TS.simps(3)* **by** *blast*

**then show** *final-iteration*  $M2 \ M1 \ \Omega \ V \ m \ ?i$   
**by** *blast*

**qed**

**lemma** *TS-non-containment-causes-final* :

**assumes**  $vs@xs \notin TS \ M2 \ M1 \ \Omega \ V \ m \ i$

**and**  $mcp \ (vs@xs) \ V \ vs$

**and**  $set \ xs \subseteq inputs \ M2$

**and** *final-iteration*  $M2 \ M1 \ \Omega \ V \ m \ i$

**and** *OFSM*  $M2$

**shows**  $(\exists \ xr \ j . xr \neq xs$   
 $\quad \wedge \ prefix \ xr \ xs$   
 $\quad \wedge \ j \leq i$   
 $\quad \wedge \ vs@xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j)$

**proof** –

**let**  $?TS = \lambda \ n . TS \ M2 \ M1 \ \Omega \ V \ m \ n$

**let**  $?C = \lambda \ n . C \ M2 \ M1 \ \Omega \ V \ m \ n$

**let**  $?RM = \lambda \ n . RM \ M2 \ M1 \ \Omega \ V \ m \ n$

**have**  $\{\} \neq V$

**using** *assms(2)* **by** *fastforce*

**then have**  $?TS \ 0 \neq ?TS \ (Suc \ 0)$

**by** *simp*

**then have**  $0 < i$

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

**have**  $ncc1 : (\exists \ xr \ j . xr \neq xs \wedge prefix \ xr \ xs \wedge j \leq i \wedge vs @ xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j) \vee$   
 $(\exists \ xc . xc \neq xs \wedge prefix \ xc \ xs \wedge vs @ xc \in C \ M2 \ M1 \ \Omega \ V \ m \ i - RM \ M2 \ M1 \ \Omega \ V \ m \ i)$

**using** *TS-non-containment-causes(1)*[*OF* *assms(1-3)*]  $\langle 0 < i \rangle$  **by** *assumption*

**have**  $ncc2 : \neg ((\exists \ xr \ j . xr \neq xs \wedge prefix \ xr \ xs \wedge j \leq i \wedge vs @ xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j) \wedge$

$(\exists \ xc . xc \neq xs \wedge prefix \ xc \ xs \wedge vs @ xc \in C \ M2 \ M1 \ \Omega \ V \ m \ i - RM \ M2 \ M1 \ \Omega \ V \ m \ i))$

**using** *TS-non-containment-causes(2)*[*OF* *assms(1-3)*]  $\langle 0 < i \rangle$  **by** *assumption*

**from** *ncc1* **show** *?thesis*

**proof**

**show**  $\exists \ xr \ j . xr \neq xs \wedge prefix \ xr \ xs \wedge j \leq i \wedge vs @ xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j \implies$

$\exists \ xr \ j . xr \neq xs \wedge prefix \ xr \ xs \wedge j \leq i \wedge vs @ xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j$

**by** *simp*

**show**  $\exists \ xc . xc \neq xs \wedge prefix \ xc \ xs \wedge vs @ xc \in C \ M2 \ M1 \ \Omega \ V \ m \ i - RM \ M2 \ M1 \ \Omega \ V \ m \ i \implies$

$\exists \ xr \ j . xr \neq xs \wedge prefix \ xr \ xs \wedge j \leq i \wedge vs @ xr \in RM \ M2 \ M1 \ \Omega \ V \ m \ j$

**proof** –

**assume**  $\exists \ xc . xc \neq xs \wedge prefix \ xc \ xs \wedge vs @ xc \in C \ M2 \ M1 \ \Omega \ V \ m \ i - RM \ M2 \ M1 \ \Omega \ V \ m \ i$

**then obtain**  $xc$  **where**  $xc \neq xs \wedge prefix \ xc \ xs \wedge vs @ xc \in ?C \ i - ?RM \ i$

**by** *blast*

```

then have  $vs @ xc \in ?C i$ 
  by blast
have  $mcp (vs @ xc) V vs$ 
  using  $\langle prefix\ xc\ xs \rangle\ assms(2)\ mcp\ prefix\ of\ suffix$  by blast
then have  $Suc (length\ xc) = i$  using  $C\ index[OF\ \langle vs @ xc \in ?C i \rangle]$ 
  by simp

have  $length\ xc < length\ xs$ 
  by  $(metis\ \langle prefix\ xc\ xs \rangle\ \langle xc \neq xs \rangle\ append\ eq\ conv\ conj\ nat\ less\ le\ prefix\ def\ prefix\ length\ le\ take\ all)$ 
then obtain  $x$  where  $prefix\ (vs @ xc @ [x])\ (vs @ xs)$ 
  using  $\langle prefix\ xc\ xs \rangle\ append\ one\ prefix\ same\ prefix\ prefix$  by blast

```

— Proof sketch:  $vs\ xs\ x$  must not be in TS (i+1), else not final iteration  $vs\ xs\ x$  can not be in TS i due to its length  $vs\ xs\ x$  must therefore not be contained in (append-set (C i - R i) (inputs M2))  $vs\ xs$  must therefore not be contained in (C i - R i) contradiction

```

have  $?TS (Suc\ i) = ?TS\ i$ 
  using  $assms(4)$  by auto

```

```

have  $vs @ xc @ [x] \notin ?C (Suc\ i)$ 

```

**proof**

```

  assume  $vs @ xc @ [x] \in ?C (Suc\ i)$ 
  then have  $vs @ xc @ [x] \notin ?TS\ i$ 
    by  $(metis\ (no\ types,\ lifting)\ C.\ simps(3)\ DiffE\ \langle Suc\ (length\ xc) = i \rangle)$ 
  then have  $?TS\ i \neq ?TS (Suc\ i)$ 
    using  $C\ subset\ \langle vs @ xc @ [x] \in C\ M2\ M1\ \Omega\ V\ m (Suc\ i) \rangle$  by blast
  then show False using  $assms(4)$ 
    by auto

```

**qed**

```

moreover have  $?C (Suc\ i) = append\ set\ (?C\ i - ?RM\ i)\ (inputs\ M2) - ?TS\ i$ 

```

```

  using  $C.\ simps(3)\ \langle Suc\ (length\ xc) = i \rangle$  by blast

```

```

ultimately have  $vs @ xc @ [x] \notin append\ set\ (?C\ i - ?RM\ i)\ (inputs\ M2) - ?TS\ i$ 
  by blast

```

```

have  $vs @ xc @ [x] \notin ?TS (Suc\ i)$ 

```

```

  by  $(metis\ Suc\ n\ not\ le\ n\ TS\ index(1)\ \langle Suc\ (length\ xc) = i \rangle\ \langle prefix\ (vs @ xc @ [x])\ (vs @ xs) \rangle\ assms(2)\ assms(4)\ length\ append\ singleton\ mcp\ prefix\ of\ suffix\ same\ prefix\ prefix)$ 

```

```

then have  $vs @ xc @ [x] \notin ?TS\ i$ 

```

```

  by  $(simp\ add:\ assms(4))$ 

```

```

have  $vs @ xc @ [x] \notin append\ set\ (?C\ i - ?RM\ i)\ (inputs\ M2)$ 

```

```

  using  $\langle vs @ xc @ [x] \notin TS\ M2\ M1\ \Omega\ V\ m\ i \rangle$ 

```

```

     $\langle vs @ xc @ [x] \notin append\ set\ (C\ M2\ M1\ \Omega\ V\ m\ i - RM\ M2\ M1\ \Omega\ V\ m\ i)\ (inputs\ M2) - TS\ M2\ M1\ \Omega\ V\ m\ i \rangle$ 

```

```

  by blast

```

```

then have  $vs @ xc \notin (?C\ i - ?RM\ i)$ 

```

**proof** —

```

  have  $f1: \forall A\ Aa.\ (a::'a) \notin A \wedge a \notin Aa \vee a \in Aa \cup A$ 

```

```

    by  $(meson\ UnCI)$ 

```

```

  obtain  $aas :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$  where

```

```

     $\forall x0\ x1.\ (\exists v2.\ x0 = x1 @ v2) = (x0 = x1 @ aas\ x0\ x1)$ 

```

```

    by moura

```

```

  then have  $vs @ xs = (vs @ xc @ [x]) @ aas\ (vs @ xs)\ (vs @ xc @ [x])$ 

```

```

    by  $(meson\ \langle prefix\ (vs @ xc @ [x])\ (vs @ xs) \rangle\ prefixE)$ 

```

```

  then have  $xs = (xc @ [x]) @ aas\ (vs @ xs)\ (vs @ xc @ [x])$ 

```

```

    by simp

```

```

  then have  $x \in inputs\ M2$ 

```

```

    using  $f1$  by  $(metis\ (no\ types)\ assms(3)\ contra\ subsetD\ insert\ iff\ list.\ set(2)\ set\ append)$ 

```

```

  then show ?thesis

```

```

    using  $\langle vs @ xc @ [x] \notin append\ set\ (C\ M2\ M1\ \Omega\ V\ m\ i - RM\ M2\ M1\ \Omega\ V\ m\ i)\ (inputs\ M2) \rangle$ 

```

```

    by force

```

**qed**

```

    then have False
      using  $\langle vs @ xc \in ?C i - ?RM i \rangle$  by blast
    then show ?thesis by simp
  qed
qed
qed

```

**lemma** *TS-non-containment-causes-final-suc* :

```

  assumes  $vs@xs \notin TS M2 M1 \Omega V m i$ 
  and  $mcp (vs@xs) V vs$ 
  and  $set xs \subseteq inputs M2$ 
  and  $final-iteration M2 M1 \Omega V m i$ 
  and OFSM M2

```

**obtains**  $xr j$

**where**  $xr \neq xs \wedge prefix xr xs \wedge Suc j \leq i \wedge vs@xr \in RM M2 M1 \Omega V m (Suc j)$

**proof** –

```

  obtain  $xr j$  where  $xr \neq xs \wedge prefix xr xs \wedge j \leq i \wedge vs@xr \in RM M2 M1 \Omega V m j$ 
    using TS-non-containment-causes-final[OF assms] by blast

```

**moreover have**  $RM M2 M1 \Omega V m 0 = \{\}$

by *auto*

**ultimately have**  $j \neq 0$

by *(metis empty-iff)*

**then obtain**  $jp$  **where**  $j = Suc jp$

using *not0-implies-Suc* by *blast*

**then have**  $xr \neq xs \wedge prefix xr xs \wedge Suc jp \leq i \wedge vs@xr \in RM M2 M1 \Omega V m (Suc jp)$

using  $\langle xr \neq xs \wedge prefix xr xs \wedge j \leq i \wedge vs@xr \in RM M2 M1 \Omega V m j \rangle$

by *blast*

**then show** *?thesis*

using *that* by *blast*

qed

**end**

**theory** *ASC-Sufficiency*

**imports** *ASC-Suite*

**begin**

## 6 Sufficiency of the test suite to test for reduction

This section provides a proof that the test suite generated by the adaptive state counting algorithm is sufficient to test for reduction.

### 6.1 Properties of minimal sequences to failures extending the deterministic state cover

The following two lemmata show that minimal sequences to failures extending the deterministic state cover do not with their extending suffix visit any state twice or visit a state also reached by a sequence in the chosen permutation of reactions to the deterministic state cover.

**lemma** *minimal-sequence-to-failure-extending-implies-Rep-Pre* :

**assumes** *minimal-sequence-to-failure-extending*  $V M1 M2 vs xs$

**and** *OFSM M1*

**and** *OFSM M2*

**and** *test-tools*  $M2 M1 FAIL PM V \Omega$

**and**  $V'' \in N (vs@xs') M1 V$

**and**  $prefix xs' xs$

**shows**  $\neg Rep-Pre M2 M1 vs xs'$

**proof**

**assume** *Rep-Pre*  $M2 M1 vs xs'$

**then obtain**  $xs1 xs2 s1 s2$  **where**  $prefix xs1 xs2$

$prefix xs2 xs'$

$xs1 \neq xs2$

$io\text{-targets } M2 \text{ (initial } M2) (vs @ xs1) = \{s2\}$   
 $io\text{-targets } M2 \text{ (initial } M2) (vs @ xs2) = \{s2\}$   
 $io\text{-targets } M1 \text{ (initial } M1) (vs @ xs1) = \{s1\}$   
 $io\text{-targets } M1 \text{ (initial } M1) (vs @ xs2) = \{s1\}$

**by auto**  
**then have**  $s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs1)$   
 $s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs2)$   
 $s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs1)$   
 $s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs2)$   
**by auto**  
**have**  $vs@xs1 \in L M1$   
**using**  $io\text{-target-implies-}L[OF \langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs1) \rangle]$  **by assumption**  
**have**  $vs@xs2 \in L M1$   
**using**  $io\text{-target-implies-}L[OF \langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs2) \rangle]$  **by assumption**  
**have**  $vs@xs1 \in L M2$   
**using**  $io\text{-target-implies-}L[OF \langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs1) \rangle]$  **by assumption**  
**have**  $vs@xs2 \in L M2$   
**using**  $io\text{-target-implies-}L[OF \langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs2) \rangle]$  **by assumption**

**obtain**  $tr1-1$  **where**  $path M1 (vs@xs1 || tr1-1) \text{ (initial } M1)$   
 $length tr1-1 = length (vs@xs1)$   
 $target (vs@xs1 || tr1-1) \text{ (initial } M1) = s1$   
**using**  $\langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs1) \rangle$  **by auto**  
**obtain**  $tr1-2$  **where**  $path M1 (vs@xs2 || tr1-2) \text{ (initial } M1)$   
 $length tr1-2 = length (vs@xs2)$   
 $target (vs@xs2 || tr1-2) \text{ (initial } M1) = s1$   
**using**  $\langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs2) \rangle$  **by auto**  
**obtain**  $tr2-1$  **where**  $path M2 (vs@xs1 || tr2-1) \text{ (initial } M2)$   
 $length tr2-1 = length (vs@xs1)$   
 $target (vs@xs1 || tr2-1) \text{ (initial } M2) = s2$   
**using**  $\langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs1) \rangle$  **by auto**  
**obtain**  $tr2-2$  **where**  $path M2 (vs@xs2 || tr2-2) \text{ (initial } M2)$   
 $length tr2-2 = length (vs@xs2)$   
 $target (vs@xs2 || tr2-2) \text{ (initial } M2) = s2$   
**using**  $\langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs2) \rangle$  **by auto**

**have**  $productF M2 M1 FAIL PM$   
**using**  $assms(4)$  **by auto**  
**have**  $well\text{-formed } M1$   
**using**  $assms(2)$  **by auto**  
**have**  $well\text{-formed } M2$   
**using**  $assms(3)$  **by auto**  
**have**  $observable PM$   
**by**  $(meson assms(2) assms(3) assms(4) observable\text{-product}F)$

**have**  $length (vs@xs1) = length tr2-1$   
**using**  $\langle length tr2-1 = length (vs @ xs1) \rangle$  **by presburger**  
**then have**  $length tr2-1 = length tr1-1$   
**using**  $\langle length tr1-1 = length (vs@xs1) \rangle$  **by presburger**

**have**  $vs@xs1 \in L PM$   
**using**  $productF\text{-path-inclusion}[OF \langle length (vs@xs1) = length tr2-1 \rangle \langle length tr2-1 = length tr1-1 \rangle$   
 $\langle productF M2 M1 FAIL PM \rangle \langle well\text{-formed } M2 \rangle \langle well\text{-formed } M1 \rangle]$   
**by**  $(meson Int\text{-iff} \langle productF M2 M1 FAIL PM \rangle \langle vs @ xs1 \in L M1 \rangle \langle vs @ xs1 \in L M2 \rangle \langle well\text{-formed } M1 \rangle$   
 $\langle well\text{-formed } M2 \rangle productF\text{-language})$

**have**  $length (vs@xs2) = length tr2-2$   
**using**  $\langle length tr2-2 = length (vs @ xs2) \rangle$  **by presburger**  
**then have**  $length tr2-2 = length tr1-2$   
**using**  $\langle length tr1-2 = length (vs@xs2) \rangle$  **by presburger**

**have**  $vs@xs2 \in L PM$   
**using**  $productF\text{-path-inclusion}[OF \langle length (vs@xs2) = length tr2-2 \rangle \langle length tr2-2 = length tr1-2 \rangle$

$\langle \text{productF } M2 \ M1 \ \text{FAIL } PM \rangle \langle \text{well-formed } M2 \rangle \langle \text{well-formed } M1 \rangle$

**by** (*meson Int-iff*  $\langle \text{productF } M2 \ M1 \ \text{FAIL } PM \rangle \langle vs @ xs2 \in L \ M1 \rangle \langle vs @ xs2 \in L \ M2 \rangle \langle \text{well-formed } M1 \rangle$   
 $\langle \text{well-formed } M2 \rangle \text{productF-language}$ )

**have** *io-targets*  $PM$  (*initial*  $M2$ , *initial*  $M1$ ) ( $vs @ xs1$ ) =  $\{(s2, s1)\}$   
**using** *productF-path-io-targets-reverse*  
 $[OF \langle \text{productF } M2 \ M1 \ \text{FAIL } PM \rangle \langle s2 \in \text{io-targets } M2 \ (\text{initial } M2) \ (vs @ xs1) \rangle$   
 $\langle s1 \in \text{io-targets } M1 \ (\text{initial } M1) \ (vs @ xs1) \rangle \langle vs @ xs1 \in L \ M2 \rangle \langle vs @ xs1 \in L \ M1 \rangle ]$

**proof** –

**have**  $\forall c f. c \neq \text{initial} (f::('a, 'b, 'c) \text{FSM}) \vee c \in \text{nodes } f$   
**by** *blast*  
**then show** *?thesis*  
**by** (*metis* (*no-types*)  $\langle \llbracket \text{observable } M2; \text{observable } M1; \text{well-formed } M2; \text{well-formed } M1; \text{initial } M2 \in \text{nodes } M2; \text{initial } M1 \in \text{nodes } M1 \rrbracket$   
 $\implies \text{io-targets } PM \ (\text{initial } M2, \ \text{initial } M1) \ (vs @ xs1) = \{(s2, s1)\}$   
 $\text{assms}(2) \ \text{assms}(3)$ )

**qed**

**have** *io-targets*  $PM$  (*initial*  $M2$ , *initial*  $M1$ ) ( $vs @ xs2$ ) =  $\{(s2, s1)\}$   
**using** *productF-path-io-targets-reverse*  
 $[OF \langle \text{productF } M2 \ M1 \ \text{FAIL } PM \rangle \langle s2 \in \text{io-targets } M2 \ (\text{initial } M2) \ (vs @ xs2) \rangle$   
 $\langle s1 \in \text{io-targets } M1 \ (\text{initial } M1) \ (vs @ xs2) \rangle \langle vs @ xs2 \in L \ M2 \rangle \langle vs @ xs2 \in L \ M1 \rangle ]$

**proof** –

**have**  $\forall c f. c \neq \text{initial} (f::('a, 'b, 'c) \text{FSM}) \vee c \in \text{nodes } f$   
**by** *blast*  
**then show** *?thesis*  
**by** (*metis* (*no-types*)  $\langle \llbracket \text{observable } M2; \text{observable } M1; \text{well-formed } M2; \text{well-formed } M1; \text{initial } M2 \in \text{nodes } M2; \text{initial } M1 \in \text{nodes } M1 \rrbracket$   
 $\implies \text{io-targets } PM \ (\text{initial } M2, \ \text{initial } M1) \ (vs @ xs2) = \{(s2, s1)\}$   
 $\text{assms}(2) \ \text{assms}(3)$ )

**qed**

**have** *prefix* ( $vs @ xs1$ ) ( $vs @ xs2$ )  
**using**  $\langle \text{prefix } xs1 \ xs2 \rangle$  **by** *auto*

**have** *sequence-to-failure*  $M1 \ M2$  ( $vs@xs$ )  
**using**  $\text{assms}(1)$  **by** *auto*

**have** *prefix* ( $vs@xs1$ ) ( $vs@xs'$ )  
**using**  $\langle \text{prefix } xs1 \ xs2 \rangle \langle \text{prefix } xs2 \ xs' \rangle$  *prefix-order.dual-order.trans same-prefix-prefix*  
**by** *blast*  
**have** *prefix* ( $vs@xs2$ ) ( $vs@xs'$ )  
**using**  $\langle \text{prefix } xs2 \ xs' \rangle$  *prefix-order.dual-order.trans same-prefix-prefix* **by** *blast*

**have** *io-targets*  $PM$  (*initial*  $PM$ ) ( $vs @ xs1$ ) =  $\{(s2, s1)\}$   
**using**  $\langle \text{io-targets } PM \ (\text{initial } M2, \ \text{initial } M1) \ (vs @ xs1) = \{(s2, s1)\} \rangle$   $\text{assms}(4)$  **by** *auto*

**have** *io-targets*  $PM$  (*initial*  $PM$ ) ( $vs @ xs2$ ) =  $\{(s2, s1)\}$   
**using**  $\langle \text{io-targets } PM \ (\text{initial } M2, \ \text{initial } M1) \ (vs @ xs2) = \{(s2, s1)\} \rangle$   $\text{assms}(4)$  **by** *auto*

**have** ( $vs @ xs2$ )  $@$  (*drop* (*length*  $xs2$ )  $xs$ ) =  $vs@xs$   
**by** (*metis*  $\langle \text{prefix } xs2 \ xs' \rangle$  *append-eq-appendI append-eq-conv-conj*  $\text{assms}(6)$  *prefixE*)  
**moreover have** *io-targets*  $PM$  (*initial*  $PM$ ) ( $vs@xs$ ) =  $\{\text{FAIL}\}$   
**using** *sequence-to-failure-reaches-FAIL-ob*  $[OF \langle \text{sequence-to-failure } M1 \ M2 \ (vs@xs) \rangle \text{assms}(2,3)$   
 $\langle \text{productF } M2 \ M1 \ \text{FAIL } PM \rangle]$

**by** *assumption*

**ultimately have** *io-targets*  $PM$  (*initial*  $PM$ ) ( $(vs @ xs2) @ (\text{drop} (\text{length } xs2) \ xs)$ ) =  $\{\text{FAIL}\}$   
**by** *auto*

```

have io-targets PM (s2,s1) (drop (length xs2) xs) = {FAIL}
  using observable-io-targets-split
    [OF <observable PM>
      <io-targets PM (initial PM) ((vs @ xs2) @ (drop (length xs2) xs)) = {FAIL}>
      <io-targets PM (initial PM) (vs @ xs2) = {(s2, s1)}>]
  by assumption

have io-targets PM (initial PM) (vs@xs1@(drop (length xs2) xs)) = {FAIL}
  using observable-io-targets-append
    [OF <observable PM> <io-targets PM (initial PM) (vs @ xs1) = {(s2,s1)}>
      <io-targets PM (s2,s1) (drop (length xs2) xs) = {FAIL}>]
  by simp
have sequence-to-failure M1 M2 (vs@xs1@(drop (length xs2) xs))
  using sequence-to-failure-alt-def
    [OF <io-targets PM (initial PM) (vs@xs1@(drop (length xs2) xs)) = {FAIL}>]
    assms(4)
  by blast

have length xs1 < length xs2
  using <prefix xs1 xs2> <xs1 ≠ xs2> prefix-length-prefix by fastforce

have prefix-drop: ys = ys1 @ (drop (length ys1)) ys if prefix ys1 ys
  for ys ys1 :: ('a × 'b) list
  using that by (induction ys1) (auto elim: prefixE)
then have xs = (xs1 @ (drop (length xs1) xs))
  using <prefix xs1 xs2> <prefix xs2 xs'> <prefix xs' xs> by simp
then have length xs1 < length xs
  using prefix-drop[OF <prefix xs2 xs'>] <prefix xs2 xs'> <prefix xs' xs>
  using <length xs1 < length xs2>
  by (auto dest!: prefix-length-le)
have length (xs1@(drop (length xs2) xs)) < length xs
  using <length xs1 < length xs2> <length xs1 < length xs> by auto

have vs ∈ Lin M1 V
  ∧ sequence-to-failure M1 M2 (vs @ xs1@(drop (length xs2) xs))
  ∧ length (xs1@(drop (length xs2) xs)) < length xs
  using <length (xs1 @ drop (length xs2) xs) < length xs>
    <sequence-to-failure M1 M2 (vs @ xs1 @ drop (length xs2) xs)>
    assms(1) minimal-sequence-to-failure-extending.simps
  by blast

then have ¬ minimal-sequence-to-failure-extending V M1 M2 vs xs
  by (meson minimal-sequence-to-failure-extending.elims(2))

then show False
  using assms(1) by linarith
qed

lemma minimal-sequence-to-failure-extending-implies-Rep-Cov :
  assumes minimal-sequence-to-failure-extending V M1 M2 vs xs
  and OFSM M1
  and OFSM M2
  and test-tools M2 M1 FAIL PM V Ω
  and V'' ∈ N (vs@xsR) M1 V
  and prefix xsR xs
shows ¬ Rep-Cov M2 M1 V'' vs xsR
proof
  assume Rep-Cov M2 M1 V'' vs xsR
  then obtain xs' vs' s2 s1 where xs' ≠ []
    prefix xs' xsR

```

$vs' \in V''$   
 $io\text{-targets } M2 \text{ (initial } M2) (vs @ xs') = \{s2\}$   
 $io\text{-targets } M2 \text{ (initial } M2) (vs') = \{s2\}$   
 $io\text{-targets } M1 \text{ (initial } M1) (vs @ xs') = \{s1\}$   
 $io\text{-targets } M1 \text{ (initial } M1) (vs') = \{s1\}$

**by auto**

**then have**  $s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs')$   
 $s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs')$   
 $s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs')$   
 $s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs')$

**by auto**

**have**  $vs@xs' \in L M1$   
**using**  $io\text{-target-implies-}L[OF \langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs') \rangle]$  **by assumption**  
**have**  $vs' \in L M1$   
**using**  $io\text{-target-implies-}L[OF \langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs') \rangle]$  **by assumption**  
**have**  $vs@xs' \in L M2$   
**using**  $io\text{-target-implies-}L[OF \langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs') \rangle]$  **by assumption**  
**have**  $vs' \in L M2$   
**using**  $io\text{-target-implies-}L[OF \langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs') \rangle]$  **by assumption**

**obtain**  $tr1-1$  **where**  $path M1 (vs@xs' || tr1-1) \text{ (initial } M1)$   
 $length tr1-1 = length (vs@xs')$   
 $target (vs@xs' || tr1-1) \text{ (initial } M1) = s1$   
**using**  $\langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs @ xs') \rangle$  **by auto**  
**obtain**  $tr1-2$  **where**  $path M1 (vs' || tr1-2) \text{ (initial } M1)$   
 $length tr1-2 = length (vs')$   
 $target (vs' || tr1-2) \text{ (initial } M1) = s1$   
**using**  $\langle s1 \in io\text{-targets } M1 \text{ (initial } M1) (vs') \rangle$  **by auto**  
**obtain**  $tr2-1$  **where**  $path M2 (vs@xs' || tr2-1) \text{ (initial } M2)$   
 $length tr2-1 = length (vs@xs')$   
 $target (vs@xs' || tr2-1) \text{ (initial } M2) = s2$   
**using**  $\langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs @ xs') \rangle$  **by auto**  
**obtain**  $tr2-2$  **where**  $path M2 (vs' || tr2-2) \text{ (initial } M2)$   
 $length tr2-2 = length (vs')$   
 $target (vs' || tr2-2) \text{ (initial } M2) = s2$   
**using**  $\langle s2 \in io\text{-targets } M2 \text{ (initial } M2) (vs') \rangle$  **by auto**

**have**  $productF M2 M1 FAIL PM$   
**using**  $assms(4)$  **by auto**  
**have**  $well\text{-formed } M1$   
**using**  $assms(2)$  **by auto**  
**have**  $well\text{-formed } M2$   
**using**  $assms(3)$  **by auto**  
**have**  $observable PM$   
**by**  $(meson assms(2) assms(3) assms(4) observable\text{-product}F)$

**have**  $length (vs@xs') = length tr2-1$   
**using**  $\langle length tr2-1 = length (vs @ xs') \rangle$  **by presburger**  
**then have**  $length tr2-1 = length tr1-1$   
**using**  $\langle length tr1-1 = length (vs@xs') \rangle$  **by presburger**

**have**  $vs@xs' \in L PM$   
**using**  $productF\text{-path-inclusion}[OF \langle length (vs@xs') = length tr2-1 \rangle \langle length tr2-1 = length tr1-1 \rangle$   
 $\langle productF M2 M1 FAIL PM \rangle \langle well\text{-formed } M2 \rangle \langle well\text{-formed } M1 \rangle]$   
**by**  $(meson Int\text{-iff } \langle productF M2 M1 FAIL PM \rangle \langle vs @ xs' \in L M1 \rangle \langle vs @ xs' \in L M2 \rangle \langle well\text{-formed } M1 \rangle$   
 $\langle well\text{-formed } M2 \rangle productF\text{-language})$

**have**  $length (vs') = length tr2-2$   
**using**  $\langle length tr2-2 = length (vs') \rangle$  **by presburger**  
**then have**  $length tr2-2 = length tr1-2$   
**using**  $\langle length tr1-2 = length (vs') \rangle$  **by presburger**

**have**  $vs' \in L PM$   
**using**  $productF\text{-path-inclusion}[OF \langle length (vs') = length tr2-2 \rangle \langle length tr2-2 = length tr1-2 \rangle$   
 $\langle productF M2 M1 FAIL PM \rangle \langle well\text{-formed } M2 \rangle \langle well\text{-formed } M1 \rangle]$   
**by** ( $meson Int\text{-iff} \langle productF M2 M1 FAIL PM \rangle \langle vs' \in L M1 \rangle \langle vs' \in L M2 \rangle \langle well\text{-formed } M1 \rangle$   
 $\langle well\text{-formed } M2 \rangle productF\text{-language}$ )

**have**  $io\text{-targets } PM (initial M2, initial M1) (vs @ xs') = \{(s2, s1)\}$   
**using**  $productF\text{-path-io-targets-reverse}$   
 $[OF \langle productF M2 M1 FAIL PM \rangle \langle s2 \in io\text{-targets } M2 (initial M2) (vs @ xs') \rangle$   
 $\langle s1 \in io\text{-targets } M1 (initial M1) (vs @ xs') \rangle \langle vs @ xs' \in L M2 \rangle \langle vs @ xs' \in L M1 \rangle ]$

**proof** –  
**have**  $\forall c f. c \neq initial (f::('a, 'b, 'c) FSM) \vee c \in nodes f$   
**by**  $blast$   
**then show**  $?thesis$   
**by** ( $metis (no-types) \langle [observable M2; observable M1; well\text{-formed } M2; well\text{-formed } M1;$   
 $initial M2 \in nodes M2; initial M1 \in nodes M1] \rangle$   
 $\implies io\text{-targets } PM (initial M2, initial M1) (vs @ xs') = \{(s2, s1)\}$   
 $assms(2) assms(3)$ )

**qed**

**have**  $io\text{-targets } PM (initial M2, initial M1) (vs') = \{(s2, s1)\}$   
**using**  $productF\text{-path-io-targets-reverse}$   
 $[OF \langle productF M2 M1 FAIL PM \rangle \langle s2 \in io\text{-targets } M2 (initial M2) (vs') \rangle$   
 $\langle s1 \in io\text{-targets } M1 (initial M1) (vs') \rangle \langle vs' \in L M2 \rangle \langle vs' \in L M1 \rangle ]$

**proof** –  
**have**  $\forall c f. c \neq initial (f::('a, 'b, 'c) FSM) \vee c \in nodes f$   
**by**  $blast$   
**then show**  $?thesis$   
**by** ( $metis (no-types) \langle [observable M2; observable M1; well\text{-formed } M2; well\text{-formed } M1;$   
 $initial M2 \in nodes M2; initial M1 \in nodes M1] \rangle$   
 $\implies io\text{-targets } PM (initial M2, initial M1) (vs') = \{(s2, s1)\}$   
 $assms(2) assms(3)$ )

**qed**

**have**  $io\text{-targets } PM (initial PM) (vs') = \{(s2, s1)\}$   
**by** ( $metis (no-types) \langle io\text{-targets } PM (initial M2, initial M1) vs' = \{(s2, s1)\} \rangle$   
 $\langle productF M2 M1 FAIL PM \rangle productF\text{-simps}(4)$ )

**have**  $sequence\text{-to-failure } M1 M2 (vs@xs)$   
**using**  $assms(1)$  **by**  $auto$

**have**  $xs = xs' @ (drop (length xs') xs)$   
**by** ( $metis \langle prefix xs' xsR \rangle append\text{-assoc} append\text{-eq-conv-conj} assms(6) prefixE$ )  
**then have**  $io\text{-targets } PM (initial M2, initial M1) (vs @ xs' @ (drop (length xs') xs)) = \{FAIL\}$   
**by** ( $metis \langle productF M2 M1 FAIL PM \rangle \langle sequence\text{-to-failure } M1 M2 (vs @ xs) \rangle assms(2) assms(3)$   
 $productF\text{-simps}(4) sequence\text{-to-failure-reaches-FAIL-ob}$ )  
**then have**  $io\text{-targets } PM (initial M2, initial M1) ((vs @ xs') @ (drop (length xs') xs)) = \{FAIL\}$   
**by**  $auto$   
**have**  $io\text{-targets } PM (s2, s1) (drop (length xs') xs) = \{FAIL\}$   
**using**  $observable\text{-io-targets-split}$   
 $[OF \langle observable PM \rangle$   
 $\langle io\text{-targets } PM (initial M2, initial M1) ((vs @ xs') @ (drop (length xs') xs)) = \{FAIL\} \rangle$   
 $\langle io\text{-targets } PM (initial M2, initial M1) (vs @ xs') = \{(s2, s1)\} \rangle]$   
**by**  $assumption$

**have**  $io\text{-targets } PM (initial PM) (vs' @ (drop (length xs') xs)) = \{FAIL\}$   
**using**  $observable\text{-io-targets-append}$   
 $[OF \langle observable PM \rangle \langle io\text{-targets } PM (initial PM) (vs') = \{(s2, s1)\} \rangle$   
 $\langle io\text{-targets } PM (s2, s1) (drop (length xs') xs) = \{FAIL\} \rangle]$   
**by**  $assumption$

**have**  $sequence\text{-to-failure } M1 M2 (vs' @ (drop (length xs') xs))$   
**using**  $sequence\text{-to-failure-alt-def}$

```

    [OF ‹io-targets PM (initial PM) (vs' @ (drop (length xs') xs)) = {FAIL}› assms(2,3)]
    assms(4)
  by blast

have length (drop (length xs') xs) < length xs
by (metis (no-types) ‹xs = xs' @ drop (length xs') xs› ‹xs' ≠ []› length-append
length-greater-0-conv less-add-same-cancel2)

have vs' ∈ Lin M1 V
proof -
  have V'' ∈ Perm V M1
  using assms(5) unfolding N.simps by blast

  then obtain f where f-def : V'' = image f V
    ∧ (∀ v ∈ V . f v ∈ language-state-for-input M1 (initial M1) v)
  unfolding Perm.simps by blast
  then obtain v where v ∈ V vs' = f v
  using ‹vs' ∈ V''› by auto
  then have vs' ∈ language-state-for-input M1 (initial M1) v
  using f-def by auto

  have language-state-for-input M1 (initial M1) v = Lin M1 {v}
  by auto
  moreover have {v} ⊆ V
  using ‹v ∈ V› by blast
  ultimately have language-state-for-input M1 (initial M1) v ⊆ Lin M1 V
  unfolding language-state-for-inputs.simps language-state-for-input.simps by blast
  then show ?thesis
  using ‹vs' ∈ language-state-for-input M1 (initial M1) v› by blast
qed

have ¬ minimal-sequence-to-failure-extending V M1 M2 vs xs
using ‹vs' ∈ Lin M1 V›
  ‹sequence-to-failure M1 M2 (vs' @ (drop (length xs') xs))›
  ‹length (drop (length xs') xs) < length xs›
using minimal-sequence-to-failure-extending.elims(2) by blast
then show False
using assms(1) by linarith
qed

```

```

lemma mstfe-no-repetition :
  assumes minimal-sequence-to-failure-extending V M1 M2 vs xs
  and OFSM M1
  and OFSM M2
  and test-tools M2 M1 FAIL PM V Ω
  and V'' ∈ N (vs@xs') M1 V
  and prefix xs' xs
shows ¬ Rep-Pre M2 M1 vs xs'
and ¬ Rep-Cov M2 M1 V'' vs xs'
using minimal-sequence-to-failure-extending-implies-Rep-Pre[OF assms]
  minimal-sequence-to-failure-extending-implies-Rep-Cov[OF assms]
by linarith+

```

## 6.2 Sufficiency of the test suite to test for reduction

The following lemma proves that set of input sequences generated in the final iteration of the TS function constitutes a test suite sufficient to test for reduction the FSMs it has been generated for.

This proof is performed by contradiction: If the test suite is not sufficient, then some minimal sequence to a failure extending the deterministic state cover must exist. Due to the test suite being assumed insufficient, this sequence cannot be contained in it and hence a prefix of it must have been contained in one of the sets calculated by the R function. This is only possible if the prefix is not a minimal sequence to a failure extending the deterministic state cover or if the test suite observes a failure, both of which violates the assumptions.

**lemma** *asc-sufficiency* :  
**assumes** *OFSM M1*  
**and** *OFSM M2*  
**and** *asc-fault-domain M2 M1 m*  
**and** *test-tools M2 M1 FAIL PM V  $\Omega$*   
**and** *final-iteration M2 M1  $\Omega$  V m i*  
**shows**  $M1 \preceq [(TS\ M2\ M1\ \Omega\ V\ m\ i) . \Omega] M2 \longrightarrow M1 \preceq M2$   
**proof**  
**assume** *atc-io-reduction-on-sets M1 (TS M2 M1  $\Omega$  V m i)  $\Omega$  M2*  
**show**  $M1 \preceq M2$   
**proof** (*rule ccontr*)  
  
**let**  $?TS = \lambda n . TS\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?C = \lambda n . C\ M2\ M1\ \Omega\ V\ m\ n$   
**let**  $?RM = \lambda n . RM\ M2\ M1\ \Omega\ V\ m\ n$   
  
**assume**  $\neg M1 \preceq M2$   
**obtain** *vs xs where minimal-sequence-to-failure-extending V M1 M2 vs xs*  
**using** *assms(1) assms(2) assms(4)*  
*minimal-sequence-to-failure-extending-det-state-cover-ob[OF - - - -  $\neg M1 \preceq M2$ , of V]*  
**by** *blast*  
  
**then have**  $vs \in L_{in}\ M1\ V$   
*sequence-to-failure M1 M2 (vs @ xs)*  
 $\neg (\exists io' . \exists w' \in L_{in}\ M1\ V . \text{sequence-to-failure } M1\ M2\ (w' @ io')$   
 $\wedge \text{length } io' < \text{length } xs)$   
**by** *auto*  
  
**then have**  $vs@xs \in L\ M1 - L\ M2$   
**by** *auto*  
  
**have**  $vs@xs \in L_{in}\ M1\ \{\text{map } fst\ (vs@xs)\}$   
**by** (*metis (full-types) Diff-iff  $\langle vs @ xs \in L\ M1 - L\ M2 \rangle$  insertI1*  
*language-state-for-inputs-map-fst*)  
  
**have**  $vs@xs \notin L_{in}\ M2\ \{\text{map } fst\ (vs@xs)\}$   
**by** (*meson Diff-iff  $\langle vs @ xs \in L\ M1 - L\ M2 \rangle$  language-state-for-inputs-in-language-state*  
*subsetCE*)  
  
**have** *finite V*  
**using** *det-state-cover-finite assms(4,2) by auto*  
**then have** *finite (?TS i)*  
**using** *TS-finite[of V M2] assms(2) by auto*  
**then have** *io-reduction-on M1 (?TS i) M2*  
**using** *io-reduction-from-atc-io-reduction*  
 $[OF\ \langle \text{atc-io-reduction-on-sets } M1\ (TS\ M2\ M1\ \Omega\ V\ m\ i)\ \Omega\ M2 \rangle]$   
**by** *auto*  
  
**have**  $\text{map } fst\ (vs@xs) \notin ?TS\ i$   
**proof** -  
**have**  $f1: \forall ps\ P\ Pa . (ps::('a \times 'b)\ \text{list}) \notin P - Pa \vee ps \in P \wedge ps \notin Pa$   
**by** *blast*  
**have**  $\forall P\ Pa\ ps . \neg P \subseteq Pa \vee (ps::('a \times 'b)\ \text{list}) \in Pa \vee ps \notin P$   
**by** *blast*  
**then show** *thesis*  
**using**  $f1$  **by** (*metis (no-types)  $\langle vs @ xs \in L\ M1 - L\ M2 \rangle$   $\langle \text{io-reduction-on } M1\ (?TS\ i)\ M2 \rangle$*   
*language-state-for-inputs-in-language-state language-state-for-inputs-map-fst*)  
**qed**  
  
**have**  $\text{map } fst\ vs \in V$   
**using**  $\langle vs \in L_{in}\ M1\ V \rangle$  **by** *auto*  
  
**let**  $?stf = \text{map } fst\ (vs@xs)$   
**let**  $?stfV = \text{map } fst\ vs$   
**let**  $?stfX = \text{map } fst\ xs$

**have**  $?stf = ?stfV @ ?stfX$   
**by** *simp*

**then have**  $?stfV @ ?stfX \notin ?TS\ i$   
**using**  $\langle ?stf \notin ?TS\ i \rangle$  **by** *auto*

**have**  $mcp (?stfV @ ?stfX)\ V\ ?stfV$   
**by** (*metis*  $\langle map\ fst\ (vs @ xs) = map\ fst\ vs @ map\ fst\ xs \rangle$   
 $\langle minimal-sequence-to-failure-extending\ V\ M1\ M2\ vs\ xs \rangle$  *assms*(1) *assms*(2) *assms*(4)  
*minimal-sequence-to-failure-extending-mcp*)

**have**  $set\ ?stf \subseteq inputs\ M1$   
**by** (*meson* *DiffD1*  $\langle vs @ xs \in L\ M1 - L\ M2 \rangle$  *assms*(1) *language-state-inputs*)  
**then have**  $set\ ?stf \subseteq inputs\ M2$   
**using** *assms*(3) **by** *blast*  
**moreover have**  $set\ ?stf = set\ ?stfV \cup set\ ?stfX$   
**by** *simp*  
**ultimately have**  $set\ ?stfX \subseteq inputs\ M2$   
**by** *blast*

**obtain**  $xr\ j$  **where**  $xr \neq ?stfX$   
 $prefix\ xr\ ?stfX$   
 $Suc\ j \leq i$   
 $?stfV @ xr \in RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ j)$   
**using** *TS-non-containment-causes-final-suc*[*OF*  $\langle ?stfV @ ?stfX \notin ?TS\ i \rangle$   
 $\langle mcp\ (?stfV @ ?stfX)\ V\ ?stfV \rangle$   $\langle set\ ?stfX \subseteq inputs\ M2 \rangle$  *assms*(5,2)]  
**by** *blast*

**let**  $?yr = take\ (length\ xr)\ (map\ snd\ xs)$   
**have**  $length\ ?yr = length\ xr$   
**using**  $\langle prefix\ xr\ (map\ fst\ xs) \rangle$  *prefix-length-le* **by** *fastforce*  
**have**  $(xr \parallel ?yr) = take\ (length\ xr)\ xs$   
**by** (*metis* (*no-types*, *opaque-lifting*)  $\langle prefix\ xr\ (map\ fst\ xs) \rangle$  *append-eq-conv-conj* *prefixE* *take-zip*  
*zip-map-fst-snd*)

**have**  $prefix\ (vs@(xr \parallel ?yr))\ (vs@xs)$   
**by** (*simp* *add*:  $\langle xr \parallel take\ (length\ xr)\ (map\ snd\ xs) = take\ (length\ xr)\ xs \rangle$  *take-is-prefix*)

**have**  $xr = take\ (length\ xr)\ (map\ fst\ xs)$   
**by** (*metis*  $\langle length\ (take\ (length\ xr)\ (map\ snd\ xs)) = length\ xr \rangle$   
 $\langle xr \parallel take\ (length\ xr)\ (map\ snd\ xs) = take\ (length\ xr)\ xs \rangle$  *map-fst-zip* *take-map*)

**have**  $vs@(xr \parallel ?yr) \in L\ M1$   
**by** (*metis* *DiffD1*  $\langle prefix\ (vs @ (xr \parallel take\ (length\ xr)\ (map\ snd\ xs))) \rangle$   $\langle vs @ xs \rangle$   
 $\langle vs @ xs \in L\ M1 - L\ M2 \rangle$  *language-state-prefix* *prefixE*)

**then have**  $vs@(xr \parallel ?yr) \in L_{in}\ M1\ \{?stfV @ xr\}$   
**by** (*metis*  $\langle length\ (take\ (length\ xr)\ (map\ snd\ xs)) = length\ xr \rangle$  *insertI1*  
*language-state-for-inputs-map-fst* *map-append* *map-fst-zip*)

**have**  $length\ xr < length\ xs$   
**by** (*metis*  $\langle xr = take\ (length\ xr)\ (map\ fst\ xs) \rangle$   $\langle xr \neq map\ fst\ xs \rangle$  *not-le-imp-less* *take-all*  
*take-map*)

**from**  $\langle ?stfV @ xr \in RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) \rangle$  **have**  $?stfV @ xr \in \{xs' \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ j)\}$ .  
 $(\neg (L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\}))$   
 $\vee (\forall\ io \in L_{in}\ M1\ \{xs'\} .$   
 $(\exists\ V'' \in N\ io\ M1\ V .$   
 $(\exists\ S1 .$   
 $(\exists\ vs\ xs .$   
 $io = (vs@xs)$   
 $\wedge\ mcp\ (vs@xs)\ V''\ vs$

$\wedge S1 \subseteq \text{nodes } M2$   
 $\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP\ M2\ s1\ vs\ xs\ V'' .$   
 $\quad \forall io2 \in RP\ M2\ s2\ vs\ xs\ V'' .$   
 $\quad B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega ))$   
 $\wedge m < LB\ M2\ M1\ vs\ xs\ (TS\ M2\ M1\ \Omega\ V\ m\ j\ \cup\ V)\ S1\ \Omega\ V'' ))\}$   
**unfolding** *RM.simps* **by** *blast*

**moreover have**  $\forall xs' \in ?C\ (Suc\ j) . L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\}$

**proof**

**fix** *xs'* **assume**  $xs' \in ?C\ (Suc\ j)$   
**from**  $\langle Suc\ j \leq i \rangle$  **have**  $?C\ (Suc\ j) \subseteq ?TS\ i$   
**using** *C-subset TS-subset* **by** *blast*  
**then have**  $\{xs'\} \subseteq ?TS\ i$   
**using**  $\langle xs' \in ?C\ (Suc\ j) \rangle$  **by** *blast*  
**show**  $L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\}$   
**using** *io-reduction-on-subset[OF <io-reduction-on M1 (?TS i) M2> <{xs'} <= ?TS i>]*  
**by** *assumption*

**qed**

**ultimately have**  $(\forall io \in L_{in}\ M1\ \{?stfV@xr\} .$

$(\exists V'' \in N\ io\ M1\ V .$

$(\exists S1 .$

$(\exists vs\ xs .$

$io = (vs@xs)$

$\wedge mcp\ (vs@xs)\ V''\ vs$

$\wedge S1 \subseteq \text{nodes } M2$

$\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP\ M2\ s1\ vs\ xs\ V'' .$

$\quad \forall io2 \in RP\ M2\ s2\ vs\ xs\ V'' .$

$\quad B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega ))$

$\wedge m < LB\ M2\ M1\ vs\ xs\ (TS\ M2\ M1\ \Omega\ V\ m\ j\ \cup\ V)\ S1\ \Omega\ V'' ))\}$

**by** *blast*

**then have**

$(\exists V'' \in N\ (vs@(xr \parallel ?yr))\ M1\ V .$

$(\exists S1 .$

$(\exists vs'\ xs' .$

$vs@(xr \parallel ?yr) = (vs'@xs')$

$\wedge mcp\ (vs'@xs')\ V''\ vs'$

$\wedge S1 \subseteq \text{nodes } M2$

$\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP\ M2\ s1\ vs'\ xs'\ V'' .$

$\quad \forall io2 \in RP\ M2\ s2\ vs'\ xs'\ V'' .$

$\quad B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega ))$

$\wedge m < LB\ M2\ M1\ vs'\ xs'\ (TS\ M2\ M1\ \Omega\ V\ m\ j\ \cup\ V)\ S1\ \Omega\ V'' ))\}$

**using**  $\langle vs@(xr \parallel ?yr) \in L_{in}\ M1\ \{?stfV\ @\ xr\} \rangle$

**by** *blast*

**then obtain**  $V''\ S1\ vs'\ xs'$  **where** *RM-impl* :

$V'' \in N\ (vs@(xr \parallel ?yr))\ M1\ V$

$vs@(xr \parallel ?yr) = (vs'@xs')$

$mcp\ (vs'@xs')\ V''\ vs'$

$S1 \subseteq \text{nodes } M2$

$(\forall s1 \in S1 . \forall s2 \in S1 .$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP\ M2\ s1\ vs'\ xs'\ V'' .$

$\quad \forall io2 \in RP\ M2\ s2\ vs'\ xs'\ V'' .$

$\quad B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega ))$

$m < LB\ M2\ M1\ vs'\ xs'\ (TS\ M2\ M1\ \Omega\ V\ m\ j\ \cup\ V)\ S1\ \Omega\ V''$

**by** *blast*

**have**  $?stfV = mcp' (map\ fst\ (vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs))))\ V$   
**by**  $(metis\ (full-types)\ \langle length\ (take\ (length\ xr)\ (map\ snd\ xs)) = length\ xr \rangle$   
 $\langle mcp\ (map\ fst\ vs\ @\ map\ fst\ xs)\ V\ (map\ fst\ vs) \rangle\ \langle prefix\ xr\ (map\ fst\ xs) \rangle\ map-append$   
 $map-fst-zip\ mcp'-intro\ mcp-prefix-of-suffix)$

**have**  $is-det-state-cover\ M2\ V$   
**using**  $assms(4)$  **by**  $blast$   
**moreover** **have**  $well-formed\ M2$   
**using**  $assms(2)$  **by**  $auto$   
**moreover** **have**  $finite\ V$   
**using**  $det-state-cover-finite\ assms(4,2)$  **by**  $auto$   
**ultimately** **have**  $vs \in V''$   
 $vs = mcp' (vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs)))\ V''$   
**using**  $N-mcp-prefix[OF\ \langle ?stfV = mcp' (map\ fst\ (vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs))))\ V \rangle$   
 $\langle V'' \in N\ (vs@(xr\ ||\ ?yr))\ M1\ V \rangle, of\ M2]$   
**by**  $simp+$

**have**  $vs' = vs$   
**by**  $(metis\ (no-types)\ \langle mcp\ (vs'\ @\ xs')\ V''\ vs' \rangle$   
 $\langle vs = mcp' (vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs)))\ V'' \rangle$   
 $\langle vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs)) = vs'\ @\ xs' \rangle\ mcp'-intro)$

**then** **have**  $xs' = (xr\ ||\ ?yr)$   
**using**  $\langle vs\ @\ (xr\ ||\ take\ (length\ xr)\ (map\ snd\ xs)) = vs'\ @\ xs' \rangle$  **by**  $blast$

**have**  $V \subseteq ?TS\ i$   
**proof**  $-$   
**have**  $1 \leq i$   
**using**  $\langle Suc\ j \leq i \rangle$  **by**  $linarith$   
**then** **have**  $?TS\ 1 \subseteq ?TS\ i$   
**using**  $TS-subset$  **by**  $blast$   
**then** **show**  $?thesis$   
**by**  $auto$   
**qed**

**have**  $?stfV@xr \in ?C\ (Suc\ j)$   
**using**  $\langle ?stfV@xr \in RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) \rangle$  **unfolding**  $RM.simps$  **by**  $blast$

— show that the prerequisites (**Prereq**) for LB are met by construction

**have**  $(\forall\ vs'a \in V''.\ prefix\ vs'a\ (vs'\ @\ xs') \longrightarrow length\ vs'a \leq length\ vs')$   
**using**  $\langle mcp\ (vs'\ @\ xs')\ V''\ vs' \rangle$  **by**  $auto$

**moreover** **have**  $atc-io-reduction-on-sets\ M1\ (?TS\ j \cup V)\ \Omega\ M2$   
**proof**  $-$   
**have**  $j < i$   
**using**  $\langle Suc\ j \leq i \rangle$  **by**  $auto$   
**then** **have**  $?TS\ j \subseteq ?TS\ i$   
**by**  $(simp\ add:\ TS-subset)$   
**then** **show**  $?thesis$   
**using**  $atc-io-reduction-on-subset$   
 $[OF\ \langle atc-io-reduction-on-sets\ M1\ (TS\ M2\ M1\ \Omega\ V\ m\ i)\ \Omega\ M2 \rangle, of\ ?TS\ j]$   
**by**  $(meson\ Un-subset-iff\ \langle V \subseteq ?TS\ i \rangle\ \langle atc-io-reduction-on-sets\ M1\ (TS\ M2\ M1\ \Omega\ V\ m\ i)\ \Omega\ M2 \rangle$   
 $atc-io-reduction-on-subset)$   
**qed**

**moreover** **have**  $finite\ (?TS\ j \cup V)$   
**proof**  $-$   
**have**  $finite\ (?TS\ j)$   
**using**  $TS-finite[OF\ \langle finite\ V \rangle, of\ M2\ M1\ \Omega\ m\ j]$   $assms(2)$  **by**  $auto$   
**then** **show**  $?thesis$   
**using**  $\langle finite\ V \rangle$  **by**  $blast$   
**qed**

**moreover have**  $V \subseteq ?TS\ j \cup V$   
**by** *blast*

**moreover have**  $(\forall p . (prefix\ p\ xs' \wedge p \neq xs') \longrightarrow map\ fst\ (vs' @ p) \in ?TS\ j \cup V)$   
**proof**

**fix**  $p$

**show**  $prefix\ p\ xs' \wedge p \neq xs' \longrightarrow map\ fst\ (vs' @ p) \in TS\ M2\ M1\ \Omega\ V\ m\ j \cup V$

**proof**

**assume**  $prefix\ p\ xs' \wedge p \neq xs'$

**have**  $prefix\ (map\ fst\ (vs' @ p))\ (map\ fst\ (vs' @ xs'))$

**by**  $(simp\ add: \langle prefix\ p\ xs' \wedge p \neq xs' \rangle\ map\ mono\ prefix)$

**have**  $prefix\ (map\ fst\ (vs' @ p))\ (?stfV @ xr)$

**using**  $\langle length\ (take\ (length\ xr)\ (map\ snd\ xs)) = length\ xr \rangle$

$\langle prefix\ (map\ fst\ (vs' @ p))\ (map\ fst\ (vs' @ xs')) \rangle$

$\langle vs' = vs \rangle\ \langle xs' = xr \parallel take\ (length\ xr)\ (map\ snd\ xs) \rangle$

**by** *auto*

**then have**  $prefix\ (map\ fst\ vs' @ map\ fst\ p)\ (?stfV @ xr)$

**by** *simp*

**then have**  $prefix\ (map\ fst\ p)\ xr$

**by**  $(simp\ add: \langle vs' = vs \rangle)$

**have**  $?stfV @ xr \in ?TS\ (Suc\ j)$

**proof**  $(cases\ j)$

**case**  $0$

**then show** *?thesis*

**using**  $\langle map\ fst\ vs @ xr \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) \rangle$  **by** *auto*

**next**

**case**  $(Suc\ nat)$

**then show** *?thesis*

**using**  $TS.simps(\beta)\ \langle map\ fst\ vs @ xr \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ j) \rangle$  **by** *blast*

**qed**

**have**  $mcp\ (map\ fst\ vs @ xr)\ V\ (map\ fst\ vs)$

**using**  $\langle mcp\ (map\ fst\ vs @ map\ fst\ xs)\ V\ (map\ fst\ vs) \rangle\ \langle prefix\ xr\ (map\ fst\ xs) \rangle$   
*mcp-prefix-of-suffix*

**by** *blast*

**have**  $map\ fst\ vs @ map\ fst\ p \in TS\ M2\ M1\ \Omega\ V\ m\ (Suc\ j)$

**using**  $TS.prefix\ containment[OF\ \langle ?stfV @ xr \in ?TS\ (Suc\ j) \rangle$

$\langle mcp\ (map\ fst\ vs @ xr)\ V\ (map\ fst\ vs) \rangle$

$\langle prefix\ (map\ fst\ p)\ xr \rangle]$

**by** *assumption*

**have**  $Suc\ (length\ xr) = (Suc\ j)$

**using**  $C-index[OF\ \langle ?stfV @ xr \in ?C\ (Suc\ j) \rangle\ \langle mcp\ (map\ fst\ vs @ xr)\ V\ (map\ fst\ vs) \rangle]$

**by** *assumption*

**have**  $Suc\ (length\ p) < (Suc\ j)$

**proof**  $-$

**have**  $map\ fst\ xs' = xr$

**by**  $(metis\ \langle xr = take\ (length\ xr)\ (map\ fst\ xs) \rangle$

$\langle xr \parallel take\ (length\ xr)\ (map\ snd\ xs) = take\ (length\ xr)\ xs \rangle$

$\langle xs' = xr \parallel take\ (length\ xr)\ (map\ snd\ xs) \rangle\ take\ map)$

**then show** *?thesis*

**by**  $(metis\ (no-types)\ Suc\ less\ eq\ \langle Suc\ (length\ xr) = Suc\ j \rangle\ \langle prefix\ p\ xs' \wedge p \neq xs' \rangle$

$append\ eq\ conv\ conj\ length\ map\ nat\ less\ le\ prefixE\ prefix\ length\ le\ take\ all)$

**qed**

**have**  $mcp\ (map\ fst\ vs @ map\ fst\ p)\ V\ (map\ fst\ vs)$

**using**  $\langle mcp\ (map\ fst\ vs @ xr)\ V\ (map\ fst\ vs) \rangle\ \langle prefix\ (map\ fst\ p)\ xr \rangle\ mcp\ prefix\ of\ suffix$

**by** *blast*

**then have**  $map\ fst\ vs @ map\ fst\ p \in ?C\ (Suc\ (length\ (map\ fst\ p)))$

```

using TS-index(2)[OF ⟨map fst vs @ map fst p ∈ TS M2 M1 Ω V m (Suc j)⟩] by auto

have map fst vs @ map fst p ∈ ?TS j
  using TS-union[of M2 M1 Ω V m j]
proof -
  have Suc (length p) ∈ {0..

```

```

moreover have vs' @ xs' ∈ L M2 ∩ L M1
  by (metis (no-types, lifting) IntI RM-impl(2)
    ⟨∀xs'∈C M2 M1 Ω V m (Suc j). Lin M1 {xs'} ⊆ Lin M2 {xs'}⟩
    ⟨map fst vs @ xr ∈ C M2 M1 Ω V m (Suc j)⟩
    ⟨vs @ (xr || take (length xr) (map snd xs)) ∈ Lin M1 {map fst vs @ xr}⟩
    language-state-for-inputs-in-language-state subsetCE)

```

```

ultimately have Prereq M2 M1 vs' xs' (?TS j ∪ V) S1 Ω V''
  using RM-impl(4,5) unfolding Prereq.simps by blast

```

```

have V'' ∈ Perm V M1
  using ⟨V'' ∈ N (vs@(xr || ?yr)) M1 V⟩ unfolding N.simps by blast

```

```

have ⟨prefix (xr || ?yr) xs⟩
  by (simp add: ⟨xr || take (length xr) (map snd xs) = take (length xr) xs⟩ take-is-prefix)

```

— show that furthermore neither **Rep\_Pre** nor **Rep\_Cov** holds

```

have ¬ Rep-Pre M2 M1 vs (xr || ?yr)
  using minimal-sequence-to-failure-extending-implies-Rep-Pre
    [OF ⟨minimal-sequence-to-failure-extending V M1 M2 vs xs⟩ assms(1,2)
    ⟨test-tools M2 M1 FAIL PM V Ω⟩ RM-impl(1)
    ⟨prefix (xr || take (length xr) (map snd xs)) xs⟩]
  by assumption
then have ¬ Rep-Pre M2 M1 vs' xs'
  using ⟨vs' = vs⟩ ⟨xs' = xr || ?yr⟩ by blast

```

```

have ¬ Rep-Cov M2 M1 V'' vs (xr || ?yr)
  using minimal-sequence-to-failure-extending-implies-Rep-Cov
    [OF ⟨minimal-sequence-to-failure-extending V M1 M2 vs xs⟩ assms(1,2)
    ⟨test-tools M2 M1 FAIL PM V Ω⟩ RM-impl(1)
    ⟨prefix (xr || take (length xr) (map snd xs)) xs⟩]
  by assumption
then have ¬ Rep-Cov M2 M1 V'' vs' xs'
  using ⟨vs' = vs⟩ ⟨xs' = xr || ?yr⟩ by blast

```

```

have vs'@xs' ∈ L M1
  using ⟨vs @ (xr || take (length xr) (map snd xs)) ∈ L M1⟩
    ⟨vs' = vs⟩ ⟨xs' = xr || take (length xr) (map snd xs)⟩
  by blast

```

— therefore it is impossible to remove the prefix of the minimal sequence to a failure, as this would require **M1** to have more than  $m$  states

```

have  $LB\ M2\ M1\ vs'\ xs' (?TS\ j\ \cup\ V)\ S1\ \Omega\ V'' \leq card\ (nodes\ M1)$ 
using  $LB\ count[OF\ \langle vs' @ xs' \in L\ M1 \rangle\ assms(1,2,3)\ \langle test\ tools\ M2\ M1\ FAIL\ PM\ V\ \Omega \rangle$ 
 $\langle V'' \in Perm\ V\ M1 \rangle\ \langle Prereq\ M2\ M1\ vs'\ xs' (?TS\ j\ \cup\ V)\ S1\ \Omega\ V'' \rangle$ 
 $\langle \neg\ Rep\ Pre\ M2\ M1\ vs'\ xs' \rangle\ \langle \neg\ Rep\ Cov\ M2\ M1\ V''\ vs'\ xs' \rangle]$ 
by assumption
then have  $LB\ M2\ M1\ vs'\ xs' (?TS\ j\ \cup\ V)\ S1\ \Omega\ V'' \leq m$ 
using  $assms(3)$  by linarith

then show False
using  $\langle m < LB\ M2\ M1\ vs'\ xs' (?TS\ j\ \cup\ V)\ S1\ \Omega\ V'' \rangle$  by linarith
qed
qed

```

### 6.3 Main result

The following lemmata add to the previous result to show that some FSM  $M1$  is a reduction of FSM  $M2$  if and only if it is a reduction on the test suite generated by the adaptive state counting algorithm for these FSMs.

```

lemma asc-soundness :
assumes  $OFSM\ M1$ 
and  $OFSM\ M2$ 
shows  $M1 \preceq M2 \longrightarrow atc\ io\ reduction\ on\ sets\ M1\ T\ \Omega\ M2$ 
using atc-io-reduction-on-sets-reduction assms by blast

```

```

lemma asc-main-theorem :
assumes  $OFSM\ M1$ 
and  $OFSM\ M2$ 
and  $asc\ fault\ domain\ M2\ M1\ m$ 
and  $test\ tools\ M2\ M1\ FAIL\ PM\ V\ \Omega$ 
and  $final\ iteration\ M2\ M1\ \Omega\ V\ m\ i$ 
shows  $M1 \preceq M2 \iff atc\ io\ reduction\ on\ sets\ M1\ (TS\ M2\ M1\ \Omega\ V\ m\ i)\ \Omega\ M2$ 
by (metis asc-sufficiency  $assms(1-5)$  atc-io-reduction-on-sets-reduction)

```

```

end
theory ASC-Hoare
imports ASC-Sufficiency HOL-Hoare.Hoare-Logic
begin

```

## 7 Correctness of the Adaptive State Counting Algorithm in Hoare-Logic

In this section we give an example implementation of the adaptive state counting algorithm in a simple WHILE-language and prove that this implementation produces a certain output if and only if input FSM  $M1$  is a reduction of input FSM  $M2$ .

```

lemma atc-io-reduction-on-sets-from-obs :
assumes  $L_{in}\ M1\ T \subseteq L_{in}\ M2\ T$ 
and  $(\bigcup_{io \in L_{in}\ M1\ T} \{io\} \times B\ M1\ io\ \Omega) \subseteq (\bigcup_{io \in L_{in}\ M2\ T} \{io\} \times B\ M2\ io\ \Omega)$ 
shows atc-io-reduction-on-sets  $M1\ T\ \Omega\ M2$ 
unfolding atc-io-reduction-on-sets.simps atc-io-reduction-on.simps
proof
fix  $iseq$  assume  $iseq \in T$ 
have  $L_{in}\ M1\ \{iseq\} \subseteq L_{in}\ M2\ \{iseq\}$ 
by (metis  $\langle iseq \in T \rangle$   $assms(1)$  bot.extremum insert-mono io-reduction-on-subset mk-disjoint-insert)
moreover have  $\forall io \in L_{in}\ M1\ \{iseq\}. B\ M1\ io\ \Omega \subseteq B\ M2\ io\ \Omega$ 
proof
fix  $io$  assume  $io \in L_{in}\ M1\ \{iseq\}$ 
then have  $io \in L_{in}\ M2\ \{iseq\}$ 

```

using *calculation by blast*  
 show  $B M1 io \Omega \subseteq B M2 io \Omega$   
 proof  
 fix  $x$  assume  $x \in B M1 io \Omega$   
  
 have  $io \in L_{in} M1 T$   
 using  $\langle io \in L_{in} M1 \{iseq\} \rangle \langle iseq \in T \rangle$  by *auto*  
 moreover have  $(io, x) \in \{io\} \times B M1 io \Omega$   
 using  $\langle x \in B M1 io \Omega \rangle$  by *blast*  
 ultimately have  $(io, x) \in (\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega})$   
 by *blast*  
  
 then have  $(io, x) \in (\bigcup_{io \in L_{in} M2 T. \{io\} \times B M2 io \Omega})$   
 using *assms(2)* by *blast*  
 then have  $(io, x) \in \{io\} \times B M2 io \Omega$   
 by *blast*  
 then show  $x \in B M2 io \Omega$   
 by *blast*  
 qed  
 qed  
 ultimately show  $L_{in} M1 \{iseq\} \subseteq L_{in} M2 \{iseq\}$   
 $\wedge (\forall io \in L_{in} M1 \{iseq\}. B M1 io \Omega \subseteq B M2 io \Omega)$   
 by *linarith*  
 qed

lemma *atc-io-reduction-on-sets-to-obs* :

assumes *atc-io-reduction-on-sets*  $M1 T \Omega M2$   
 shows  $L_{in} M1 T \subseteq L_{in} M2 T$   
 and  $(\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega) \subseteq (\bigcup_{io \in L_{in} M2 T. \{io\} \times B M2 io \Omega})$   
 proof  
 fix  $x$  assume  $x \in L_{in} M1 T$   
 show  $x \in L_{in} M2 T$   
 using *assms unfolding atc-io-reduction-on-sets.simps atc-io-reduction-on.simps*  
 proof –  
 assume *a1*:  $\forall iseq \in T. L_{in} M1 \{iseq\} \subseteq L_{in} M2 \{iseq\}$   
 $\wedge (\forall io \in L_{in} M1 \{iseq\}. B M1 io \Omega \subseteq B M2 io \Omega)$   
 have *f2*:  $x \in UNION T$  (*language-state-for-input M1 (initial M1)*)  
 by (*metis (no-types) \langle x \in L\_{in} M1 T \rangle language-state-for-inputs-alt-def*)  
 obtain *aas* :: *'a list set*  $\Rightarrow$  *'a list*  $\Rightarrow$  *'a*  $\times$  *'b list set*  $\Rightarrow$  *'a*  $\times$  *'b list*  $\Rightarrow$  *'a list*  
 where  
 $\forall x0 x1 x2. (\exists v3. v3 \in x0 \wedge x2 \in x1 v3) = (aas x0 x1 x2 \in x0 \wedge x2 \in x1 (aas x0 x1 x2))$   
 by *moura*  
 then have  $\forall ps f A. (ps \notin UNION A f \vee aas A f ps \in A \wedge ps \in f (aas A f ps))$   
 $\wedge (ps \in UNION A f \vee (\forall as. as \notin A \vee ps \notin f as))$   
 by *blast*  
 then show *?thesis*  
 using *f2 a1* by (*metis (no-types) contra-subsetD language-state-for-input-alt-def language-state-for-inputs-alt-def*)

qed

next

show  $(\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega) \subseteq (\bigcup_{io \in L_{in} M2 T. \{io\} \times B M2 io \Omega)$

proof

fix  $iox$  assume  $iox \in (\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega)$

then obtain  $io x$  where  $iox = (io, x)$

by *blast*

have  $io \in L_{in} M1 T$

using  $\langle iox = (io, x) \rangle \langle iox \in (\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega) \rangle$  by *blast*

have  $(io, x) \in \{io\} \times B M1 io \Omega$

using  $\langle iox = (io, x) \rangle \langle iox \in (\bigcup_{io \in L_{in} M1 T. \{io\} \times B M1 io \Omega) \rangle$  by *blast*

then have  $x \in B M1 io \Omega$

by *blast*

then have  $x \in B M2 io \Omega$

using *assms unfolding atc-io-reduction-on-sets.simps atc-io-reduction-on.simps*

**by** (*metis (no-types, lifting) UN-E*  $\langle io \in L_{in} M1 T \rangle$  *language-state-for-input-alt-def*  
*language-state-for-inputs-alt-def subsetCE*)  
**then have**  $(io, x) \in \{io\} \times B M2 io \Omega$   
**by** *blast*  
**then have**  $(io, x) \in (\bigcup_{io \in L_{in} M2 T}. \{io\} \times B M2 io \Omega)$   
**using**  $\langle io \in L_{in} M1 T \rangle$  **by** *auto*  
**then show**  $iox \in (\bigcup_{io \in L_{in} M2 T}. \{io\} \times B M2 io \Omega)$   
**using**  $\langle iox = (io, x) \rangle$  **by** *auto*  
**qed**  
**qed**

**lemma** *atc-io-reduction-on-sets-alt-def* :  
**shows** *atc-io-reduction-on-sets*  $M1 T \Omega M2 =$   
 $(L_{in} M1 T \subseteq L_{in} M2 T$   
 $\wedge (\bigcup_{io \in L_{in} M1 T}. \{io\} \times B M1 io \Omega)$   
 $\subseteq (\bigcup_{io \in L_{in} M2 T}. \{io\} \times B M2 io \Omega))$   
**using** *atc-io-reduction-on-sets-to-obs*[*of*  $M1 T \Omega M2$ ]  
**and** *atc-io-reduction-on-sets-from-obs*[*of*  $M1 T M2 \Omega$ ]  
**by** *blast*

**lemma** *asc-algorithm-correctness*:

**VARS**  $tsN cN rmN obs obsI obs_{\Omega} obsI_{\Omega} iter isReduction$   
 $\{$   
 $OFSM M1 \wedge OFSM M2 \wedge asc\text{-}fault\text{-}domain M2 M1 m \wedge test\text{-}tools M2 M1 FAIL PM V \Omega$   
 $\}$   
 $tsN := \{\};$   
 $cN := V;$   
 $rmN := \{\};$   
 $obs := L_{in} M2 cN;$   
 $obsI := L_{in} M1 cN;$   
 $obs_{\Omega} := (\bigcup_{io \in L_{in} M2 cN}. \{io\} \times B M2 io \Omega);$   
 $obsI_{\Omega} := (\bigcup_{io \in L_{in} M1 cN}. \{io\} \times B M1 io \Omega);$   
 $iter := 1;$   
 $WHILE (cN \neq \{\}) \wedge obsI \subseteq obs \wedge obsI_{\Omega} \subseteq obs_{\Omega}$   
 $INV \{$   
 $0 < iter$   
 $\wedge tsN = TS M2 M1 \Omega V m (iter-1)$   
 $\wedge cN = C M2 M1 \Omega V m iter$   
 $\wedge rmN = RM M2 M1 \Omega V m (iter-1)$   
 $\wedge obs = L_{in} M2 (tsN \cup cN)$   
 $\wedge obsI = L_{in} M1 (tsN \cup cN)$   
 $\wedge obs_{\Omega} = (\bigcup_{io \in L_{in} M2 (tsN \cup cN)}. \{io\} \times B M2 io \Omega)$   
 $\wedge obsI_{\Omega} = (\bigcup_{io \in L_{in} M1 (tsN \cup cN)}. \{io\} \times B M1 io \Omega)$   
 $\wedge OFSM M1 \wedge OFSM M2 \wedge asc\text{-}fault\text{-}domain M2 M1 m \wedge test\text{-}tools M2 M1 FAIL PM V \Omega$   
 $\}$   
**DO**  
 $iter := iter + 1;$   
 $rmN := \{xs' \in cN .$   
 $(\neg (L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\}))$   
 $\vee (\forall io \in L_{in} M1 \{xs'\} .$   
 $(\exists V'' \in N io M1 V .$   
 $(\exists S1 .$   
 $(\exists vs xs .$   
 $io = (vs@xs)$   
 $\wedge mcp (vs@xs) V'' vs$   
 $\wedge S1 \subseteq nodes M2$   
 $\wedge (\forall s1 \in S1 . \forall s2 \in S1 .$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP M2 s1 vs xs V'' .$   
 $\forall io2 \in RP M2 s2 vs xs V'' .$   
 $B M1 io1 \Omega \neq B M1 io2 \Omega))$   
 $\wedge m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V''))));$

```

tsN := tsN ∪ cN;
cN := append-set (cN - rmN) (inputs M2) - tsN;
obs := obs ∪ Lin M2 cN;
obsI := obsI ∪ Lin M1 cN;
obsΩ := obsΩ ∪ (⋃io∈Lin M2 cN. {io} × B M2 io Ω);
obsIΩ := obsIΩ ∪ (⋃io∈Lin M1 cN. {io} × B M1 io Ω)
OD;
isReduction := ((obsI ⊆ obs) ∧ (obsIΩ ⊆ obsΩ))
{
  isReduction = M1 ≲ M2 — variable isReduction is used only as a return value, it is true if and only if M1 is a
reduction of M2
}
proof (vcg)
assume precondition : OFSM M1 ∧ OFSM M2 ∧ asc-fault-domain M2 M1 m ∧ test-tools M2 M1 FAIL PM V Ω
have {} = TS M2 M1 Ω V m (1-1)
  V = C M2 M1 Ω V m 1
  {} = RM M2 M1 Ω V m (1-1)
  Lin M2 V = Lin M2 ({} ∪ V)
  Lin M1 V = Lin M1 ({} ∪ V)
  (⋃io∈Lin M2 V. {io} × B M2 io Ω)
  = (⋃io∈Lin M2 ({} ∪ V). {io} × B M2 io Ω)
  (⋃io∈Lin M1 V. {io} × B M1 io Ω)
  = (⋃io∈Lin M1 ({} ∪ V). {io} × B M1 io Ω)
using precondition by auto
moreover have OFSM M1 ∧ OFSM M2 ∧ asc-fault-domain M2 M1 m ∧ test-tools M2 M1 FAIL PM V Ω
using precondition by assumption
ultimately show 0 < (1::nat) ∧
  {} = TS M2 M1 Ω V m (1 - 1) ∧
  V = C M2 M1 Ω V m 1 ∧
  {} = RM M2 M1 Ω V m (1 - 1) ∧
  Lin M2 V = Lin M2 ({} ∪ V) ∧
  Lin M1 V = Lin M1 ({} ∪ V) ∧
  (⋃io∈Lin M2 V. {io} × B M2 io Ω)
  = (⋃io∈Lin M2 ({} ∪ V). {io} × B M2 io Ω) ∧
  (⋃io∈Lin M1 V. {io} × B M1 io Ω)
  = (⋃io∈Lin M1 ({} ∪ V). {io} × B M1 io Ω) ∧
  OFSM M1 ∧ OFSM M2 ∧ asc-fault-domain M2 M1 m ∧ test-tools M2 M1 FAIL PM V Ω
by linarith+
next
fix tsN cN rmN obs obsI obsΩ obsIΩ iter isReduction
assume precondition : (0 < iter ∧
  tsN = TS M2 M1 Ω V m (iter - 1) ∧
  cN = C M2 M1 Ω V m iter ∧
  rmN = RM M2 M1 Ω V m (iter - 1) ∧
  obs = Lin M2 (tsN ∪ cN) ∧
  obsI = Lin M1 (tsN ∪ cN) ∧
  obsΩ = (⋃io∈Lin M2 (tsN ∪ cN). {io} × B M2 io Ω) ∧
  obsIΩ = (⋃io∈Lin M1 (tsN ∪ cN). {io} × B M1 io Ω) ∧
  OFSM M1 ∧ OFSM M2 ∧ asc-fault-domain M2 M1 m ∧ test-tools M2 M1 FAIL PM V Ω)
  ∧ cN ≠ {} ∧ obsI ⊆ obs ∧ obsIΩ ⊆ obsΩ
then have 0 < iter
  OFSM M1
  OFSM M2
  asc-fault-domain M2 M1 m
  test-tools M2 M1 FAIL PM V Ω
  cN ≠ {}
  obsI ⊆ obs
  tsN = TS M2 M1 Ω V m (iter-1)
  cN = C M2 M1 Ω V m iter
  rmN = RM M2 M1 Ω V m (iter-1)
  obs = Lin M2 (tsN ∪ cN)
  obsI = Lin M1 (tsN ∪ cN)
  obsΩ = (⋃io∈Lin M2 (tsN ∪ cN). {io} × B M2 io Ω)
  obsIΩ = (⋃io∈Lin M1 (tsN ∪ cN). {io} × B M1 io Ω)
by linarith+

```

**obtain**  $k$  where  $iter = Suc\ k$   
**using**  $gr0\text{-implies}\text{-Suc}[OF\ \langle 0 < iter \rangle]$  **by** *blast*  
**then have**  $cN = C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k)$   
 $tsN = TS\ M2\ M1\ \Omega\ V\ m\ k$   
**using**  $\langle cN = C\ M2\ M1\ \Omega\ V\ m\ iter \ \langle tsN = TS\ M2\ M1\ \Omega\ V\ m\ (iter-1) \rangle$  **by** *auto*  
**have**  $TS\ M2\ M1\ \Omega\ V\ m\ iter = TS\ M2\ M1\ \Omega\ V\ m\ (Suc\ k)$   
 $C\ M2\ M1\ \Omega\ V\ m\ iter = C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k)$   
 $RM\ M2\ M1\ \Omega\ V\ m\ iter = RM\ M2\ M1\ \Omega\ V\ m\ (Suc\ k)$   
**using**  $\langle iter = Suc\ k \rangle$  **by** *presburger+*

**have**  $rmN\text{-calc}[simp] : \{xs' \in cN.$   
 $\neg\ io\text{-reduction-on}\ M1\ \{xs'\}\ M2\ \vee$   
 $(\forall\ io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists\ V'' \in N\ io\ M1\ V.$   
 $\exists\ S1\ vs\ xs.$   
 $io = vs\ @\ xs\ \wedge$   
 $mcp\ (vs\ @\ xs)\ V''\ vs\ \wedge$   
 $S1 \subseteq nodes\ M2\ \wedge$   
 $(\forall\ s1 \in S1.$   
 $\forall\ s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall\ io1 \in RP\ M2\ s1\ vs\ xs\ V''.\ \forall\ io2 \in RP\ M2\ s2\ vs\ xs\ V''.$   
 $B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega))\ \wedge$   
 $m < LB\ M2\ M1\ vs\ xs\ (tsN \cup V)\ S1\ \Omega\ V'')\} =$   
 $RM\ M2\ M1\ \Omega\ V\ m\ iter$

**proof** –

**have**  $\{xs' \in cN.$   
 $\neg\ io\text{-reduction-on}\ M1\ \{xs'\}\ M2\ \vee$   
 $(\forall\ io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists\ V'' \in N\ io\ M1\ V.$   
 $\exists\ S1\ vs\ xs.$   
 $io = vs\ @\ xs\ \wedge$   
 $mcp\ (vs\ @\ xs)\ V''\ vs\ \wedge$   
 $S1 \subseteq nodes\ M2\ \wedge$   
 $(\forall\ s1 \in S1.$   
 $\forall\ s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall\ io1 \in RP\ M2\ s1\ vs\ xs\ V''.\ \forall\ io2 \in RP\ M2\ s2\ vs\ xs\ V''.$   
 $B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega))\ \wedge$   
 $m < LB\ M2\ M1\ vs\ xs\ (tsN \cup V)\ S1\ \Omega\ V'')\} =$   
 $\{xs' \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k).$   
 $\neg\ io\text{-reduction-on}\ M1\ \{xs'\}\ M2\ \vee$   
 $(\forall\ io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists\ V'' \in N\ io\ M1\ V.$   
 $\exists\ S1\ vs\ xs.$   
 $io = vs\ @\ xs\ \wedge$   
 $mcp\ (vs\ @\ xs)\ V''\ vs\ \wedge$   
 $S1 \subseteq nodes\ M2\ \wedge$   
 $(\forall\ s1 \in S1.$   
 $\forall\ s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall\ io1 \in RP\ M2\ s1\ vs\ xs\ V''.\ \forall\ io2 \in RP\ M2\ s2\ vs\ xs\ V''.$   
 $B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega))\ \wedge$   
 $m < LB\ M2\ M1\ vs\ xs\ ((TS\ M2\ M1\ \Omega\ V\ m\ k) \cup V)\ S1\ \Omega\ V'')\}$   
**using**  $\langle cN = C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k) \ \langle tsN = TS\ M2\ M1\ \Omega\ V\ m\ k \rangle$  **by** *blast*

**moreover have**  $\{xs' \in C\ M2\ M1\ \Omega\ V\ m\ (Suc\ k).$   
 $\neg\ io\text{-reduction-on}\ M1\ \{xs'\}\ M2\ \vee$   
 $(\forall\ io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists\ V'' \in N\ io\ M1\ V.$   
 $\exists\ S1\ vs\ xs.$   
 $io = vs\ @\ xs\ \wedge$

$mcp (vs @ xs) V'' vs \wedge$   
 $S1 \subseteq nodes M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\quad \forall s2 \in S1.$   
 $\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad (\forall io1 \in RP M2 s1 vs xs V''. \forall io2 \in RP M2 s2 vs xs V''.$   
 $\quad\quad\quad B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $m < LB M2 M1 vs xs ((TS M2 M1 \Omega V m k) \cup V) S1 \Omega V'' \} =$   
 $RM M2 M1 \Omega V m (Suc k)$   
**using**  $RM.simps(2)[of M2 M1 \Omega V m k]$  **by** *blast*  
**ultimately have**  $\{xs' \in cN.$   
 $\quad \neg io\text{-reduction-on } M1 \{xs'\} M2 \vee$   
 $\quad (\forall io \in Lin M1 \{xs'\}.$   
 $\quad\quad \exists V'' \in N io M1 V.$   
 $\quad\quad \exists S1 vs xs.$   
 $\quad\quad\quad io = vs @ xs \wedge$   
 $\quad\quad\quad mcp (vs @ xs) V'' vs \wedge$   
 $\quad\quad\quad S1 \subseteq nodes M2 \wedge$   
 $\quad\quad\quad (\forall s1 \in S1.$   
 $\quad\quad\quad\quad \forall s2 \in S1.$   
 $\quad\quad\quad\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad\quad\quad\quad (\forall io1 \in RP M2 s1 vs xs V''. \forall io2 \in RP M2 s2 vs xs V''.$   
 $\quad\quad\quad\quad\quad\quad B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $\quad\quad\quad\quad m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V'' \} =$   
 $RM M2 M1 \Omega V m (Suc k)$   
**by** *presburger*  
**then show** *?thesis*  
**using**  $\langle iter = Suc k \rangle$  **by** *presburger*  
**qed**  
**moreover have**  $RM M2 M1 \Omega V m iter = RM M2 M1 \Omega V m (iter + 1 - 1)$  **by** *simp*  
**ultimately have**  $rmN\text{-calc}' : \{xs' \in cN.$   
 $\quad \neg io\text{-reduction-on } M1 \{xs'\} M2 \vee$   
 $\quad (\forall io \in Lin M1 \{xs'\}.$   
 $\quad\quad \exists V'' \in N io M1 V.$   
 $\quad\quad \exists S1 vs xs.$   
 $\quad\quad\quad io = vs @ xs \wedge$   
 $\quad\quad\quad mcp (vs @ xs) V'' vs \wedge$   
 $\quad\quad\quad S1 \subseteq nodes M2 \wedge$   
 $\quad\quad\quad (\forall s1 \in S1.$   
 $\quad\quad\quad\quad \forall s2 \in S1.$   
 $\quad\quad\quad\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad\quad\quad\quad (\forall io1 \in RP M2 s1 vs xs V''. \forall io2 \in RP M2 s2 vs xs V''.$   
 $\quad\quad\quad\quad\quad\quad B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $\quad\quad\quad\quad m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V'' \} =$   
 $RM M2 M1 \Omega V m (iter + 1 - 1)$  **by** *presburger*  
**have**  $tsN \cup cN = TS M2 M1 \Omega V m (Suc k)$   
**proof** (*cases k*)  
**case** 0  
**then show** *?thesis*  
**using**  $\langle tsN = TS M2 M1 \Omega V m k \rangle \langle cN = C M2 M1 \Omega V m (Suc k) \rangle$  **by** *auto*  
**next**  
**case** (*Suc nat*)  
**then have**  $TS M2 M1 \Omega V m (Suc k) = TS M2 M1 \Omega V m k \cup C M2 M1 \Omega V m (Suc k)$   
**using**  $TS.simps(3)$  **by** *blast*  
**moreover have**  $tsN \cup cN = TS M2 M1 \Omega V m k \cup C M2 M1 \Omega V m (Suc k)$   
**using**  $\langle tsN = TS M2 M1 \Omega V m k \rangle \langle cN = C M2 M1 \Omega V m (Suc k) \rangle$  **by** *auto*  
**ultimately show** *?thesis*  
**by** *auto*  
**qed**  
**then have**  $tsN\text{-calc} : tsN \cup cN = TS M2 M1 \Omega V m iter$   
**using**  $\langle iter = Suc k \rangle$  **by** *presburger*  
  
**have**  $cN\text{-calc} : append\text{-set}$   
 $(cN -$

$\{xs' \in cN.$   
 $\neg \text{io-reduction-on } M1 \{xs'\} M2 \vee$   
 $(\forall io \in Lin \ M1 \{xs'\}.$   
 $\exists V'' \in N \ \text{io } M1 \ V.$   
 $\exists S1 \ \text{vs } xs.$   
 $\text{io} = \text{vs} @ xs \wedge$   
 $\text{mcp} (\text{vs} @ xs) \ V'' \ \text{vs} \wedge$   
 $S1 \subseteq \text{nodes } M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\forall s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP \ M2 \ s1 \ \text{vs } xs \ V''.$   
 $\forall io2 \in RP \ M2 \ s2 \ \text{vs } xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge$   
 $m < LB \ M2 \ M1 \ \text{vs } xs \ (\text{tsN} \cup V) \ S1 \ \Omega \ V''))$   
 $(\text{inputs } M2) -$   
 $(\text{tsN} \cup cN) =$   
 $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} + 1)$

**proof** –

**have** *append-set*

$(cN -$   
 $\{xs' \in cN.$   
 $\neg \text{io-reduction-on } M1 \{xs'\} M2 \vee$   
 $(\forall io \in Lin \ M1 \{xs'\}.$   
 $\exists V'' \in N \ \text{io } M1 \ V.$   
 $\exists S1 \ \text{vs } xs.$   
 $\text{io} = \text{vs} @ xs \wedge$   
 $\text{mcp} (\text{vs} @ xs) \ V'' \ \text{vs} \wedge$   
 $S1 \subseteq \text{nodes } M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\forall s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP \ M2 \ s1 \ \text{vs } xs \ V''.$   
 $\forall io2 \in RP \ M2 \ s2 \ \text{vs } xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge$   
 $m < LB \ M2 \ M1 \ \text{vs } xs \ (\text{tsN} \cup V) \ S1 \ \Omega \ V''))$   
 $(\text{inputs } M2) -$   
 $(\text{tsN} \cup cN) =$   
*append-set*  
 $((C \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}) -$   
 $(RM \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}))$   
 $(\text{inputs } M2) -$   
 $(TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter})$

**using**  $\langle cN = C \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} \rangle \langle \text{tsN} \cup cN = TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} \rangle$  *rmN-calc* **by** *presburger*

**moreover** *have* *append-set*

$((C \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}) -$   
 $(RM \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}))$   
 $(\text{inputs } M2) -$   
 $(TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}) = C \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} + 1)$

**proof** –

**have**  $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} + 1) = C \ M2 \ M1 \ \Omega \ V \ m \ ((\text{Suc } k) + 1)$

**using**  $\langle \text{iter} = \text{Suc } k \rangle$  **by** *presburger+*

**moreover** *have*  $(\text{Suc } k) + 1 = \text{Suc} (\text{Suc } k)$

**by** *simp*

**ultimately** *have*  $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} + 1) = C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc} (\text{Suc } k))$

**by** *presburger*

**have**  $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc} (\text{Suc } k))$   
 $= \text{append-set} (C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k) - RM \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k)) (\text{inputs } M2)$   
 $- TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k)$

**using**  $C.\text{simps}(\beta)$  *[of*  $M2 \ M1 \ \Omega \ V \ m \ k$  *]* **by** *linarith*

**show** *?thesis*

**using** *Suc-eq-plus1*

$\langle C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc} (\text{Suc } k))$   
 $= \text{append-set} (C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k) - RM \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k)) (\text{inputs } M2)$   
 $- TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k) \rangle$   
 $\langle \text{iter} = \text{Suc } k \rangle$

**by** *presburger*

qed

ultimately show ?thesis

by presburger

qed

have obs-calc : obs  $\cup$

$L_{in} M2$

(append-set

(cN -

{xs'  $\in$  cN.

$\neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee$

$(\forall io \in L_{in} M1 \{xs'\}.$

$\exists V'' \in N \ io \ M1 \ V.$

$\exists S1 \ vs \ xs.$

$io = vs \ @ \ xs \wedge$

$mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge$

$S1 \subseteq nodes \ M2 \wedge$

$(\forall s1 \in S1.$

$\forall s2 \in S1.$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''.$

$\forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge$

$m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''))$

(inputs M2) -

(tsN  $\cup$  cN)) =

$L_{in} M2$

(tsN  $\cup$  cN  $\cup$

(append-set

(cN -

{xs'  $\in$  cN.

$\neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee$

$(\forall io \in L_{in} M1 \{xs'\}.$

$\exists V'' \in N \ io \ M1 \ V.$

$\exists S1 \ vs \ xs.$

$io = vs \ @ \ xs \wedge$

$mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge$

$S1 \subseteq nodes \ M2 \wedge$

$(\forall s1 \in S1.$

$\forall s2 \in S1.$

$s1 \neq s2 \longrightarrow$

$(\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''.$

$\forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge$

$m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''))$

(inputs M2) -

(tsN  $\cup$  cN)))

proof -

have  $\bigwedge A. L_{in} M2 \ (tsN \cup cN \cup A) = obs \cup L_{in} M2 \ A$

by (metis (no-types) language-state-for-inputs-union precond)

then show ?thesis

by blast

qed

have obsI-calc : obsI  $\cup$

$L_{in} M1$

(append-set

(cN -

{xs'  $\in$  cN.

$\neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee$

$(\forall io \in L_{in} M1 \{xs'\}.$

$\exists V'' \in N \ io \ M1 \ V.$

$\exists S1 \ vs \ xs.$

$io = vs \ @ \ xs \wedge$

$mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge$

$$\begin{aligned}
& S1 \subseteq \text{nodes } M2 \wedge \\
& (\forall s1 \in S1. \\
& \quad \forall s2 \in S1. \\
& \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad (\forall io1 \in RP \ M2 \ s1 \ \text{vs } xs \ V''. \\
& \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ \text{vs } xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad m < LB \ M2 \ M1 \ \text{vs } xs \ (tsN \cup V) \ S1 \ \Omega \ V'')) \\
& (\text{inputs } M2) - \\
& (tsN \cup cN)) = \\
L_{in} \ M1 \\
& (tsN \cup cN \cup \\
& (\text{append-set} \\
& (cN - \\
& \quad \{xs' \in cN. \\
& \quad \neg L_{in} \ M1 \ \{xs'\} \subseteq L_{in} \ M2 \ \{xs'\} \vee \\
& \quad (\forall io \in L_{in} \ M1 \ \{xs'\}. \\
& \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \exists S1 \ \text{vs } xs. \\
& \quad \quad \quad io = \text{vs } @ \ xs \wedge \\
& \quad \quad \quad mcp \ (\text{vs } @ \ xs) \ V'' \ \text{vs } \wedge \\
& \quad \quad \quad S1 \subseteq \text{nodes } M2 \wedge \\
& \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ \text{vs } xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ \text{vs } xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad \quad \quad \quad m < LB \ M2 \ M1 \ \text{vs } xs \ (tsN \cup V) \ S1 \ \Omega \ V'')) \\
& \quad \quad \quad \quad (\text{inputs } M2) - \\
& \quad \quad \quad \quad (tsN \cup cN)))
\end{aligned}$$

**proof** –

**have**  $\bigwedge A. L_{in} \ M1 \ (tsN \cup cN \cup A) = \text{obsI} \cup L_{in} \ M1 \ A$   
**by** (*metis* (*no-types*) *language-state-for-inputs-union precond*)  
**then show** *?thesis*  
**by** *blast*  
**qed**

**have** *obs<sub>Ω</sub>-calc* : *obs<sub>Ω</sub>*  $\cup$

$$\begin{aligned}
& (\bigcup_{io \in L_{in} \ M2} \\
& \quad (\text{append-set} \\
& \quad (cN - \\
& \quad \quad \{xs' \in cN. \\
& \quad \quad \neg L_{in} \ M1 \ \{xs'\} \subseteq L_{in} \ M2 \ \{xs'\} \vee \\
& \quad \quad (\forall io \in L_{in} \ M1 \ \{xs'\}. \\
& \quad \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \quad \exists S1 \ \text{vs } xs. \\
& \quad \quad \quad \quad io = \text{vs } @ \ xs \wedge \\
& \quad \quad \quad \quad mcp \ (\text{vs } @ \ xs) \ V'' \ \text{vs } \wedge \\
& \quad \quad \quad \quad S1 \subseteq \text{nodes } M2 \wedge \\
& \quad \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ \text{vs } xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ \text{vs } xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad \quad \quad \quad \quad m < LB \ M2 \ M1 \ \text{vs } xs \ (tsN \cup V) \ S1 \ \Omega \ V'')) \\
& \quad \quad \quad \quad (\text{inputs } M2) - \\
& \quad \quad \quad \quad (tsN \cup cN)). \\
& \quad \{io\} \times B \ M2 \ io \ \Omega) = \\
& (\bigcup_{io \in L_{in} \ M2} \\
& \quad (tsN \cup cN \cup \\
& \quad (\text{append-set} \\
& \quad (cN - \\
& \quad \quad \{xs' \in cN. \\
& \quad \quad \neg L_{in} \ M1 \ \{xs'\} \subseteq L_{in} \ M2 \ \{xs'\} \vee \\
& \quad \quad (\forall io \in L_{in} \ M1 \ \{xs'\}. \\
& \quad \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \quad \exists S1 \ \text{vs } xs.
\end{aligned}$$

$io = vs @ xs \wedge$   
 $mcp (vs @ xs) V'' vs \wedge$   
 $S1 \subseteq nodes M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\quad \forall s2 \in S1.$   
 $\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad (\forall io1 \in RP M2 s1 vs xs V''.$   
 $\quad\quad\quad \forall io2 \in RP M2 s2 vs xs V''. B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $\quad m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V''))$   
 $(inputs M2) -$   
 $(tsN \cup cN)).$   
 $\{io\} \times B M2 io \Omega)$   
**using**  $\langle obs = L_{in} M2 (tsN \cup cN) \rangle$   
 $\langle obs_{\Omega} = (\bigcup_{io \in L_{in} M2} (tsN \cup cN). \{io\} \times B M2 io \Omega) \rangle$   
 $obs-calc$   
**by blast**

**have**  $obsI_{\Omega-calc} : obsI_{\Omega} \cup$   
 $(\bigcup_{io \in L_{in} M1}$   
 $(append-set$   
 $(cN -$   
 $\{xs' \in cN.$   
 $\neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee$   
 $(\forall io \in L_{in} M1 \{xs'\}.$   
 $\quad \exists V'' \in N io M1 V.$   
 $\quad \exists S1 vs xs.$   
 $\quad\quad io = vs @ xs \wedge$   
 $\quad\quad mcp (vs @ xs) V'' vs \wedge$   
 $\quad\quad S1 \subseteq nodes M2 \wedge$   
 $\quad\quad (\forall s1 \in S1.$   
 $\quad\quad\quad \forall s2 \in S1.$   
 $\quad\quad\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad\quad\quad (\forall io1 \in RP M2 s1 vs xs V''.$   
 $\quad\quad\quad\quad\quad \forall io2 \in RP M2 s2 vs xs V''. B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $\quad\quad\quad m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V''))$   
 $(inputs M2) -$   
 $(tsN \cup cN)).$   
 $\{io\} \times B M1 io \Omega) =$   
 $(\bigcup_{io \in L_{in} M1}$   
 $(tsN \cup cN \cup$   
 $(append-set$   
 $(cN -$   
 $\{xs' \in cN.$   
 $\neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee$   
 $(\forall io \in L_{in} M1 \{xs'\}.$   
 $\quad \exists V'' \in N io M1 V.$   
 $\quad \exists S1 vs xs.$   
 $\quad\quad io = vs @ xs \wedge$   
 $\quad\quad mcp (vs @ xs) V'' vs \wedge$   
 $\quad\quad S1 \subseteq nodes M2 \wedge$   
 $\quad\quad (\forall s1 \in S1.$   
 $\quad\quad\quad \forall s2 \in S1.$   
 $\quad\quad\quad\quad s1 \neq s2 \longrightarrow$   
 $\quad\quad\quad\quad (\forall io1 \in RP M2 s1 vs xs V''.$   
 $\quad\quad\quad\quad\quad \forall io2 \in RP M2 s2 vs xs V''. B M1 io1 \Omega \neq B M1 io2 \Omega)) \wedge$   
 $\quad\quad\quad m < LB M2 M1 vs xs (tsN \cup V) S1 \Omega V''))$   
 $(inputs M2) -$   
 $(tsN \cup cN)).$   
 $\{io\} \times B M1 io \Omega)$   
**using**  $\langle obsI = L_{in} M1 (tsN \cup cN) \rangle$   
 $\langle obsI_{\Omega} = (\bigcup_{io \in L_{in} M1} (tsN \cup cN). \{io\} \times B M1 io \Omega) \rangle$   
 $obsI-calc$   
**by blast**

**have**  $0 < iter + 1$   
**using**  $\langle 0 < iter \rangle$  **by** *simp*  
**have**  $tsN \cup cN = TS\ M2\ M1\ \Omega\ V\ m\ (iter + 1 - 1)$   
**using** *tsN-calc* **by** *simp*

**from**  $\langle 0 < iter + 1 \rangle$   
 $\langle tsN \cup cN = TS\ M2\ M1\ \Omega\ V\ m\ (iter + 1 - 1) \rangle$   
*cN-calc*  
*rmN-calc'*  
*obs-calc*  
*obsI-calc*  
*obs $\Omega$ -calc*  
*obsI $\Omega$ -calc*  
 $\langle OFSM\ M1 \rangle$   
 $\langle OFSM\ M2 \rangle$   
 $\langle asc\text{-}fault\text{-}domain\ M2\ M1\ m \rangle$   
 $\langle test\text{-}tools\ M2\ M1\ FAIL\ PM\ V\ \Omega \rangle$

**show**  $0 < iter + 1 \wedge$   
 $tsN \cup cN = TS\ M2\ M1\ \Omega\ V\ m\ (iter + 1 - 1) \wedge$   
*append-set*  
 $(cN -$   
 $\{xs' \in cN.$   
 $\neg L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\} \vee$   
 $(\forall io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists V'' \in N\ io\ M1\ V.$   
 $\exists S1\ vs\ xs.$   
 $io = vs @ xs \wedge$   
 $mcp\ (vs @ xs)\ V''\ vs \wedge$   
 $S1 \subseteq nodes\ M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\forall s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP\ M2\ s1\ vs\ xs\ V''.$   
 $\forall io2 \in RP\ M2\ s2\ vs\ xs\ V''.\ B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega)) \wedge$   
 $m < LB\ M2\ M1\ vs\ xs\ (tsN \cup V)\ S1\ \Omega\ V'')) \wedge$   
 $(inputs\ M2) -$   
 $(tsN \cup cN) =$   
 $C\ M2\ M1\ \Omega\ V\ m\ (iter + 1) \wedge$   
 $\{xs' \in cN.$   
 $\neg L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\} \vee$   
 $(\forall io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists V'' \in N\ io\ M1\ V.$   
 $\exists S1\ vs\ xs.$   
 $io = vs @ xs \wedge$   
 $mcp\ (vs @ xs)\ V''\ vs \wedge$   
 $S1 \subseteq nodes\ M2 \wedge$   
 $(\forall s1 \in S1.$   
 $\forall s2 \in S1.$   
 $s1 \neq s2 \longrightarrow$   
 $(\forall io1 \in RP\ M2\ s1\ vs\ xs\ V''.\ \forall io2 \in RP\ M2\ s2\ vs\ xs\ V''.$   
 $B\ M1\ io1\ \Omega \neq B\ M1\ io2\ \Omega)) \wedge$   
 $m < LB\ M2\ M1\ vs\ xs\ (tsN \cup V)\ S1\ \Omega\ V'')) =$   
 $RM\ M2\ M1\ \Omega\ V\ m\ (iter + 1 - 1) \wedge$   
 $obs \cup$   
 $L_{in}\ M2$   
*append-set*  
 $(cN -$   
 $\{xs' \in cN.$   
 $\neg L_{in}\ M1\ \{xs'\} \subseteq L_{in}\ M2\ \{xs'\} \vee$   
 $(\forall io \in L_{in}\ M1\ \{xs'\}.$   
 $\exists V'' \in N\ io\ M1\ V.$   
 $\exists S1\ vs\ xs.$   
 $io = vs @ xs \wedge$   
 $mcp\ (vs @ xs)\ V''\ vs \wedge$



$$\begin{aligned}
& m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''\}) \\
& (inputs \ M2) - \\
& (tsN \cup cN))) \wedge \\
& obs_{\Omega} \cup \\
& (\bigcup_{io \in Lin \ M2} \\
& \quad (append-set \\
& \quad (cN - \\
& \quad \{xs' \in cN. \\
& \quad \neg Lin \ M1 \ \{xs'\} \subseteq Lin \ M2 \ \{xs'\} \vee \\
& \quad (\forall io \in Lin \ M1 \ \{xs'\}. \\
& \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \exists S1 \ vs \ xs. \\
& \quad \quad \quad io = vs \ @ \ xs \wedge \\
& \quad \quad \quad mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge \\
& \quad \quad \quad S1 \subseteq nodes \ M2 \wedge \\
& \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad \quad \quad m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''\})) \\
& (inputs \ M2) - \\
& (tsN \cup cN)). \\
& \{io\} \times B \ M2 \ io \ \Omega) = \\
& (\bigcup_{io \in Lin \ M2} \\
& \quad (tsN \cup cN \cup \\
& \quad (append-set \\
& \quad (cN - \\
& \quad \{xs' \in cN. \\
& \quad \neg Lin \ M1 \ \{xs'\} \subseteq Lin \ M2 \ \{xs'\} \vee \\
& \quad (\forall io \in Lin \ M1 \ \{xs'\}. \\
& \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \exists S1 \ vs \ xs. \\
& \quad \quad \quad io = vs \ @ \ xs \wedge \\
& \quad \quad \quad mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge \\
& \quad \quad \quad S1 \subseteq nodes \ M2 \wedge \\
& \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad \quad \quad m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''\})) \\
& (inputs \ M2) - \\
& (tsN \cup cN))). \\
& \{io\} \times B \ M2 \ io \ \Omega) \wedge \\
& obs_{\Omega} \cup \\
& (\bigcup_{io \in Lin \ M1} \\
& \quad (append-set \\
& \quad (cN - \\
& \quad \{xs' \in cN. \\
& \quad \neg Lin \ M1 \ \{xs'\} \subseteq Lin \ M2 \ \{xs'\} \vee \\
& \quad (\forall io \in Lin \ M1 \ \{xs'\}. \\
& \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \exists S1 \ vs \ xs. \\
& \quad \quad \quad io = vs \ @ \ xs \wedge \\
& \quad \quad \quad mcp \ (vs \ @ \ xs) \ V'' \ vs \wedge \\
& \quad \quad \quad S1 \subseteq nodes \ M2 \wedge \\
& \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad s1 \neq s2 \longrightarrow \\
& \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq B \ M1 \ io2 \ \Omega)) \wedge \\
& \quad \quad \quad m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V''\})) \\
& (inputs \ M2) - \\
& (tsN \cup cN)). \\
& \{io\} \times B \ M1 \ io \ \Omega) =
\end{aligned}$$

$$\begin{aligned}
& (\bigcup_{io \in L_{in} M1} \\
& \quad (tsN \cup cN \cup \\
& \quad \quad (append-set \\
& \quad \quad \quad (cN - \\
& \quad \quad \quad \quad \{xs' \in cN. \\
& \quad \quad \quad \quad \quad \neg L_{in} M1 \{xs'\} \subseteq L_{in} M2 \{xs'\} \vee \\
& \quad \quad \quad \quad \quad (\forall io \in L_{in} M1 \{xs'\}. \\
& \quad \quad \quad \quad \quad \quad \exists V'' \in N \ io \ M1 \ V. \\
& \quad \quad \quad \quad \quad \quad \quad \exists S1 \ vs \ xs. \\
& \quad \quad \quad \quad \quad \quad \quad \quad io = vs \ @ \ xs \ \wedge \\
& \quad \quad \quad \quad \quad \quad \quad \quad mcp \ (vs \ @ \ xs) \ V'' \ vs \ \wedge \\
& \quad \quad \quad \quad \quad \quad \quad \quad S1 \subseteq nodes \ M2 \ \wedge \\
& \quad \quad \quad \quad \quad \quad \quad \quad (\forall s1 \in S1. \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \forall s2 \in S1. \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad s1 \neq s2 \ \longrightarrow \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (\forall io1 \in RP \ M2 \ s1 \ vs \ xs \ V''. \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \forall io2 \in RP \ M2 \ s2 \ vs \ xs \ V''. \ B \ M1 \ io1 \ \Omega \neq \ B \ M1 \ io2 \ \Omega)) \ \wedge \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad m < LB \ M2 \ M1 \ vs \ xs \ (tsN \cup V) \ S1 \ \Omega \ V'')) \\
& \quad \quad (inputs \ M2) - \\
& \quad \quad (tsN \cup cN))). \\
& \quad \{io\} \times B \ M1 \ io \ \Omega) \ \wedge \\
& \quad OFSM \ M1 \ \wedge \ OFSM \ M2 \ \wedge \ asc-fault-domain \ M2 \ M1 \ m \ \wedge \ test-tools \ M2 \ M1 \ FAIL \ PM \ V \ \Omega \\
& \text{by } \mathit{linarith}
\end{aligned}$$

**next**

**fix**  $tsN \ cN \ rmN \ obs \ obsI \ obs_{\Omega} \ obsI_{\Omega} \ iter \ isReduction$

**assume**  $precond : (0 < iter \ \wedge$

$$\begin{aligned}
& \quad tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ (iter - 1) \ \wedge \\
& \quad cN = C \ M2 \ M1 \ \Omega \ V \ m \ iter \ \wedge \\
& \quad rmN = RM \ M2 \ M1 \ \Omega \ V \ m \ (iter - 1) \ \wedge \\
& \quad obs = L_{in} \ M2 \ (tsN \cup cN) \ \wedge \\
& \quad obsI = L_{in} \ M1 \ (tsN \cup cN) \ \wedge \\
& \quad obs_{\Omega} = (\bigcup_{io \in L_{in} \ M2 \ (tsN \cup cN)}. \{io\} \times B \ M2 \ io \ \Omega) \ \wedge \\
& \quad obsI_{\Omega} = (\bigcup_{io \in L_{in} \ M1 \ (tsN \cup cN)}. \{io\} \times B \ M1 \ io \ \Omega) \ \wedge \\
& \quad OFSM \ M1 \ \wedge \ OFSM \ M2 \ \wedge \ asc-fault-domain \ M2 \ M1 \ m \ \wedge \ test-tools \ M2 \ M1 \ FAIL \ PM \ V \ \Omega) \ \wedge \\
& \quad \neg (cN \neq \{\}) \ \wedge \ obsI \subseteq obs \ \wedge \ obsI_{\Omega} \subseteq obs_{\Omega})
\end{aligned}$$

**then have**  $0 < iter$

$OFSM \ M1$

$OFSM \ M2$

$asc-fault-domain \ M2 \ M1 \ m$

$test-tools \ M2 \ M1 \ FAIL \ PM \ V \ \Omega$

$cN = \{\} \vee \neg obsI \subseteq obs \vee \neg obsI_{\Omega} \subseteq obs_{\Omega}$

$tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ (iter-1)$

$cN = C \ M2 \ M1 \ \Omega \ V \ m \ iter$

$rmN = RM \ M2 \ M1 \ \Omega \ V \ m \ (iter-1)$

$obs = L_{in} \ M2 \ (tsN \cup cN)$

$obsI = L_{in} \ M1 \ (tsN \cup cN)$

$obs_{\Omega} = (\bigcup_{io \in L_{in} \ M2 \ (tsN \cup cN)}. \{io\} \times B \ M2 \ io \ \Omega)$

$obsI_{\Omega} = (\bigcup_{io \in L_{in} \ M1 \ (tsN \cup cN)}. \{io\} \times B \ M1 \ io \ \Omega)$

**by**  $\mathit{linarith+}$

**show**  $(obsI \subseteq obs \ \wedge \ obsI_{\Omega} \subseteq obs_{\Omega}) = M1 \preceq M2$

**proof**  $(cases \ cN = \{\})$

**case**  $True$

**then have**  $C \ M2 \ M1 \ \Omega \ V \ m \ iter = \{\}$

**using**  $\langle cN = C \ M2 \ M1 \ \Omega \ V \ m \ iter \rangle$  **by**  $\mathit{auto}$

**have**  $is-det-state-cover \ M2 \ V$

**using**  $\langle test-tools \ M2 \ M1 \ FAIL \ PM \ V \ \Omega \rangle$  **by**  $\mathit{auto}$

**then have**  $\square \in V$

**using**  $det-state-cover-initial[of \ M2 \ V]$  **by**  $\mathit{simp}$

**then have**  $V \neq \{\}$

**by**  $\mathit{blast}$

**have**  $Suc \ 0 < iter$

**proof**  $(rule \ ccontr)$

**assume**  $\neg \text{Suc } 0 < \text{iter}$   
**then have**  $\text{iter} = \text{Suc } 0$   
**using**  $\langle 0 < \text{iter} \rangle$  **by** *auto*  
**then have**  $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } 0) = \{\}$   
**using**  $\langle C \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} = \{\} \rangle$  **by** *auto*  
**moreover have**  $C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } 0) = V$   
**by** *auto*  
**ultimately show** *False*  
**using**  $\langle V \neq \{\} \rangle$  **by** *blast*  
**qed**

**obtain**  $k$  **where**  $\text{iter} = \text{Suc } k$   
**using**  $\text{gr0-implies-Suc}[OF \ \langle 0 < \text{iter} \rangle]$  **by** *blast*  
**then have**  $\text{Suc } 0 < \text{Suc } k$   
**using**  $\langle \text{Suc } 0 < \text{iter} \rangle$  **by** *auto*  
**then have**  $0 < k$   
**by** *simp*  
**then obtain**  $k'$  **where**  $k = \text{Suc } k'$   
**using**  $\text{gr0-implies-Suc}$  **by** *blast*  
**have**  $\text{iter} = \text{Suc } (\text{Suc } k')$   
**using**  $\langle \text{iter} = \text{Suc } k \rangle \ \langle k = \text{Suc } k' \rangle$  **by** *simp*

**have**  $TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } (\text{Suc } k')) = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k') \cup C \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } (\text{Suc } k'))$   
**using**  $TS.\text{simps}(3)[of \ M2 \ M1 \ \Omega \ V \ m \ k']$  **by** *blast*  
**then have**  $TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{Suc } k')$   
**using**  $\text{True} \ \langle cN = C \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} \rangle \ \langle \text{iter} = \text{Suc } (\text{Suc } k') \rangle$  **by** *blast*  
**moreover have**  $\text{Suc } k' = \text{iter} - 1$   
**using**  $\langle \text{iter} = \text{Suc } (\text{Suc } k') \rangle$  **by** *presburger*  
**ultimately have**  $TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1)$   
**by** *auto*  
**then have**  $tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter}$   
**using**  $\langle tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1) \rangle$  **by** *simp*

**then have**  $TS \ M2 \ M1 \ \Omega \ V \ m \ \text{iter} = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1)$   
**using**  $\langle tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1) \rangle$  **by** *auto*  
**then have**  $\text{final-iteration } M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1)$   
**using**  $\langle 0 < \text{iter} \rangle$  **by** *auto*

**have**  $M1 \preceq M2 = \text{atc-io-reduction-on-sets } M1 \ tsN \ \Omega \ M2$   
**using**  $\text{asc-main-theorem}[OF \ \langle OFSM \ M1 \rangle \ \langle OFSM \ M2 \rangle$   
 $\langle \text{asc-fault-domain } M2 \ M1 \ m \rangle$   
 $\langle \text{test-tools } M2 \ M1 \ FAIL \ PM \ V \ \Omega \rangle$   
 $\langle \text{final-iteration } M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1) \rangle]$   
**using**  $\langle tsN = TS \ M2 \ M1 \ \Omega \ V \ m \ (\text{iter} - 1) \rangle$   
**by** *blast*  
**moreover have**  $tsN \cup cN = tsN$   
**using**  $\langle cN = \{\} \rangle$  **by** *blast*  
**ultimately have**  $M1 \preceq M2 = \text{atc-io-reduction-on-sets } M1 \ (tsN \cup cN) \ \Omega \ M2$   
**by** *presburger*

**have**  $\text{obs}I \subseteq \text{obs} \equiv L_{in} \ M1 \ (tsN \cup cN) \subseteq L_{in} \ M2 \ (tsN \cup cN)$   
**by**  $(\text{simp add: } \langle \text{obs} = L_{in} \ M2 \ (tsN \cup cN) \rangle \ \langle \text{obs}I = L_{in} \ M1 \ (tsN \cup cN) \rangle)$

**have**  $\text{obs}I_{\Omega} \subseteq \text{obs}_{\Omega} \equiv (\bigcup_{io \in L_{in} \ M1 \ (tsN \cup cN)}. \{io\} \times B \ M1 \ io \ \Omega)$   
 $\subseteq (\bigcup_{io \in L_{in} \ M2 \ (tsN \cup cN)}. \{io\} \times B \ M2 \ io \ \Omega)$   
**by**  $(\text{simp add: } \langle \text{obs}I_{\Omega} = (\bigcup_{io \in L_{in} \ M1 \ (tsN \cup cN)}. \{io\} \times B \ M1 \ io \ \Omega) \rangle$   
 $\langle \text{obs}_{\Omega} = (\bigcup_{io \in L_{in} \ M2 \ (tsN \cup cN)}. \{io\} \times B \ M2 \ io \ \Omega) \rangle)$

**have**  $(\text{obs}I \subseteq \text{obs} \wedge \text{obs}I_{\Omega} \subseteq \text{obs}_{\Omega}) = \text{atc-io-reduction-on-sets } M1 \ (tsN \cup cN) \ \Omega \ M2$   
**proof**

**assume**  $\text{obs}I \subseteq \text{obs} \wedge \text{obs}I_{\Omega} \subseteq \text{obs}_{\Omega}$   
**show**  $\text{atc-io-reduction-on-sets } M1 \ (tsN \cup cN) \ \Omega \ M2$   
**using**  $\text{atc-io-reduction-on-sets-from-obs}[of \ M1 \ tsN \cup cN \ M2 \ \Omega]$   
**using**  $\langle \text{obs}I \subseteq \text{obs} \wedge \text{obs}I_{\Omega} \subseteq \text{obs}_{\Omega} \rangle \ \langle \text{obs}I \subseteq \text{obs} \equiv L_{in} \ M1 \ (tsN \cup cN) \subseteq L_{in} \ M2 \ (tsN \cup cN) \rangle$   
 $\langle \text{obs}I_{\Omega} \subseteq \text{obs}_{\Omega} \equiv (\bigcup_{io \in L_{in} \ M1 \ (tsN \cup cN)}. \{io\} \times B \ M1 \ io \ \Omega) \rangle$

```

       $\subseteq (\bigcup_{io \in L_{in}} M2 (tsN \cup cN). \{io\} \times B M2 io \Omega)$ 
    by linarith
  next
  assume atc-io-reduction-on-sets  $M1 (tsN \cup cN) \Omega M2$ 
  show  $obsI \subseteq obs \wedge obsI_{\Omega} \subseteq obs_{\Omega}$ 
  using atc-io-reduction-on-sets-to-obs[of  $M1 \langle tsN \cup cN \rangle \Omega M2$ ]
     $\langle atc-io-reduction-on-sets M1 (tsN \cup cN) \Omega M2 \rangle$ 
     $\langle obsI \subseteq obs \equiv L_{in} M1 (tsN \cup cN) \subseteq L_{in} M2 (tsN \cup cN) \rangle$ 
     $\langle obsI_{\Omega} \subseteq obs_{\Omega} \equiv (\bigcup_{io \in L_{in}} M1 (tsN \cup cN). \{io\} \times B M1 io \Omega) \subseteq (\bigcup_{io \in L_{in}} M2 (tsN \cup cN). \{io\} \times B M2 io \Omega) \rangle$ 
  by blast
  qed
  then show ?thesis
  using  $\langle M1 \preceq M2 = atc-io-reduction-on-sets M1 (tsN \cup cN) \Omega M2 \rangle$  by linarith
next
case False

  then have  $\neg obsI \subseteq obs \vee \neg obsI_{\Omega} \subseteq obs_{\Omega}$ 
  using  $\langle cN = \{\} \vee \neg obsI \subseteq obs \vee \neg obsI_{\Omega} \subseteq obs_{\Omega} \rangle$  by auto

  have  $\neg atc-io-reduction-on-sets M1 (tsN \cup cN) \Omega M2$ 
  using atc-io-reduction-on-sets-to-obs[of  $M1 tsN \cup cN \Omega M2$ ]
     $\langle \neg obsI \subseteq obs \vee \neg obsI_{\Omega} \subseteq obs_{\Omega} \rangle$  precond
  by fastforce

  have  $\neg M1 \preceq M2$ 
  proof
  assume  $M1 \preceq M2$ 
  have atc-io-reduction-on-sets  $M1 (tsN \cup cN) \Omega M2$ 
  using asc-soundness[OF  $\langle OFSM M1 \rangle \langle OFSM M2 \rangle$ ]  $\langle M1 \preceq M2 \rangle$  by blast
  then show False
  using  $\langle \neg atc-io-reduction-on-sets M1 (tsN \cup cN) \Omega M2 \rangle$  by blast
  qed

  then show ?thesis
  using  $\langle \neg obsI \subseteq obs \vee \neg obsI_{\Omega} \subseteq obs_{\Omega} \rangle$  by blast

  qed
qed

end
theory ASC-Example
  imports ASC-Hoare
begin

```

## 8 Example product machines and properties

This section provides example FSMs and shows that the assumptions on the inputs of the adaptive state counting algorithm are not vacuous.

### 8.1 Constructing FSMs from transition relations

This subsection provides a function to more easily create FSMs, only requiring a set of transition-tuples and an initial state.

```

fun from-rel :: ('state  $\times$  ('in  $\times$  'out)  $\times$  'state) set  $\Rightarrow$  'state  $\Rightarrow$  ('in, 'out, 'state) FSM where
from-rel rel q0 = ( $\downarrow succ = \lambda io p . \{ q . (p, io, q) \in rel \},$ 
  inputs = image (fst  $\circ$  fst  $\circ$  snd) rel,
  outputs = image (snd  $\circ$  fst  $\circ$  snd) rel,
  initial = q0  $\downarrow$ )

```

```

lemma nodes-from-rel : nodes (from-rel rel q0)  $\subseteq$  insert q0 (image (snd  $\circ$  snd) rel)
  (is nodes ?M  $\subseteq$  insert q0 (image (snd  $\circ$  snd) rel))
proof -
  have  $\bigwedge q$  io p . q  $\in$  succ ?M io p  $\implies$  q  $\in$  image (snd  $\circ$  snd) rel
    by force
  have  $\bigwedge q$  . q  $\in$  nodes ?M  $\implies$  q = q0  $\vee$  q  $\in$  image (snd  $\circ$  snd) rel
proof -
  fix q assume q  $\in$  nodes ?M
  then show q = q0  $\vee$  q  $\in$  image (snd  $\circ$  snd) rel
  proof (cases rule: FSM.nodes.cases)
    case initial
    then show ?thesis by auto
  next
    case (execute p a)
    then show ?thesis
      using  $\langle \bigwedge q$  io p . q  $\in$  succ ?M io p  $\implies$  q  $\in$  image (snd  $\circ$  snd) rel by blast
  qed
qed
then show nodes ?M  $\subseteq$  insert q0 (image (snd  $\circ$  snd) rel)
  by blast
qed

```

```

fun well-formed-rel :: ('state  $\times$  ('in  $\times$  'out)  $\times$  'state) set  $\implies$  bool where
  well-formed-rel rel = (finite rel
     $\wedge$  ( $\forall s1$  x y . (x  $\notin$  image (fst  $\circ$  fst  $\circ$  snd) rel
       $\vee$  y  $\notin$  image (snd  $\circ$  fst  $\circ$  snd) rel)
       $\longrightarrow$   $\neg$ ( $\exists s2$  . (s1,(x,y),s2)  $\in$  rel))
     $\wedge$  rel  $\neq$  {})

```

```

lemma well-formed-from-rel :
  assumes well-formed-rel rel
  shows well-formed (from-rel rel q0) (is well-formed ?M)
proof -
  have nodes ?M  $\subseteq$  insert q0 (image (snd  $\circ$  snd) rel)
    using nodes-from-rel[of rel q0] by auto
  moreover have finite (insert q0 (image (snd  $\circ$  snd) rel))
    using assms by auto
  ultimately have finite (nodes ?M)
    by (simp add: Finite-Set.finite-subset)
  moreover have finite (inputs ?M) finite (outputs ?M)
    using assms by auto
  ultimately have finite-FSM ?M
    by auto

  moreover have inputs ?M  $\neq$  {}
    using assms by auto
  moreover have outputs ?M  $\neq$  {}
    using assms by auto
  moreover have  $\bigwedge s1$  x y . (x  $\notin$  inputs ?M  $\vee$  y  $\notin$  outputs ?M)  $\longrightarrow$  succ ?M (x,y) s1 = {}
    using assms by auto

  ultimately show ?thesis
    by auto
qed

```

```

fun completely-specified-rel-over :: ('state  $\times$  ('in  $\times$  'out)  $\times$  'state) set  $\implies$  'state set  $\implies$  bool
  where
  completely-specified-rel-over rel nods = ( $\forall s1$   $\in$  nods .
     $\forall x$   $\in$  image (fst  $\circ$  fst  $\circ$  snd) rel .
     $\exists y$   $\in$  image (snd  $\circ$  fst  $\circ$  snd) rel .
     $\exists s2$  . (s1,(x,y),s2)  $\in$  rel)

```

```

lemma completely-specified-from-rel :
  assumes completely-specified-rel-over rel (nodes ((from-rel rel q0)))
  shows completely-specified (from-rel rel q0) (is completely-specified ?M)
  unfolding completely-specified.simps
proof
  fix s1 assume s1 ∈ nodes (from-rel rel q0)
  show ∀ x ∈ inputs ?M. ∃ y ∈ outputs ?M. ∃ s2. s2 ∈ succ ?M (x, y) s1
  proof
    fix x assume x ∈ inputs (from-rel rel q0)
    then have x ∈ image (fst ∘ fst ∘ snd) rel
      using assms by auto

    obtain y s2 where y ∈ image (snd ∘ fst ∘ snd) rel (s1, (x, y), s2) ∈ rel
      using assms ⟨s1 ∈ nodes (from-rel rel q0)⟩ ⟨x ∈ image (fst ∘ fst ∘ snd) rel⟩
      by (meson completely-specified-rel-over.elims(2))

    then have y ∈ outputs (from-rel rel q0) s2 ∈ succ (from-rel rel q0) (x, y) s1
      by auto

    then show ∃ y ∈ outputs (from-rel rel q0). ∃ s2. s2 ∈ succ (from-rel rel q0) (x, y) s1
      by blast
  qed
qed

```

```

fun observable-rel :: ('state × ('in × 'out) × 'state) set ⇒ bool where
  observable-rel rel = (∀ io s1 . { s2 . (s1, io, s2) ∈ rel } = {}
    ∨ (∃ s2 . { s2' . (s1, io, s2') ∈ rel } = {s2}))

```

```

lemma observable-from-rel :
  assumes observable-rel rel
  shows observable (from-rel rel q0) (is observable ?M)
proof –
  have ⋀ io s1 . { s2 . (s1, io, s2) ∈ rel } = succ ?M io s1
    by auto
  then show ?thesis using assms by auto
qed

```

```

abbreviation OFSM-rel rel q0 ≡ well-formed-rel rel
  ∧ completely-specified-rel-over rel (nodes (from-rel rel q0))
  ∧ observable-rel rel

```

```

lemma OFMS-from-rel :
  assumes OFSM-rel rel q0
  shows OFMS (from-rel rel q0)
  by (metis assms completely-specified-from-rel observable-from-rel well-formed-from-rel)

```

## 8.2 Example FSMs and properties

```

abbreviation MS-rel :: (nat × (nat × nat) × nat) set ≡ {(0, (0, 0), 1), (0, (0, 1), 1), (1, (0, 2), 1)}
abbreviation MS :: (nat, nat, nat) FSM ≡ from-rel MS-rel 0

```

```

abbreviation MI-rel :: (nat × (nat × nat) × nat) set ≡ {(0, (0, 0), 1), (0, (0, 1), 1), (1, (0, 2), 0)}
abbreviation MI :: (nat, nat, nat) FSM ≡ from-rel MI-rel 0

```

```

lemma example-nodes :
  nodes MS = {0, 1} nodes MI = {0, 1}
proof –

```

**have**  $0 \in \text{nodes } M_S$  **by** *auto*  
**have**  $1 \in \text{succ } M_S (0,0) 0$  **by** *auto*  
**have**  $1 \in \text{nodes } M_S$   
**by** (*meson*  $\langle 0 \in \text{nodes } M_S \rangle \langle 1 \in \text{succ } M_S (0, 0) 0 \rangle \text{succ-nodes}$ )

**have**  $\{0,1\} \subseteq \text{nodes } M_S$   
**using**  $\langle 0 \in \text{nodes } M_S \rangle \langle 1 \in \text{nodes } M_S \rangle$  **by** *auto*  
**moreover have**  $\text{nodes } M_S \subseteq \{0,1\}$   
**using** *nodes-from-rel*[of  $M_S\text{-rel } 0$ ] **by** *auto*  
**ultimately show**  $\text{nodes } M_S = \{0,1\}$   
**by** *blast*

**next**

**have**  $0 \in \text{nodes } M_I$  **by** *auto*  
**have**  $1 \in \text{succ } M_I (0,0) 0$  **by** *auto*  
**have**  $1 \in \text{nodes } M_I$   
**by** (*meson*  $\langle 0 \in \text{nodes } M_I \rangle \langle 1 \in \text{succ } M_I (0, 0) 0 \rangle \text{succ-nodes}$ )

**have**  $\{0,1\} \subseteq \text{nodes } M_I$   
**using**  $\langle 0 \in \text{nodes } M_I \rangle \langle 1 \in \text{nodes } M_I \rangle$  **by** *auto*  
**moreover have**  $\text{nodes } M_I \subseteq \{0,1\}$   
**using** *nodes-from-rel*[of  $M_I\text{-rel } 0$ ] **by** *auto*  
**ultimately show**  $\text{nodes } M_I = \{0,1\}$   
**by** *blast*

**qed**

**lemma** *example-OFSM* :

*OFSM*  $M_S$  *OFSM*  $M_I$

**proof** –

**have** *well-formed-rel*  $M_S\text{-rel}$   
**unfolding** *well-formed-rel.simps* **by** *auto*

**moreover have** *completely-specified-rel-over*  $M_S\text{-rel}$  (*nodes* (*from-rel*  $M_S\text{-rel } 0$ ))  
**unfolding** *completely-specified-rel-over.simps*

**proof**

**fix**  $s1$  **assume**  $(s1::\text{nat}) \in \text{nodes } (\text{from-rel } M_S\text{-rel } 0)$   
**then have**  $s1 \in (\text{insert } 0 (\text{image } (\text{snd} \circ \text{snd}) M_S\text{-rel}))$   
**using** *nodes-from-rel*[of  $M_S\text{-rel } 0$ ] **by** *blast*  
**moreover have** *completely-specified-rel-over*  $M_S\text{-rel}$  (*insert*  $0$  (*image* (*snd*  $\circ$  *snd*)  $M_S\text{-rel}$ ))  
**unfolding** *completely-specified-rel-over.simps* **by** *auto*  
**ultimately show**  $\forall x \in (\text{fst} \circ \text{fst} \circ \text{snd}) \text{ ' } M_S\text{-rel}.$   
 $\exists y \in (\text{snd} \circ \text{fst} \circ \text{snd}) \text{ ' } M_S\text{-rel}. \exists s2. (s1, (x, y), s2) \in M_S\text{-rel}$   
**by** *simp*

**qed**

**moreover have** *observable-rel*  $M_S\text{-rel}$   
**by** *auto*

**ultimately have** *OFSM-rel*  $M_S\text{-rel } 0$   
**by** *auto*

**then show** *OFSM*  $M_S$   
**using** *OFSM-from-rel*[of  $M_S\text{-rel } 0$ ] **by** *linarith*

**next**

**have** *well-formed-rel*  $M_I\text{-rel}$   
**unfolding** *well-formed-rel.simps* **by** *auto*

**moreover have** *completely-specified-rel-over*  $M_I\text{-rel}$  (*nodes* (*from-rel*  $M_I\text{-rel } 0$ ))  
**unfolding** *completely-specified-rel-over.simps*

**proof**

**fix**  $s1$  **assume**  $(s1::\text{nat}) \in \text{nodes } (\text{from-rel } M_I\text{-rel } 0)$   
**then have**  $s1 \in (\text{insert } 0 (\text{image } (\text{snd} \circ \text{snd}) M_I\text{-rel}))$   
**using** *nodes-from-rel*[of  $M_I\text{-rel } 0$ ] **by** *blast*  
**have** *completely-specified-rel-over*  $M_I\text{-rel}$  (*insert*  $0$  (*image* (*snd*  $\circ$  *snd*)  $M_I\text{-rel}$ ))  
**unfolding** *completely-specified-rel-over.simps* **by** *auto*

**show**  $\forall x \in (fst \circ fst \circ snd) \text{ ' } M_I\text{-rel.}$   
 $\exists y \in (snd \circ fst \circ snd) \text{ ' } M_I\text{-rel. } \exists s2. (s1, (x, y), s2) \in M_I\text{-rel}$   
**by**  $(meson \langle \text{completely-specified-rel-over } M_I\text{-rel } (\text{insert } 0 \text{ } ((snd \circ snd) \text{ ' } M_I\text{-rel})) \rangle$   
 $\langle s1 \in \text{insert } 0 \text{ } ((snd \circ snd) \text{ ' } M_I\text{-rel}) \rangle \text{ completely-specified-rel-over.elims}(2))$   
**qed**

**moreover have** *observable-rel*  $M_I\text{-rel}$   
**by** *auto*

**ultimately have** *OFSM-rel*  $M_I\text{-rel } 0$   
**by** *auto*

**then show** *OFSM*  $M_I$   
**using** *OFMS-from-rel*[of  $M_I\text{-rel } 0$ ] **by** *linarith*  
**qed**

**lemma** *example-fault-domain* : *asc-fault-domain*  $M_S M_I 2$   
**proof** –  
**have** *inputs*  $M_S = \text{inputs } M_I$   
**by** *auto*  
**moreover have** *card*  $(\text{nodes } M_I) \leq 2$   
**using** *example-nodes*(2) **by** *auto*  
**ultimately show** *asc-fault-domain*  $M_S M_I 2$   
**by** *auto*  
**qed**

**abbreviation**  $FAIL_I :: (\text{nat} \times \text{nat}) \equiv (3, 3)$   
**abbreviation**  $PM_I :: (\text{nat}, \text{nat}, \text{nat} \times \text{nat}) \text{ FSM} \equiv ()$   
 $\text{succ} = (\lambda a (p1, p2) . (\text{if } (p1 \in \text{nodes } M_S \wedge p2 \in \text{nodes } M_I \wedge (\text{fst } a \in \text{inputs } M_S)$   
 $\wedge (\text{snd } a \in \text{outputs } M_S \cup \text{outputs } M_I))$   
 $\text{then } (\text{if } (\text{succ } M_S a p1 = \{\}) \wedge \text{succ } M_I a p2 \neq \{\})$   
 $\text{then } \{FAIL_I\}$   
 $\text{else } (\text{succ } M_S a p1 \times \text{succ } M_I a p2))$   
 $\text{else } \{\})$ ),  
 $\text{inputs} = \text{inputs } M_S,$   
 $\text{outputs} = \text{outputs } M_S \cup \text{outputs } M_I,$   
 $\text{initial} = (\text{initial } M_S, \text{initial } M_I)$   
 $)$

**lemma** *example-productF* : *productF*  $M_S M_I FAIL_I PM_I$   
**proof** –  
**have** *inputs*  $M_S = \text{inputs } M_I$   
**by** *auto*  
**moreover have** *fst*  $FAIL_I \notin \text{nodes } M_S$   
**using** *example-nodes*(1) **by** *auto*  
**moreover have** *snd*  $FAIL_I \notin \text{nodes } M_I$   
**using** *example-nodes*(2) **by** *auto*  
**ultimately show** *?thesis*  
**unfolding** *productF.simps* **by** *blast*  
**qed**

**abbreviation**  $V_I :: \text{nat list set} \equiv \{\[], [0]\}$

**lemma** *example-det-state-cover* : *is-det-state-cover*  $M_S V_I$   
**proof** –  
**have** *d-reaches*  $M_S (\text{initial } M_S) [] (\text{initial } M_S)$   
**by** *auto*  
**then have**  $\text{initial } M_S \in \text{d-reachable } M_S (\text{initial } M_S)$   
**unfolding** *d-reachable.simps* **by** *blast*

**have** *d-reached-by*  $M_S (\text{initial } M_S) [0] 1 [1] [0]$   
**proof**

```

show length [0] = length [0] ∧
length [0] = length [1] ∧ path MS (([0] || [0]) || [1]) (initial MS)
    ∧ target (([0] || [0]) || [1]) (initial MS) = 1
by auto

have ∧ys2 tr2.
    length [0] = length ys2
    ∧ length [0] = length tr2
    ∧ path MS (([0] || ys2) || tr2) (initial MS)
    → target (([0] || ys2) || tr2) (initial MS) = 1
proof
fix ys2 tr2 assume length [0] = length ys2 ∧ length [0] = length tr2
    ∧ path MS (([0] || ys2) || tr2) (initial MS)
then have length ys2 = 1 length tr2 = 1 path MS (([0] || ys2) || tr2) (initial MS)
by auto
moreover obtain y2 where ys2 = [y2]
using ⟨length ys2 = 1⟩
by (metis One-nat-def ⟨length [0] = length ys2 ∧ length [0] = length tr2
    ∧ path MS (([0] || ys2) || tr2) (initial MS)⟩ append.simps(1) append-butlast-last-id
    butlast-snoc length-butlast length-greater-0-conv list.size(3) nat.simps(3))
moreover obtain t2 where tr2 = [t2]
using ⟨length tr2 = 1⟩
by (metis One-nat-def ⟨length [0] = length ys2 ∧ length [0] = length tr2
    ∧ path MS (([0] || ys2) || tr2) (initial MS)⟩ append.simps(1) append-butlast-last-id
    butlast-snoc length-butlast length-greater-0-conv list.size(3) nat.simps(3))
ultimately have path MS [(0,y2),t2] (initial MS)
by auto
then have t2 ∈ succ MS (0,y2) (initial MS)
by auto
moreover have ∧ y . succ MS (0,y) (initial MS) ⊆ {1}
by auto
ultimately have t2 = 1
by blast

show target (([0] || ys2) || tr2) (initial MS) = 1
using ⟨ys2 = [y2]⟩ ⟨tr2 = [t2]⟩ ⟨t2 = 1⟩ by auto
qed
then show ∀ys2 tr2.
    length [0] = length ys2 ∧ length [0] = length tr2
    ∧ path MS (([0] || ys2) || tr2) (initial MS)
    → target (([0] || ys2) || tr2) (initial MS) = 1
by auto
qed

then have d-reaches MS (initial MS) [0] 1
unfolding d-reaches.simps by blast
then have 1 ∈ d-reachable MS (initial MS)
unfolding d-reachable.simps by blast

then have {0,1} ⊆ d-reachable MS (initial MS)
using ⟨initial MS ∈ d-reachable MS (initial MS)⟩ by auto
moreover have d-reachable MS (initial MS) ⊆ nodes MS
proof
fix s assume s ∈ d-reachable MS (initial MS)
then have s ∈ reachable MS (initial MS)
using d-reachable-reachable by auto
then show s ∈ nodes MS
by blast
qed
ultimately have d-reachable MS (initial MS) = {0,1}
using example-nodes(1) by blast

fix f' :: nat ⇒ nat list
let ?f = f'( 0 := [], 1 := [0])

```

```

have is-det-state-cover-ass  $M_S$  ?f
  unfolding is-det-state-cover-ass.simps
proof
  show ?f (initial  $M_S$ ) = [] by auto
  show  $\forall s \in d\text{-reachable } M_S$  (initial  $M_S$ ). d-reaches  $M_S$  (initial  $M_S$ ) (?f  $s$ )  $s$ 
  proof
    fix  $s$  assume  $s \in d\text{-reachable } M_S$  (initial  $M_S$ )
    then have  $s \in \text{reachable } M_S$  (initial  $M_S$ )
      using d-reachable-reachable by auto
    then have  $s \in \text{nodes } M_S$ 
      by blast
    then have  $s = 0 \vee s = 1$ 
      using example-nodes(1) by blast
    then show d-reaches  $M_S$  (initial  $M_S$ ) (?f  $s$ )  $s$ 
    proof
      assume  $s = 0$ 
      then show d-reaches  $M_S$  (initial  $M_S$ ) (?f  $s$ )  $s$ 
        using  $\langle d\text{-reachable } M_S$  (initial  $M_S$ ) [] (initial  $M_S$ )  $\rangle$  by auto
      next
      assume  $s = 1$ 
      then show d-reaches  $M_S$  (initial  $M_S$ ) (?f  $s$ )  $s$ 
        using  $\langle d\text{-reachable } M_S$  (initial  $M_S$ ) [0] 1  $\rangle$  by auto
    qed
  qed
qed

moreover have  $V_I = \text{image } ?f$  (d-reachable  $M_S$  (initial  $M_S$ ))
  using  $\langle d\text{-reachable } M_S$  (initial  $M_S$ ) = {0,1}  $\rangle$  by auto

ultimately show ?thesis
  unfolding is-det-state-cover.simps by blast
qed

```

**abbreviation**  $\Omega_I :: (\text{nat}, \text{nat})$  *ATC set*  $\equiv \{ \text{Node } 0 \ (\lambda y . \text{Leaf}) \}$

**lemma** *applicable-set*  $M_S$   $\Omega_I$   
**by** *auto*

**lemma** *example-test-tools* : *test-tools*  $M_S$   $M_I$  *FAIL<sub>I</sub>* *PM<sub>I</sub>*  $V_I$   $\Omega_I$   
**using** *example-productF* *example-det-state-cover* **by** *auto*

**lemma** *OFSM-not-vacuous* :  
 $\exists M :: (\text{nat}, \text{nat}, \text{nat})$  *FSM* . *OFSM*  $M$   
**using** *example-OFSM(1)* **by** *blast*

**lemma** *fault-domain-not-vacuous* :  
 $\exists (M2 :: (\text{nat}, \text{nat}, \text{nat})$  *FSM*) ( $M1 :: (\text{nat}, \text{nat}, \text{nat})$  *FSM*)  $m$  . *asc-fault-domain*  $M2$   $M1$   $m$   
**using** *example-fault-domain* **by** *blast*

**lemma** *test-tools-not-vacuous* :  
 $\exists (M2 :: (\text{nat}, \text{nat}, \text{nat})$  *FSM*)  
 ( $M1 :: (\text{nat}, \text{nat}, \text{nat})$  *FSM*)  
 (*FAIL* :: ( $\text{nat} \times \text{nat}$ ))  
 (*PM* :: ( $\text{nat}, \text{nat}, \text{nat} \times \text{nat}$ ) *FSM*)  
 ( $V :: (\text{nat}$  *list set*)  
 ( $\Omega :: (\text{nat}, \text{nat})$  *ATC set*) . *test-tools*  $M2$   $M1$  *FAIL* *PM*  $V$   $\Omega$   
**proof** (*rule exI*, *rule exI*)

```

show  $\exists$  FAIL PM V  $\Omega$ . test-tools  $M_S M_I$  FAIL PM V  $\Omega$ 
  using example-test-tools by blast
qed

lemma precondition-not-vacuous :
  shows  $\exists$  ( $M2::(\text{nat},\text{nat},\text{nat})$  FSM)
    ( $M1::(\text{nat},\text{nat},\text{nat})$  FSM)
    ( $FAIL::(\text{nat}\times\text{nat})$ )
    ( $PM::(\text{nat},\text{nat},\text{nat}\times\text{nat})$  FSM)
    ( $V::(\text{nat list set})$ )
    ( $\Omega::(\text{nat},\text{nat})$  ATC set)
    ( $m :: \text{nat}$ ) .
    OFSM  $M1 \wedge$  OFSM  $M2 \wedge$  asc-fault-domain  $M2 M1 m \wedge$  test-tools  $M2 M1$  FAIL PM V  $\Omega$ 
proof (intro exI)
  show OFSM  $M_I \wedge$  OFSM  $M_S \wedge$  asc-fault-domain  $M_S M_I 2 \wedge$  test-tools  $M_S M_I$  FAILI PMI VI  $\Omega_I$ 
  using example-OFSM(2,1) example-fault-domain example-test-tools by linarith
qed

end

```

## References

- [1] J. Brunner. Transition systems and automata. *Archive of Formal Proofs*, Oct. 2017. [http://isa-afp.org/entries/Transition\\_Systems\\_and\\_Automata.html](http://isa-afp.org/entries/Transition_Systems_and_Automata.html), Formal proof development.
- [2] R. M. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.
- [3] R. Sachtleben, J. Peleska, R. Hierons, and W.-L. Huang. A mechanised proof of an adaptive state counting algorithm. In *IFIP International Conference on Testing Software and Systems*. Springer, 2019. to appear.