# Formalisation of an Adaptive State Counting Algorithm 

Robert Sachtleben

September 13, 2023


#### 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

1 Finite state machines ..... 2
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
2 Product machines with an additional fail state ..... 31
2.1 Sequences to failure in the product machine ..... 46
3 Adaptive test cases ..... 56
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
4 The lower bound function ..... 67
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
5 Test suite generated by the Adaptive State Counting Algorithm ..... 111
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
6 Sufficiency of the test suite to test for reduction ..... 139
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
7 Correctness of the Adaptive State Counting Algorithm in Hoare-Logic ..... 152
8 Example product machines and properties ..... 167
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 }\times\mathrm{ 'out) }=>\mathrm{ 'state }=>\mathrm{ '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
    \(\lambda a p\). snd \(a \quad\) - execute
    \(\lambda a p\). snd \(a \in \operatorname{succ} A(f s t a) p-\) enabled
    \(\lambda p . p=\operatorname{initial} A \quad-\) initial
    for \(A\)
    defines path \(=\) FSM.path
        and run \(=\) FSM.run
        and reachable \(=\) FSM. reachable
        and nodes \(=F S M\).nodes
    by this
```

abbreviation size-FSM $M \equiv$ 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 $\equiv$ FSM.target
abbreviation states $\equiv$ 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) $F S M \Rightarrow$ 'state
$\Rightarrow$ ('in $\times$ 'out) list set ( $L S$ )
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 $L M \equiv L S M($ initial $M)$

```
lemma language-state-alt-def : LS M \(q=\{\) io | io tr . path \(M(i o \| t r) q \wedge\) length io \(=\) length \(t r\}\)
proof -
    have \(L S M q \subseteq\{\) io | io tr . path \(M\) (io \|tr) \(q \wedge\) length io \(=\) length \(t r\}\)
    proof
        fix \(x r\) assume \(x r\)-assm : \(x r \in L S M q\)
        then obtain \(r\) where \(r\)-def : map fst \(r=x r\) path \(M r q\)
            unfolding language-state-def by auto
        then obtain xs ys where xr-split :xr \(=x s \| y s\)
                        length \(x s=\) length \(y s\)
                        length \(x s=\) length \(x r\)
        by (metis length-map zip-map-fst-snd)
    then have \((x s \| y s) \in\{\) io | io tr . path \(M(i o \| t r) q \wedge\) length io \(=\) length \(t r\}\)
    proof -
            have f1: xs \| ys = map fst r
            by (simp add: \(r\)-def(1) xr-split(1))
            then have f2: path \(M((x s \| y s) \|\) take (min (length \((x s \| y s))(l e n g t h(m a p ~ s n d ~ r)))\)
                                    \((\) map snd \(r)) q\)
            by (simp add: \(r\) - \(\operatorname{def}(2))\)
            have length ( \(x s \| y s\) ) = length
                                    (take \((\) min \((\) length \((x s \| y s))(l e n g t h(m a p ~ s n d r)))(\) map snd \(r))\)
            using \(f 1\) by force
            then show ?thesis
                using \(f 2\) by blast
    qed
    then show \(x r \in\{\) io \(\mid\) io tr . path \(M(i o \| t r) q \wedge\) length io \(=\) length \(t r\}\)
        using xr-split by metis
    qed
    moreover have \(\{\) io \(\mid\) io tr. path \(M(i o \| t r) q \wedge\) length io \(=\) length \(t r\} \subseteq L S M q\)
    proof
        fix \(x s\) assume \(x s\)-assm \(: x s \in\{\) io | io tr . path \(M(i o \| t r) q \wedge\) length io \(=\) length tr \(\}\)
        then obtain \(y s\) where \(y s\)-def : path \(M(x s \| y s) q\) length \(x s=\) length ys
            by auto
        then have \(x s=\) map \(f s t(x s \| y s)\)
            by auto
    then show \(x s \in L S M q\)
        using ys-def unfolding language-state-def by blast
    qed
    ultimately show?thesis
        by auto
qed
```

lemma language-state[intro]:
assumes path $M(w \| r) q$ length $w=$ length $r$
shows $w \in L S M q$
using assms unfolding language-state-def by force

```
lemma language-state-elim [elim]:
    assumes \(w \in L S M q\)
    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 \(\in L S M q\)
    obtains tr1 tr2
    where path \(M(w 1 \|\) tr1 \()\) q length \(w 1=\) length tr1
        path \(M(w 2 \| \operatorname{tr2})(\operatorname{target}(w 1 \| \operatorname{tr} 1) q)\) length \(w 2=\) length tr2
proof -
    obtain \(t r\) where \(t r\)-def : path \(M((w 1\) @ w2) \| tr) q length \((w 1 @ w 2)=\) length \(t r\)
        using assms by blast
    let ? tr1 = take (length w1) tr
    let ?tr2 \(=\) drop \((\) length \(w 1)\) tr
    have tr-split: ? tr1 @ ?tr2 = tr
        by auto
    then show ?thesis
    proof -
        have f1: length \(w 1+\) length \(w 2=\) length \(t r\)
            using \(t r-\operatorname{def}(2)\) by auto
        then have f2: length \(w 2=\) length \(t r-\) length \(w 1\)
            by presburger
        then have length \(w 1=\) length (take (length w1) tr)
            using \(f 1\) by (metis (no-types) tr-split diff-add-inverse2 length-append length-drop)
        then show ?thesis
            using fo 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 \(\in L S M q\)
shows \(w 1 \in L S M q\)
    using assms by (meson language-state language-state-split)
lemma succ-nodes :
    fixes \(A::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)\) FSM
    and \(w::\left({ }^{\prime} a \times{ }^{\prime} b\right)\)
    assumes \(q 2 \in \operatorname{succ} A w q 1\)
    and \(\quad q 1 \in\) nodes \(A\)
shows \(q 2 \in\) nodes \(A\)
proof -
    obtain \(x y\) where \(w=(x, y)\)
        by (meson surj-pair)
    then have \(q 2 \in\) successors \(A\) q1
        using assms by auto
    then have \(q 2 \in\) reachable \(A\) q1
        by blast
    then have \(q 2 \in\) reachable \(A(\) initial \(A)\)
        using assms by blast
    then show \(q 2 \in\) 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 => ('in, 'out,' 'state2) FSM =>
    ('in, 'out, 'state1 >'state2) FSM where
    product A B 三
    O
        succ = \lambda a (p1, p
        inputs = inputs A \cup inputs B,
        outputs = outputs A\cup outputs B,
        initial = (initial A, initial B)
    l
lemma product-simps[simp]:
    succ (product A B) a (p
    inputs (product A B) = inputs }A\cup\mathrm{ inputs }
    outputs (product A B) = outputs A \cup outputs B
    initial (product A B)}=(\mathrm{ initial A, initial B)
    unfolding product-def by simp+
lemma product-target[simp]:
    assumes length w = length }\mp@subsup{r}{1}{}\mathrm{ length }\mp@subsup{r}{1}{}=\mathrm{ length }\mp@subsup{r}{2}{
```



```
    using assms by (induct arbitrary: p1 p2 rule: list-induct3) (auto)
lemma product-path[iff]:
    assumes length w = length r}\mp@subsup{r}{1}{}\mathrm{ length }\mp@subsup{r}{1}{}=\mathrm{ length }\mp@subsup{r}{2}{
```



```
    using assms by (induct arbitrary: p1 p}\mp@subsup{p}{2}{}\mathrm{ rule: list-induct3) (auto)
lemma product-language-state[simp]:LS(product A B) (q1,q2) = LS A q1\capLS B q2
    by (fastforce iff: split-zip)
lemma product-nodes:
    nodes (product A B)\subseteq nodes }A\times\mathrm{ nodes }
proof
    fix q assume q\in nodes (product A B)
    then show q}\in\mathrm{ nodes }A\times\mathrm{ 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\mathrm{ nodes }A\mathrm{ snd }p\in\mathrm{ nodes }
            by auto
        have snd a \in(succ A (fst a) (fst p)) × (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 <fst p \in nodes A〉\langlefst (snd a) \in succ A (fst a) (fst p)>
            by (metis FSM.nodes.simps fst-conv snd-conv)
        moreover have snd (snd a) \in nodes B
            using <snd p \in nodes B\rangle<snd (snd a) \in succ B (fst a) (snd p)〉
            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 intro－ duced 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 => bool where
    finite-FSM M = (finite (nodes M)
                    \wedge ~ f i n i t e ~ ( i n p u t s ~ M ) ~
                    ^ finite (outputs M))
fun observable :: ('in,'out,'state) FSM => bool where
    observable M = (\forallt.\forall s1. ((succ M) t s1 = {})
                                    \vee (\exists s2 . (succ M) t s1 = {s2}))
fun completely-specified :: ('in, 'out, 'state) FSM => bool where
    completely-specified M}=(\forall\mathrm{ s1 }\in\mathrm{ nodes M . }\forallx\in\mathrm{ inputs M .
                                    \existsy\in outputs M.
                                    \exists s2.s2 \in (succ M) (x,y) s1)
fun well-formed :: ('in, 'out, 'state) FSM => bool where
    well-formed M = (finite-FSM M
                \wedge(\forall s1 x y. (x\not\in inputs M\vee y & outputs M)
                        succ M (x,y) s1={})
 inputs M = {}
^ outputs M = {})
abbreviation OFSM M \equiv well-formed M
                    ^ observable M
                            ^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 :
```

lemma set-of-succs-finite :
assumes well-formed M
assumes well-formed M
and }\quadq\in\mathrm{ nodes M
and }\quadq\in\mathrm{ nodes M
shows finite (succ M io q)
shows finite (succ M io q)
proof (rule ccontr)
proof (rule ccontr)
assume infinite (succ M io q)
assume infinite (succ M io q)
moreover have succ M io q\subseteq nodes M
moreover have succ M io q\subseteq nodes M
using assms by (simp add: subsetI succ-nodes)
using assms by (simp add: subsetI succ-nodes)
ultimately have infinite (nodes M)
ultimately have infinite (nodes M)
using infinite-super by blast
using infinite-super by blast
then show False
then show False
using assms by auto
using assms by auto
qed
qed
lemma well-formed-path-io-containment:
lemma well-formed-path-io-containment:
assumes well-formed M
assumes well-formed M
and path M pq
and path M pq
shows set (map fst p)\subseteq(inputs M }\times\mathrm{ outputs M)
shows set (map fst p)\subseteq(inputs M }\times\mathrm{ outputs M)
using assms proof (induction p arbitrary:q)
using assms proof (induction p arbitrary:q)
case Nil
case Nil
then show ?case by auto
then show ?case by auto
next
next
case (Cons a p)
case (Cons a p)
have fst a }\in\mathrm{ (inputs M }\times\mathrm{ outputs M)
have fst a }\in\mathrm{ (inputs M }\times\mathrm{ outputs M)
proof (rule ccontr)
proof (rule ccontr)
assume fst a \& inputs M }\times\mathrm{ outputs }
assume fst a \& inputs M }\times\mathrm{ outputs }
then have fst (fst a) \& inputs M\vee snd (fst a) \& outputs M
then have fst (fst a) \& inputs M\vee snd (fst a) \& outputs M
by (metis SigmaI prod.collapse)
by (metis SigmaI prod.collapse)
then have succ M (fst a) q={}
then have succ M (fst a) q={}
using Cons by (metis prod.collapse well-formed.elims(2))
using Cons by (metis prod.collapse well-formed.elims(2))
moreover have (snd a) \in succ M (fst a) q
moreover have (snd a) \in succ M (fst a) q
using Cons by auto
using Cons by auto
ultimately show False
ultimately show False
by auto
by auto
qed

```
    qed
```

```
    moreover have set (map fst p)\subseteq(inputs M }\times\mathrm{ outputs M)
    using Cons by blast
    ultimately show ?case
    by auto
qed
lemma path-input-containment :
    assumes well-formed M
    and path M pq
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\mathrm{ outputs M)
        using well-formed-path-io-containment[OF snoc.prems] by assumption
    then have (fst a) \in(inputs M }\times\mathrm{ 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.prems(1)]
        using snoc.prems(2) by blast
    ultimately show ?case
        by simp
qed
lemma path-state-containment :
    assumes path M p q
    and }\quadq\in\mathrm{ nodes }
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 Mq
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)
            \thesis. (\tr. \llbracketpath M (io| tr) q; length tr = length io\rrbracket \Longrightarrow thesis) \Longrightarrow thesis`
            assms(1) map-fst-zip path-input-containment)
qed
lemma set-of-paths-finite :
    assumes well-formed M
    and }q1\in\mathrm{ nodes M
shows finite { p.path M p q1 ^ target p q1 = q2 ^ length p\leqk}
proof -
    let ?trs = { tr. set tr \subseteq nodes M^ length tr \leqk}
    let ?ios = { io.set io \subseteqinputs M > outputs M ^ length io \leqk}
    let ?iotrs = image ( }\lambda(\mathrm{ io,tr ). io | tr) (?ios }\times\mathrm{ ? ?trs)
    let ?paths ={ p. path M p q1 ^ target p q1 = q2 ^ length p \leqk}
    have finite (inputs M }\times\mathrm{ outputs M)
        using assms by auto
    then have finite?ios
        using assms by (simp add: finite-lists-length-le)
```

```
    moreover have finite ?trs
    using assms by (simp add: finite-lists-length-le)
    ultimately have finite ?iotrs
    by auto
    moreover have ?paths \subseteq? ?otrs
    proof
    fix p assume p-assm: p\in{p. path M pq1 ^ target pq1 = q2 ^ length p\leqk}
    then obtain io tr where p-split : p=io| |r ^ length io = length tr
        using that by (metis (no-types) length-map zip-map-fst-snd)
    then have io }\in\mathrm{ ?ios
        using well-formed-path-io-containment
    proof -
        have f1: path M p q1 ^ target p q1 = q2 ^ length p \leqk
            using p-assm by force
        then have set io \subseteqinputs M }\times\mathrm{ outputs M
            by (metis (no-types) assms(1) map-fst-zip p-split well-formed-path-io-containment)
        then show ?thesis
            using f1 by (simp add: p-split)
    qed
    moreover have tr f ?trs using p-split
    proof -
        have f1: path M (io ||tr) q1 ^ target (io|tr) q1 = q2
                    ^ length (io | tr) \leqk using < p\in{p. path M p q1
                    ^target p q1 = q2 ^ length p\leqk}> p-split by force
        then have f2: length tr \leqk by (simp add: p-split)
        have set tr\subseteq nodes M
            using f1 by (metis (no-types) assms(2) length-map p-split path-state-containment
                zip-eq zip-map-fst-snd)
            then show ?thesis
            using f2 by blast
    qed
    ultimately show p\in ?iotrs
        using p-split by auto
    qed
    ultimately show ?thesis
    using Finite-Set.finite-subset by blast
qed
lemma non-distinct-duplicate-indices :
    assumes \neg distinct xs
shows \exists i1 i2. i1 \not= i2 \wedge xs ! i1 = xs ! iQ ^ i1 \leqlength xs ^ i2 \leqlength xs
    using assms by (meson distinct-conv-nth less-imp-le)
lemma reaching-path-without-repetition:
    assumes well-formed M
    and q2 \in reachable M q1
    and q1 \in nodes M
shows \exists p. path M p q1 ^ target p q1 = q2 ^ distinct (q1 # states p q1)
proof -
    have shorten-nondistinct: }\forallp.(\mathrm{ path M pq1 ^target p q1 = q2 ^ ᄀdistinct (q1 # states p q1))
                    \longrightarrow(\exists p' . path M p'q1 ^ target p' q1 = q2 ^ length p
    proof
        fix p
        show (path M p q1 ^ target p q1 = q2 ^ ᄀ distinct (q1 # states p q1))
            \longrightarrow ( \exists p ^ { \prime } . p a t h ~ M ~ p ~ q ' ~ q 1 ~ \wedge ~ t a r g e t ~ p ' q 1 ~ = ~ q 2 ~ \ ~ l e n g t h ~ p ' < l e n g t h ~ p ) ~
        proof
            assume assm : path M p q1 ^ target p q1 = q2 ^ ᄀ distinct (q1 # states p q1)
            then show ( }\exists\mp@subsup{p}{}{\prime}\mathrm{ . path M p}\mp@subsup{p}{}{\prime}q1\wedge\mathrm{ target }\mp@subsup{p}{}{\prime}q1=q2\wedge length \mp@subsup{p}{}{\prime}<l\mathrm{ length p)
            proof (cases q1 \in set (states p q1))
                case True
                have \exists i1 .target (take i1 p) q1 = q1 ^ i1 \leq length p ^i1>0
            proof (rule ccontr)
```

assume $\neg(\exists$ i1. target $($ take i1 $p) q 1=q 1 \wedge i 1 \leq$ length $p \wedge i 1>0)$
then have $\neg(\exists i 1 .($ states $p q 1)!i 1=q 1 \wedge i 1 \leq$ length $($ states $p q 1))$
by (metis True in-set-conv-nth less-eq-Suc-le scan-length scan-nth zero-less-Suc)
then have $q 1 \notin$ set (states $p$ q1)
by (meson in-set-conv-nth less-imp-le)
then show False
using True by auto
qed
then obtain $i 1$ where $i 1$-def : target (take i1 $p$ ) $q 1=q 1 \wedge i 1 \leq$ length $p \wedge i 1>0$ by auto
then have path $M$ (take i1 p) q1 using assm by (metis FSM.path-append-elim append-take-drop-id)
moreover have path $M$ (drop i1 p) q1
using i1-def by (metis FSM.path-append-elim append-take-drop-id assm)
ultimately have path $M($ drop i1 $p) q 1 \wedge($ target $(\operatorname{drop}$ i1 $p) q 1=q 2)$
using i1-def by (metis (no-types) append-take-drop-id assm fold-append o-apply)
moreover have length (drop i1 $p$ ) < length $p$
using i1-def by auto
ultimately show ?thesis
using assms by blast
next
case False
then have assm' : path $M p q 1 \wedge \operatorname{target} p q 1=q 2 \wedge \neg \operatorname{distinct}($ states $p q 1)$ using assm by auto
have $\exists i 1$ i2 . i1 $\neq i 2 \wedge$ target $($ take i1 $p) q 1=\operatorname{target}($ take i2 $p) q 1$ $\wedge i 1 \leq$ length $p \wedge i 2 \leq$ length $p$
proof (rule ccontr)
assume $\neg(\exists i 1 i 2 . i 1 \neq i 2 \wedge$ target $($ take $i 1 p) q 1=\operatorname{target}($ take $i 2 p) q 1$ $\wedge i 1 \leq$ length $p \wedge i 2 \leq$ length $p)$
then have $\neg(\exists i 1$ i2 $. i 1 \neq i 2 \wedge($ states $p q 1)!i 1=($ states $p q 1)!i 2$

$$
\wedge i 1 \leq \text { length }(\text { states } p q 1) \wedge i 2 \leq \text { length }(\text { states } p q 1))
$$

by (metis (no-types, lifting) Suc-leI assm' distinct-conv-nth nat.inject scan-length scan-nth)
then have distinct (states $p$ q1)
using non-distinct-duplicate-indices by blast
then show False
using assm' by auto
qed
then obtain $i 1 i 2$ where $i$-def : i1 $<i 2 \wedge$ target (take i1 p) q1 $=\operatorname{target}($ take i2 $p) q 1$ $\wedge i 1 \leq$ length $p \wedge i 2 \leq$ length $p$
by (metis nat-neq-iff)
then have path $M$ (take i1 p) q1 using assm by (metis FSM.path-append-elim append-take-drop-id)
moreover have path $M$ (drop i2 p) (target (take i2 p) q1)
by (metis FSM.path-append-elim append-take-drop-id assm)
ultimately have path $M(($ take i1 p) @ (drop i2 p) ) q1
$\wedge($ target $(($ take i1 $p) @(\operatorname{drop} i 2 p)) q 1=q 2)$
using $i$-def assm
by (metis FSM.path-append append-take-drop-id fold-append o-apply)
moreover have length $(($ take i1 $p) @($ drop i2 $p))<$ length $p$
using $i$-def by auto
ultimately have path $M(($ take i1 $p) @($ drop i2 p) $) q 1$
$\wedge \operatorname{target}\left((\right.$ take i1 p) @ $(\operatorname{drop}$ i2 $p)) q 1=q_{2}$
$\wedge$ length $(($ take i1 $p)$ @ $($ drop i2 $p))<$ length $p$
by $\operatorname{simp}$
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'\leq length p)}
    let ?minPath = arg-min length ( }\lambda\mathrm{ io . io }\in\mathrm{ ?paths)
    have ?paths }\not=\mathrm{ empty
    using p-def by auto
moreover have finite ?paths
    using assms by (simp add: set-of-paths-finite)
ultimately have minPath-def : ?minPath \in ?paths }\wedge(\forall\mp@subsup{p}{}{\prime}\in\mathrm{ ?paths . length ?minPath }\leq\mathrm{ length p')
    by (meson arg-min-nat-lemma equals0I)
moreover have distinct (q1 # states ?minPath q1)
proof (rule ccontr)
    assume ᄀ distinct (q1 # states ?minPath q1)
    then have \exists 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 $\in L S M q$
and observable $M$
and $\quad$ path $M(i o \| t r 1) q$ length io $=$ length $\operatorname{tr} 1$
and path $M$ (io \| tr2) q length io $=$ length tr2
shows $\operatorname{tr} 1=\operatorname{tr2}$
proof (rule ccontr)
assume tr-assm : tr1 $\neq$ tr2
then have state-diff : (states $(i o \| t r 1) q) \neq($ states $(i o \| t r 2) q)$
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 $\operatorname{simp}$
next
case (Cons io-hd io-tl)
then obtain $\operatorname{tr} 1-h d \operatorname{tr1} 1-t l \operatorname{tr2} 2-h d t r 2-t l$ where $\operatorname{tr}-s p l i t: \operatorname{tr} 1=\operatorname{tr} 1-h d \# \operatorname{tr} 1-t l$
$\wedge \operatorname{tr2}=\operatorname{tr2}$ - $h d \# \operatorname{tr2}$ - $t l$
by (metis length-0-conv neq-Nil-conv)
have $p 1$ : path $M([i o-h d] \|[t r 1-h d]) q$
using Cons.prems tr-split by auto
have p2: path $M([i o-h d] \|[t r 2-h d]) q$
using Cons.prems tr-split by auto
have $\operatorname{tr}-h d-e q: \operatorname{tr} 1-h d=\operatorname{tr} 2-h d$
using Cons.prems unfolding observable.simps
proof -
assume $\forall t$ s1. succ $M t s 1=\{ \} \vee(\exists$ s2. succ $M t s 1=\{s \mathcal{Z}\})$
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

```
    then show ?thesis
    using Cons.IH Cons.prems(3) Cons.prems(4) Cons.prems(5) Cons.prems(6) Cons.prems(7) assms(2)
        tr-split by auto
    qed
qed
lemma observable-path-unique-ex[elim]:
    assumes observable M
    and io \inLSMq
obtains tr
where {t. path M (io|t)q\wedge 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\wedge length io = length t }}\not={
        by blast
    moreover have }\forallt\in{t. path M(io|t)q\wedge length io = length t } . t = tr
        using assms tr-def by auto
    ultimately show ?thesis
        using that by moura
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 }\times\mathrm{ nodes M1)
        by simp
    moreover have nodes ?PM \subseteq nodes M2 }\times\mathrm{ 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 \cup inputs M1
        outputs ?PM = outputs M2 \cup outputs M1
        by auto
    then show ( }\forall\mathrm{ s1 x y. x & inputs ?PM V y & outputs ?PM }\longrightarrow\mathrm{ succ ?PM (x,y) s1={})
                                    \wedge \text { inputs ?PM } \neq \{ \} \wedge \text { outputs ?PM } \neq \{ \}
        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 }=>\mathrm{ ('in }\times\mathrm{ 'out) list }=>\mathrm{ '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\in io-targets M (initial M) io
    shows io \inLM
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 \(M(w \| t r) q\)
    and length \(w=\) length tr
shows target \((w \| t r) q \in\) io-targets \(M q w\)
    using assms by auto
lemma io-targets-observable-singleton-ex :
    assumes observable \(M\)
    and \(\quad i o \in L S M q 1\)
shows \(\exists\) q2 . io-targets M q1 io \(=\{q 2\}\)
proof -
    obtain \(t r\) where \(\operatorname{tr}\)-def: \(\{t\). path \(M(i o \| t) q 1 \wedge\) length io \(=\) length \(t\}=\{t r\}\)
        using assms observable-path-unique-ex by (metis (mono-tags, lifting))
    then have io-targets \(M\) q1 io \(=\{\operatorname{target}(i o \| t r) q 1\}\)
        by fastforce
    then show?thesis
        by blast
qed
lemma io-targets-observable-singleton-ob :
    assumes observable \(M\)
    and \(\quad i o \in L S M q 1\)
obtains \(q\) 2
    where io-targets \(M\) q1 io \(=\{q 2\}\)
proof -
    obtain \(t r\) where \(t r\)-def : \(\{t\). path \(M(i o \| t) q 1 \wedge\) length io \(=\) length \(t\}=\{t r\}\)
        using assms observable-path-unique-ex by (metis (mono-tags, lifting))
    then have io-targets \(M\) q1 io \(=\{\operatorname{target}(i o \| t r) q 1\}\)
        by fastforce
    then show ?thesis using that by blast
qed
lemma io-targets-elim [elim] :
    assumes \(p \in\) io-targets \(M q\) io
obtains tr
where target (io \|tr) \(q=p \wedge\) path \(M(i o \| t r) q \wedge\) length io \(=\) length tr
    using assms unfolding io-targets.simps by force
lemma io-targets-reachable :
    assumes \(q 2 \in\) io-targets \(M\) q1 io
    shows \(q 2 \in\) reachable \(M q 1\)
    using assms unfolding io-targets.simps by blast
lemma io-targets-nodes :
    assumes q2 \(\in\) io-targets M q1 io
    and \(\quad q 1 \in\) nodes \(M\)
shows \(q 2 \in\) nodes \(M\)
    using assms by auto
lemma observable-io-targets-split
    assumes observable \(M\)
    and io-targets \(M q 1(v s @ x s)=\{q 3\}\)
    and io-targets \(M\) q1 vs \(=\{q 2\}\)
shows io-targets M q2 xs \(=\{q 3\}\)
proof -
    have vs @ xs \(\in L S M\) q1
        using assms(2) by force
    then obtain \(\operatorname{tr} V \operatorname{tr} X\) where \(t r\)-def :
                path \(M(v s \| t r V) q 1\) length \(v s=\) length \(\operatorname{tr} V\)
            path \(M(x s \| \operatorname{trX})(\) target \((v s \| \operatorname{trV}) q 1)\) length \(x s=\) length \(\operatorname{tr} X\)
        using language-state-split \([\) of vs xs \(M\) q1] by auto
    then have tgt- \(V\) : target ( \(v s \| \operatorname{trV}\) ) q1 = q2
        using assms(3) by auto
    then have path- \(X:\) path \(M(x s \| \operatorname{tr} X) q 2 \wedge\) length \(x s=\) length \(\operatorname{tr} X\)
```

using tr-def by auto

```
have tgt-all: target (vs @ \(x s \| \operatorname{tr} V\) @ trX) \(q 1=q 3\)
proof -
    have \(f 1: \exists c s . q 3=\operatorname{target}(v s @ x s \| c s) q 1\)
                    \(\wedge\) path \(M(v s @ x s \| c s) q 1 \wedge\) length \((v s @ x s)=\) length \(c s\)
        using assms(2) by auto
    have length (vs @ xs) = length tr \(V+\) length \(\operatorname{tr} X\)
        by (simp add: tr-def(2) tr-def(4))
    then have length (vs @ xs) = length (trV @ trX)
        by \(\operatorname{simp}\)
    then show ?thesis
        using \(f 1\) by (metis FSM.path-append «vs @ xs \(\in L S M\) q1〉assms(1) observable-path-unique
            \(t r-\operatorname{def}(1) t r-\operatorname{def}(2) t r-\operatorname{def}(3) z i p-a p p e n d)\)
qed
then have target \(((v s \| \operatorname{tr} V) @(x s \| \operatorname{tr} X)) q 1=q 3\)
    using \(t r\)-def by simp
then have target \((x s \| \operatorname{trX}) q 2=q 3\)
    using tgt- \(V\) by auto
then have \(q 3 \in\) io-targets \(M\) q2 xs
    using path- \(X\) by auto
then show ?thesis
    by (metis (no-types) <observable \(M\) 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 $=\{q 2\}$
and path $M(i o \| t r) q 1$
and length io $=$ length tr
shows target ( $i o \| t r$ ) $q 1=q 2$
using assms by auto
lemma target-in-states :
assumes length io $=$ length $t r$
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 $t r$
shows length io $=0 \Longrightarrow$ target (io $\| t r) q=q$
length io $>0 \Longrightarrow \operatorname{target}($ io $\| t r) q=$ last $t r$
proof -
show length io $=0 \Longrightarrow$ target (io $\|$ tr) $q=q$ by simp
show length io $>0 \Longrightarrow \operatorname{target}$ (io $\| t r) q=$ last $t r$
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 $M(i o \| t r) q$
and length io $=$ length $t r$
shows io-targets $M q$ io $=\{$ target $(i o \| t r) q\}$
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 \(=\{q 2\}\)
    and path \(M(i o \| t r) q 1\)
    and length io \(=\) length tr
shows target (io \|tr) q1 \(=q 2\)
proof -
    have target (io \|tr) q1 \(\in\) io-targets \(M\) q1 io using \(\operatorname{assms(2)~assms(3)~by~auto~}\)
    then show ?thesis using assms(1) by blast
qed
```

lemma index-last-take :
assumes $i<$ length $x s$
shows $x s!i=$ last (take (Suc i) xs)
by (simp add: assms take-Suc-conv-app-nth)
lemma path-last-io-target:
assumes path $M(x s \| t r) q$
and length $x s=$ length $t r$
and length $x s>0$
shows last tr $\in$ io-targets $M q x s$
proof -
have last tr $=$ target ( $x s \| \operatorname{tr}$ ) $q$
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 $M(x s \| t r) q$
and length $x s=$ length tr
and length $x s>0$
shows last (take (Suc i) tr) $\in$ 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 $x s$
and $\quad$ path $M(x s \| t r) q$
and length $x s=$ length $t r$
and length $x s>0$
shows (states (xs \|tr)q)!ieio-targets Mq(take (Suc i) xs)
proof -
have (states $(x s \| t r) q)!i=$ last (take (Suc i) (states $(x s \| t r) q))$
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) $\in$ io-targets $M q$ (take (Suc i) xs)
by (meson assms(2) assms(3) assms(4) path-prefix-io-targets)
ultimately show ?thesis
by $\operatorname{simp}$
qed
lemma observable-io-targets-append :
assumes observable $M$
and io-targets $M$ q1 vs $=\{q 2\}$
and io-targets $M q 2 x s=\{q 3\}$

```
shows io-targets M q1 \((v s @ x s)=\{q 3\}\)
proof -
    obtain \(\operatorname{tr} V\) where path \(M(v s \| \operatorname{tr} V) q 1 \wedge\) length \(\operatorname{tr} V=\) length vs \(\wedge\) target \((v s \| \operatorname{tr} V) q 1=q 2\)
        by (metis assms(2) io-targets-elim singletonI)
    moreover obtain \(\operatorname{tr} X\) where path \(M(x s \| \operatorname{tr} X)\) q2 \(\wedge\) length \(\operatorname{tr} X=\) length \(x s\)
                    \(\wedge \operatorname{target}(x s \| \operatorname{tr} X) q 2=q 3\)
        by (metis assms(3) io-targets-elim singletonI)
    ultimately have path \(M(v s @ x s \| t r V @ \operatorname{tr} X) q 1 \wedge\) length \((t r V @ t r X)=l e n g t h(v s @ x s)\)
                \(\wedge \operatorname{target}(v s @ x s \| \operatorname{tr} V @ \operatorname{tr} X) q 1=q 3\)
        by auto
    then show ?thesis
        by (metis assms(1) obs-target-is-io-targets)
qed
lemma io-path-states-prefix :
    assumes observable \(M\)
    and path \(M(i o 1 \| \operatorname{tr} 1) q\)
    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 \(? \operatorname{tr} 1^{\prime}=\) take (length tr1) tr2
    let ?io1' = take (length tr1) io2
    have path \(M\) (? io1' \(\|\) ?tr1') \(q\)
        by (metis FSM.path-append-elim append-take-drop-id assms(4) take-zip)
    have length ? tr1 \({ }^{\prime}=\) length ? io1'
        using assms (5) by auto
    have ? io1' \(=i o 1\)
    proof -
        have \(\forall\) ps psa. \(\neg\) prefix \(\left(p s::\left({ }^{\prime} a \times\right.\right.\) 'b) list) psa \(\vee\) length \(p s \leq\) length psa
            using prefix-length-le by blast
        then have length (take (length tr1) io2) = length io1
        using assms(3) assms(6) min.absorb2 by auto
        then show ?thesis
            by (metis assms(6) min.cobounded2 min-def-raw prefix-length-prefix
            prefix-order.dual-order.antisym take-is-prefix)
    qed
    show \(\operatorname{tr} 1=\) ? tr1 \({ }^{\prime}\)
        by (metis «length (take (length tr1) tr2) \(=\) length (take (length tr1) io2) >
            〈path \(M\) (take (length tr1) io2 || take (length tr1) tr2) q〉〈take (length tr1) io2 \(=\) io1〉
            \(\operatorname{assms}(1) \operatorname{assms(2)} \operatorname{assms}(3)\) language-state observable-path-unique)
qed
```

lemma observable－io－targets－suffix ：
assumes observable $M$
and io－targets $M$ q1 vs $=\{q 2\}$
and io－targets $M q 1(v s @ x s)=\{q 3\}$
shows io－targets $M q 2 x s=\{q 3\}$
proof－
have prefix vs（vs＠xs） by auto
obtain $\operatorname{tr} V$ where path $M(v s \| \operatorname{tr} V) q 1 \wedge$ length $\operatorname{tr} V=$ length $v s \wedge \operatorname{target}(v s \| t r V) q 1=q 2$ by（metis assms（2）io－targets－elim singletonI）
moreover obtain $\operatorname{tr} V X$ where path $M(v s @ x s \| \operatorname{tr} V X) q 1$

$$
\wedge \text { length } \operatorname{tr} V X=\text { length }(v s @ x s) \wedge \operatorname{target}(v s @ x s \| \operatorname{tr} V X) q 1=q 3
$$

by（metis assms（3）io－targets－elim singletonI）

```
    ultimately have trV = take (length trV) trVX
    using io-path-states-prefix[OF assms(1) -- - <prefix vs (vs@xs)>, of trV q1 trVX] by auto
    show ?thesis
    by (meson assms(1) assms(2) assms(3) observable-io-targets-split)
qed
```

```
lemma observable-io-target-is-singleton[simp]:
    assumes observable \(M\)
    and \(\quad p \in\) io-targets \(M q\) io
shows io-targets \(M q\) io \(=\{p\}\)
proof -
    have io \(\in L S M q\)
        using assms(2) by auto
    then obtain \(p^{\prime}\) where io-targets \(M q\) io \(=\left\{p^{\prime}\right\}\)
        using assms(1) by (meson io-targets-observable-singleton-ex)
    then show ?thesis
        using assms(2) by simp
qed
```

lemma observable-path-prefix :
assumes observable $M$
and path $M(i o \| t r) q$
and length io $=$ length tr
and path $M(i o P \| t r P) q$
and length io $P=$ length tr $P$
and prefix ioP io
shows $t r P=$ take (length ioP) $t r$
proof -
have ioP-def : ioP = take (length ioP) io
using $\operatorname{assms}(6)$ by (metis append-eq-conv-conj prefixE)
then have take (length ioP) $($ io $\| t r)=$ take (length ioP) io \|take (length ioP) tr
using take-zip by blast
moreover have path $M$ (take (length ioP) (io \|tr)) q
using assms by (metis FSM.path-append-elim append-take-drop-id)
ultimately have path $M$ (take (length ioP) io \| 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 $\|$ take (length ioP) tr) $q \wedge$ length io $P=$ length (take (length ioP) tr)
using assms(3) using ioP-def by auto
then show?thesis
by (meson assms(1) assms(4) assms(5) language-state observable-path-unique)
qed
lemma io-targets-succ :
assumes $q 2 \in$ io-targets $M q 1[x y]$
shows $q 2 \in$ succ $M x y q 1$
proof -
obtain $t r$ where $t r$-def : target $([x y] \| t r) q 1=q 2$
path $M([x y] \| t r) q 1$
length $[x y]=$ length $t r$
using assms by auto
have length tr $=$ Suc 0
using «length $[x y]=$ length tr> by auto
then obtain $q Q^{\prime}$ where $\operatorname{tr}=\left[q 2^{2}\right]$
by (metis Suc-length-conv length-0-conv)
then have target $([x y] \| t r) q 1=q 2^{\prime}$
by auto
then have $q 2^{\prime}=q$ 2
using $\langle$ target $([x y] \| t r) q 1=q 2\rangle$ by $\operatorname{simp}$
then have path $M([x y] \|[q 2]) q 1$
using $\operatorname{tr}-\operatorname{def}(2)<t r=[q 2]\rangle$ by auto
then have path $M[(x y, q 2)] q 1$

```
    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) \in succ M (fst (xy,q2)) q1 \Longrightarrow path M [] (snd (xy,q2))
        \Longrightarrow2 \in 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-reached-by Mpxs qutr ys 三
    ((length xs \(=\) length \(y s \wedge\) length \(x s=\) length \(t r\)
    \(\wedge(\) path \(M((x s \| y s) \| t r) p) \wedge \operatorname{target}((x s \| y s) \| t r) p=q)\)
    \(\wedge(\forall\) ys2 tr2.\((\) length \(x s=\) length ys2 \(\wedge\) length \(x s=\) length tr2
    \(\wedge\) path \(M((x s|\mid y s 2) \| t r 2) p) \longrightarrow \operatorname{target}((x s|\mid y s 2) \| \operatorname{tr2}) p=q))\)
fun \(d\)-reaches :: ('in, 'out, 'state) \(F S M \Rightarrow\) 'state \(\Rightarrow\) 'in list \(\Rightarrow\) 'state \(\Rightarrow\) bool where
    \(d\)-reaches \(M\) p xs \(q=(\exists\) tr ys. \(d\)-reached-by \(M\) p xs \(q\) tr \(y s)\)
fun d-reachable :: ('in, 'out, 'state) \(F S M \Rightarrow\) 'state \(\Rightarrow\) 'state set where
    \(d\)-reachable \(M p=\{q .(\exists\) xs. \(d\)-reaches \(M\) pxs \(q)\}\)
lemma d-reaches-unique \([\) elim] :
    assumes \(d\)-reaches \(M p\) xs \(q 1\)
    and d-reaches M pxs q2
shows \(q 1=q 2\)
using assms unfolding \(d\)-reaches.simps by blast
lemma \(d\)-reaches-unique-cases[simp]: \(\{q\). d-reaches \(M(\) initial \(M)\) xs \(q\}=\{ \}\)
                            \(\vee(\exists q 2 \cdot\{q \cdot d\)-reaches \(M(\) initial \(M)\) xs \(q\}=\{q 2\})\)
    unfolding \(d\)-reaches.simps by blast
lemma d-reaches-unique-obtain \([\) simp \(]\) :
    assumes \(d\)-reaches \(M(\) initial \(M)\) xs \(q\)
shows \(\{p\). \(d\)-reaches \(M(\) initial \(M)\) xs \(p\}=\{q\}\)
    using assms unfolding \(d\)-reaches.simps by blast
lemma \(d\)-reaches-io-target :
    assumes \(d\)-reaches \(M p\) xs \(q\)
    and length ys \(=\) length \(x s\)
shows io-targets \(M p(x s \| y s) \subseteq\{q\}\)
proof
    fix \(q^{\prime}\) assume \(q^{\prime} \in\) io-targets \(M p(x s \| y s)\)
    then obtain \(\operatorname{tr} Q\) where path \(M((x s \| y s) \| \operatorname{tr} Q) p \wedge\) length \((x s \| y s)=\) length \(\operatorname{tr} Q\)
        by auto
    moreover obtain \(\operatorname{trD}\) ysD where \(d\)-reached-by \(M\) p xs \(q \operatorname{trD}\) ysD using assms(1)
        by auto
    ultimately have target \(((x s \| y s) \| t r Q) p=q\)
        by (simp add: assms(2))
    then show \(q^{\prime} \in\{q\}\)
        using 〈d-reached-by \(M p\) xs \(q \operatorname{trD}\) ysD〉〈q\(q^{\prime} \in\) io-targets \(\left.M p(x s \| y s)\right\rangle\) assms(2) by auto
qed
lemma \(d\)-reachable-reachable : d-reachable \(M p \subseteq\) reachable \(M p\)
    unfolding \(d\)-reaches.simps \(d\)-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) \(F S M \Rightarrow\) ('state \(\Rightarrow\) 'in list) \(\Rightarrow\) bool where
    is-det-state-cover-ass \(M f=(f(\) initial \(M)=[] \wedge(\forall s \in d\)-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 $\quad s 1 \in d$-reachable $M($ initial $M)$
and $\quad s 2 \in d$-reachable $M($ initial $M)$
and $s 1 \neq s 2$
shows $\neg(d$-reaches $M($ initial $M)(f$ s2) $s 1)$
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 $\quad s 1 \in d$-reachable $M($ initial $M)$
and $\quad s \mathcal{Z} \in d$-reachable $M($ initial $M)$
and $s 1 \neq s 2$
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) \(F S M \Rightarrow\) '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 \(\quad v \in V\)
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 $[\operatorname{simp}]$ :
assumes is-det-state-cover $M V$
and finite (nodes M)
shows $\quad \operatorname{card}(d$-reachable $M($ initial $M))=\operatorname{card} V$
proof -
obtain $f$ where $f$-def : is-det-state-cover-ass $M f \wedge V=\operatorname{image} f(d$-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 $M(\operatorname{initial} M)))$
by $\operatorname{simp}$
have $d$-reachable $M($ initial $M) \subseteq$ nodes $M$
unfolding $d$-reachable.simps $d$-reaches.simps using $d$-reachable-reachable by blast
then have $d r$-finite : finite ( $d$-reachable $M($ initial $M)$ )
using assms infinite-super by blast
then have card-le : card $($ image $f(d$-reachable $M($ initial $M)) \leq \operatorname{card}(d$-reachable $M($ initial $M))$
using card-image-le by blast
have $\operatorname{card}($ image $f(d$-reachable $M($ initial $M)))=\operatorname{card}(d$-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 <d-reachable M (initial M)\subseteq nodes M> 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 []$\in V$
proof -
have $d$-reached-by $M$ (initial $M$ ) [] (initial $M$ ) [] []
by ( simp add: FSM.nil)
then have $d$-reaches $M($ initial $M)$ [] (initial $M)$
by auto
have initial $M \in d$-reachable $M$ (initial $M)$
by (metis (no-types) 〈d-reaches $M$ (initial $M$ ) [] (initial $M$ )〉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 [] $\in V$
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 $M)=[]$
by auto
moreover have initial $M \in d$-reachable $M$ (initial $M$ )
proof -
have $d$-reaches $M($ initial $M)[]($ 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 \(\preceq 200\) )
    where
    \(M 1 \preceq M 2=(L S M 1(\) initial M1 \() \subseteq L S M 2(\) initial M2 \())\)
```

lemma language-state-inclusion-of-state-reached-by-same-sequence :
assumes $L S$ M1 q1 $\subseteq L S$ M2 q2
and observable M1
and observable M2
and io-targets M1 q1 io $=\{q 1 t\}$
and io-targets M2 q2 io $=\{q 2 t\}$
shows $L S$ M1 q1t $\subseteq L S$ M2 q2t
proof

```
    fix \(x\) assume \(x \in L S\) M1 q1t
    obtain \(q 1 x\) where io-targets M1 q1t \(x=\{q 1 x\}\)
        by (meson \(\prec x \in L S\) M1 q1t assms(2) io-targets-observable-singleton-ex)
    have io \(\in L S\) M1 q1
        using assms(4) by auto
    have \(i o @ x \in L S M 1\) q1
        using observable-io-targets-append[OF assms(2)〈io-targets M1 q1 io \(=\{\) q1t \(\}>\)
            〈io-targets M1 q1t \(x=\{q 1 x\}\rangle]\)
        by (metis io-targets-elim language-state singletonI)
    then have \(i o @ x \in L S\) M2 q2
        using assms(1) by blast
    then obtain \(q 2 x\) where io-targets M2 \(q 2(i o @ x)=\{q 2 x\}\)
        by (meson assms(3) io-targets-observable-singleton-ex)
    show \(x \in L S\) M2 q2t
        using observable-io-targets-split[OF assms(3)〈io-targets M2 q2 (io @ \(x)=\{q 2 x\}\) 〕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-state-for-input ::
    ('in, 'out, 'state) \(F S M \Rightarrow\) 'state \(\Rightarrow\) 'in list \(\Rightarrow\) ('in \(\times\) 'out) list set where
    language-state-for-input \(M q x s=\{(x s \| y s) \mid y s .(\) length \(x s=\) length \(y s \wedge(x s \| y s) \in L S M q)\}\)
fun language-state-for-inputs ::
    ('in, 'out, 'state) FSM \(\Rightarrow\) 'state \(\Rightarrow\) 'in list set \(\Rightarrow\) ('in \(\times\) 'out) list set
        \(\left(\left(L S_{i n}--\right)[1000,1000,1000]\right)\) where
    language-state-for-inputs \(M\) I ISeqs \(=\{(x s \| y s) \mid x s y s .(x s \in\) ISeqs
                        \(\wedge\) length \(x s=\) length \(y s\)
                        \(\wedge(x s \| y s) \in L S M q)\}\)
```

abbreviation $L_{i n} M T S \equiv L S_{i n} M($ initial $M) T S$
abbreviation io-reduction-on M1 TS M2 $\equiv\left(L_{i n} M 1 T S \subseteq L_{i n} M 2 T S\right)$
notation
io-reduction-on ((- $\preceq \llbracket-\rrbracket-)[1000,0,0] 61)$
notation (latex output)
io-reduction-on ((- 〔- -) $[1000,0,0] 61)$
lemma language-state-for-input-alt-def :
language-state-for-input $M q x s=L S_{i n} M q\{x s\}$
unfolding language-state-for-input.simps language-state-for-inputs.simps by blast
lemma language-state-for-inputs-alt-def :
$L S_{i n} M q$ ISeqs $=\bigcup($ image (language-state-for-input $M$ q) ISeqs)
by auto
lemma language-state-for-inputs-in-language-state :
$L S_{\text {in }} M q T \subseteq$ language-state $M q$
unfolding language-state-for-inputs.simps language-state-def
by blast
lemma language-state-for-inputs-map-fst :
assumes io $\in$ language-state $M q$
and map fst io $\in T$
shows io $\in L S_{\text {in }} M q T$
proof -
let $? \times x=$ map fst io
let $? y s=$ map snd io
have ? $x s \in T \wedge$ length ? $x s=$ length ? $y s \wedge$ ? $x s \|$ ? $y s \in$ language-state $M q$

```
    using assms(2,1) by auto
    then have ?xs| ?ys \inLS in MqT
        unfolding language-state-for-inputs.simps by blast
    then show ?thesis
    by simp
qed
lemma language-state-for-inputs-nonempty:
    assumes set xs \subseteqinputs M
    and completely-specified M
    and }\quadq\in\mathrm{ nodes }
shows LS in M q {xs} \not={}
using assms proof (induction xs arbitrary:q)
    case Nil
    then show ?case by auto
next
    case (Cons x xs)
    then have }x\in\mathrm{ inputs M
        by simp
    then obtain y q' where x-step: q}\mp@subsup{q}{}{\prime}\in\mathrm{ succ M (x,y)q
        using Cons(3,4) unfolding completely-specified.simps by blast
    then have path M ([(x,y)]|[q]) q^ length [q] = length [(x,y)]
        target ([(x,y)]|[q]) q= q
        by auto
    then have q}\mp@subsup{q}{}{\prime}\in\mathrm{ nodes M
        using Cons(4) by (metis FSM.nodes-target)
    then have LS in M q}\mp@subsup{q}{}{\prime}{xs}\not={
        using Cons.prems Cons.IH by auto
    then obtain ys where length xs = length ys }\wedge(xs|ys)\inLSM q
        by auto
    then obtain tr where path M ((xs|ys)| |r) q'^ length tr = length (xs|ys)
        by auto
    then have path M ([(x,y)] @ (xs|ys)| [q'] @ tr) q
                ^ length ([q] @ tr ) = length ([(x,y)]@ (xs | ys))
        by (simp add: FSM.path.intros(2) x-step)
    then have path M ((x#xs|y#ys)| |q'] @ tr) q^ length ([q] @ tr ) = length (x#xs|y#ys)
        by auto
    then have (x#xs|y#ys)\inLSMq
        by (metis language-state)
    moreover have length ( }x#xs)=\mathrm{ length (y#ys)
        by (simp add: <length xs = length ys }\wedgexs|ys\inLSM q'>
    ultimately have (x#xs|y#ys)\inLS in Mq{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 \inLS Sin MqV
shows map fst vs \inV
proof -
    have (map fst vs)|(map snd vs)=vs
        by auto
    then have (map fst vs)|(map snd vs)\inLS in MqV
        using assms by auto
    then show ?thesis by auto
qed
lemma language-state-for-inputs-empty :
    assumes [] }\in
    shows [] \inLS in MqV
proof -
    have [] E language-state-for-input Mq[] 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 \(i o \in\) language-state-for-input \(M q x s\)
shows take \(n\) io \(\in\) language-state-for-input \(M q\) (take \(n x s)\)
proof -
    obtain \(y s\) where \(i o=x s \| y s\) length \(x s=\) length \(y s x s \| y s \in\) language-state \(M q\)
        using assms by auto
    then obtain \(p\) where length \(p=\) length xs path \(M((x s|\mid y s) \| p) q\)
        by auto
    then have path \(M(\) take \(n((x s \| y s) \| p)) q\)
        by (metis FSM.path-append-elim append-take-drop-id)
    then have take \(n(x s \| y s) \in\) language-state \(M q\)
        by (simp add: <length \(p=\) length \(x s 〉<l e n g t h ~ x s=l e n g t h ~ y s 〉 l a n g u a g e-s t a t e ~ t a k e-z i p) ~\)
    then have (take \(n x s) \|(\) take \(n y s) \in\) language-state \(M q\)
        by (simp add: take-zip)
    have take \(n\) io \(=(\) take \(n x s) \|(\) take \(n\) ys)
        using \(\langle i o=x s \| y s\rangle\) take-zip by blast
    moreover have length (take \(n x s)=\) length (take \(n y s)\)
        by (simp add: <length \(x s=\) length \(y s\rangle\) )
    ultimately show ?thesis
        using <(take \(n x s) \|(\) take \(n y s) \in\) language-state \(M q\rangle\)
        unfolding language-state-for-input.simps by blast
qed
lemma language-state-for-inputs-prefix :
    assumes \(v s @ x s \in L_{i n} M 1\left\{v s^{\prime} @ x s^{\prime}\right\}\)
    and length \(v s=\) length \(v s^{\prime}\)
shows \(v s \in L_{i n} M 1\left\{v s^{\prime}\right\}\)
proof -
    have \(v s @ x s \in L\) M1
        using assms(1) by auto
    then have \(v s \in L M 1\)
        by (meson language-state-prefix)
    then have \(v s \in L_{i n} M 1\{\) map fst \(v s\}\)
        by (meson insertI1 language-state-for-inputs-map-fst)
    moreover have \(v s^{\prime}=\operatorname{map} f s t\) 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 \(L S_{i n} M q T 1 \cup L S_{i n} M q T 2=L S_{i n} M q(T 1 \cup T 2)\)
    unfolding language-state-for-inputs.simps by blast
lemma io-reduction-on-subset :
    assumes io-reduction-on M1 T M2
    and \(\quad T^{\prime} \subseteq T\)
shows io-reduction-on M1 \(T^{\prime}\) M2
proof (rule ccontr)
    assume \(\neg\) io-reduction-on M1 \(T^{\prime}\) M2
    then obtain \(x s^{\prime}\) where \(x s^{\prime} \in T^{\prime} \neg L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\}\)
    proof -
        have f1: \(\forall\) ps \(P\) Pa. \(\left(p s::\left({ }^{\prime} a \times{ }^{\prime} b\right)\right.\) list \() \notin P \vee \neg P \subseteq P a \vee p s \in P a\)
        by blast
        obtain pps :: ('a×'b) list set \(\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)\) list set \(\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)\) list where
            \(\forall x 0 x 1\). \((\exists v 2 . v 2 \in x 1 \wedge v 2 \notin x 0)=(p p s x 0 x 1 \in x 1 \wedge p p s x 0 x 1 \notin x 0)\)
        by moura
        then have f2: \(\forall P\) Pa. pps \(P a P \in P \wedge p p s P a P \notin P a \vee P \subseteq P a\)
            by (meson subsetI)
        have \(f 3: \forall p s f c A .\left(p s::\left({ }^{\prime} a \times{ }^{\prime} b\right)\right.\) list \() \notin L S_{i n} f\left(c::^{\prime} c\right) A \vee\) map fst \(p s \in A\)
```

```
    by (meson language-state-for-inputs-map-fst-contained)
    then have Lin M1 T'\subseteq Lin 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' \inT
    using assms(2) by blast
    have ᄀ io-reduction-on M1 T M2
    proof -
        have f1: \forall as. as & T T}\vee\\mathrm{ as }\in
        using assms(2) by auto
    obtain pps::(' }a\times\mp@subsup{}{}{\prime}b) list set => (' a > 'b) list set => (' a > 'b) list where
        \forall0 x1. (\existsv2.v2 \inx1^v2 & x0) = (pps x0 x1 \inx1^pps x0 x1 &x0)
        by moura
    then have }\forallPPa.(\negP\subseteqPa\vee(\forallps.ps\not\inP\veeps\inPa)
                                    \wedge(P\subseteqPa\vee pps Pa PGP^pps Pa P&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 => ('in,'out,'state) FSM => ('in \times 'out) list }=>\mathrm{ bool where
    sequence-to-failure M1 M2 xs = (
        (butlast xs) ( (language-state M2 (initial M2) \cap language-state M1 (initial M1))
        \wedgexs\in(language-state M1 (initial M1) - language-state M2 (initial M2)))
```

lemma sequence-to-failure-ob:
assumes $\neg M 1 \preceq M 2$
and well-formed M1
and well-formed M2
obtains io
where sequence-to-failure M1 M2 io
proof -
let ? diff $=\{$ io.$i o \in$ 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 $i o-d e f[\operatorname{simp}]:$ io $=\arg -m i n \operatorname{length}(\lambda$ io.$i o \in$ ?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 $i o$-split $[$ simp $]: i o=i o$-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: ᄀ (io-init \in language-state M1 (initial M1)
                    ^ io-init \in language-state M2 (initial M2))
    have io-init @ [io-last] \in language-state M1 (initial M1)
        using io-diff io-split by auto
    then have io-init \inlanguage-state M1 (initial M1)
        by (meson language-state language-state-split)
    moreover have io-init & language-state M2 (initial M2)
        using assm calculation by auto
    ultimately have io-init \in?diff
        by auto
    moreover have length io-init < length io
        using io-split by auto
    ultimately have io }\not=\operatorname{arg-min length ( }\lambda\mathrm{ io . io }\in\mathrm{ ?diff)
    proof -
        have \existsps.ps \in{ps\in language-state M1 (initial M1).
                            ps #language-state M2 (initial M2)} ^\neglength io }\leq\mathrm{ length ps
        using <io-init \in{io\in language-state M1 (initial M1). io & language-state M2 (initial M2)}
            <length io-init < length io> linorder-not-less
        by blast
        then show ?thesis
        by (meson arg-min-nat-le)
    qed
    then show False using io-def by simp
    qed
    then have sequence-to-failure M1 M2 io
    using io-split io-diff by auto
    then show ?thesis
    using that by auto
qed
lemma sequence-to-failure-succ :
    assumes sequence-to-failure M1 M2 io
    shows }\forallq\in\mathrm{ io-targets M2 (initial M2) (butlast io) . succ M2 (last io) q={}
proof
    have io \not= []
        using assms by auto
    fix q}\mathrm{ assume q}\in\mp@code{io-targets M2 (initial M2) (butlast io)
    then obtain tr where q= target (butlast io |tr) (initial M2)
                        and path M2 (butlast io || tr) (initial M2)
                        and length (butlast io) = length tr
    unfolding io-targets.simps by auto
    show succ M2 (last io) q={}
    proof (rule ccontr)
        assume succ M2 (last io) q\not={}
        then obtain q' where q}\mp@subsup{q}{}{\prime}\in\mathrm{ succ M2 (last io) q
            by blast
        then have path M2 [(last io, q')] (target (butlast io | tr) (initial M2))
            using <q = target (butlast io | tr) (initial M2)> by auto
        have path M2 ((butlast io | tr) @ [(last io, q}\mp@subsup{|}{}{\prime})]) (initial M2)
        using <path M2 (butlast io || tr) (initial M2)>
                            <path M2 [(last io, q')] (target (butlast io | tr) (initial M2))> by auto
        have butlast io @ [last io] = io
        by (meson`io \not= []` append-butlast-last-id)
        have path M2 (io || (tr@[q])) (initial M2)
        proof -
            have path M2 ((butlast io | tr) @ ([last io]| |q])) (initial M2)
            by (simp add: FSM.path-append <path M2 (butlast io || tr) (initial M2)>
                    <path M2 [(last io, q')] (target (butlast io | tr) (initial M2))>)
        then show ?thesis
```

```
            by (metis (no-types) <butlast io @ [last io]= io`
            <length (butlast io) = length tr>zip-append)
    qed
    have io \inL 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 \not=[]
proof
    assume xs = []
    then have xs\inL M1 \capL M2
        by auto
    then show False using assms by auto
qed
lemma sequence-to-failure-from-arbitrary-failure :
    assumes vs@xs \inLM1 - LM2
        and vs \inL M2 \capLM1
shows \exists xs''. prefix x\mp@subsup{s}{}{\prime}}xs^\mathrm{ ^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 \inL M1
        using snoc.prems(1) by (metis Diff-iff append.assoc language-state-prefix)
    show ?case
    proof (cases vs@xs \inL M2)
        case True
        have butlast (vs@xs@[x]) \inL M2 \cap L M1
            using True<vs @ xs \inL M1> by (simp add: butlast-append)
        then show ?thesis
            using sequence-to-failure.simps snoc.prems by blast
    next
        case False
        then have vs@xs \inLM1 - LM2
            using <vs@ xs \inL M1> by blast
        then obtain xs' where prefix x\mp@subsup{s}{}{\prime}}\mathrm{ xs sequence-to-failure M1 M2 (vs@xs')
            using snoc.prems(2) snoc.IH by blast
        then show ?thesis
        using prefix-snoc by auto
    qed
qed
```

The following lemma shows that if M 1 is not a reduction of M 2 , 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 \negM1 \preceqM2
```

```
shows \exists xs . sequence-to-failure M1 M2 xs ^ 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 }\not=[
    by auto
let ?bls = butlast seq
have ?bls \inL M1 ?bls \inL M2
    using<sequence-to-failure M1 M2 seq〉 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<length ?bls = length tr2b><length tr2b = length tr1b>,
    of M2 M1 initial M2 initial M1]
    using <path M1 (butlast seq || tr1b) (initial M1)>
        <path M2 (butlast seq | tr2b) (initial M2)>
    by blast
```

let ?q1b $=$ target $(? b l s \|$ tr1b $)($ initial M1)
let ?q2b $=$ target $(? b l s \|$ tr2b) (initial M2)
have io-targets M2 (initial M2) ?bls $=\{? q 2 b\}$
by (metis 〈length (butlast seq) = length tr2b〉〈path M2 (butlast seq \| tr2b) (initial M2)〉
$\operatorname{assms}(4)$ obs-target-is-io-targets)
have io-targets M1 (initial M1) ?bls $=\{? q 1 b\}$
by (metis 〈length tr1b = length (butlast seq)〉〈path M1 (butlast seq\| tr1b) (initial M1)
assms(3) obs-target-is-io-targets)
have $(? q 2 b, ? q 1 b) \in$ reachable (product M2 M1) (initial M2, initial M1)
proof -
have target (butlast seq || tr2b || tr1b) (initial M2, initial M1)
$\in$ reachable (product M2 M1) (initial M2, initial M1)
using 〈path (product M2 M1) (butlast seq \| tr2b \| tr1b) (initial M2, initial M1) > by blast
then show ?thesis
using 〈length (butlast seq) = length tr2b〉〈length tr2b $=$ length tr1b〉by auto
qed
have（initial M2，initial M1）$\in$ nodes $($ 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 〈well－formed？？ • 〉
$\langle(? q 2 b, ? q 1 b) \in$ reachable（product M2 M1）（initial M2，initial M1）〉

```
<(initial M2, initial M1) \in nodes (product M2 M1)>]
```

by blast
then have set $($ states $p($ initial M2，initial M1）$) \subseteq$ nodes ？$P$ PM
by（simp add：FSM．nodes－states $\langle($ initial M2，initial M1）$\in$ nodes（product M2 M1）））
moreover have（initial M2，initial M1）$\notin$ 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）$) \subseteq$ 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）$\in$ nodes（product M2 M1）〉
〈（initial M2，initial M1）$\notin$ set（states $p$（initial M2，initial M1））〉
〈set $($ states $p($ initial M2，initial M1）$) \subseteq$ 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））$<|? P M|$
by linarith
then have length $p<\mid$ ？PM
by auto
let ？$p 1=\operatorname{map}($ snd $\circ$ snd $) p$
let ？$p 2=\operatorname{map}(f s t \circ s n d) p$
let $? p I O=$ map fst $p$
have $p=? p I O\|? p 2\| ? p 1$
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 $\langle p=$ ？pIO｜｜？p2｜｜？p1〉 repFreePath（1）by auto
have $(? q 2 b, ? q 1 b)=($ target $(? p I O\|? p 2\| ? p 1)($ initial M2，initial M1 $))$
using $\langle p=$ ？pIO｜｜？p2｜｜？p1 repFreePath（1）by auto
then have ？q2b $=$ target $(? p I O \| ? p 2)($ initial M2）
？q1b $=\operatorname{target}(? p I O \| ? p 1)($ 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 (?bls | tr2b) (initial M2) = target (map fst p| map (fst\circ 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 (?bls | tr1b) (initial M1) = target (map fst p| map (snd o snd) p) (initial M1)>
        assms(3) length-map obs-target-is-io-targets)
```

have seq $\in L$ M1 seq $\notin L$ M2
using «sequence-to-failure M1 M2 seq〉 by auto
have io-targets M1 (initial M1) ?bls $=\{? q 1 b\}$
by (metis 〈length tr1b = length (butlast seq)〉〈path M1 (butlast seq || tr1b) (initial M1) >
assms(3) obs-target-is-io-targets)
obtain $q 1 s$ where io－targets M1（initial M1）seq $=\{q 1 s\}$
by（meson $\langle s e q \in L$ M1〉assms（3）io－targets－observable－singleton－ob）

```
moreover have seq \(=(\) butlast seq \() @[\) last seq]
    using \(\langle s e q \neq[]\) by auto
ultimately have io-targets M1 (initial M1) ((butlast seq)@[last seq]) \(=\{q 1 s\}\)
    by auto
```

have io-targets M1 ?q1b [last seq] $=\{q 1 s\}$
using observable-io-targets-suffix[OF assms(3) 〈io-targets M1 (initial M1) ?bls = \{?q1b\}〉
〈io-targets M1 (initial M1) $(($ butlast seq $) @[$ last seq] $)=\{q 1 s\}\rangle]$ by assumption
then obtain $\operatorname{tr} 1 s$ where $q 1 s=\operatorname{target}([$ last seq] \| tr1s) $? q 1 b$
path M1 ([last seq] || tr1s) ? $q 1 b$
length $[$ last $\operatorname{seq}]=$ length tr1s
by auto
have path M1 ([last seq] || [q1s]) ?q1b
by (metis (no-types) 〈length [last seq] $=$ length $\operatorname{tr} 1 s\rangle$
<path M1 ([last seq] || tr1s) (target (butlast seq \| tr1b) (initial M1)) >
«q1s = target $([$ last seq] || tr1s) (target (butlast seq || tr1b) (initial M1)) $>$
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) $? q 2 b=\{ \}$
proof (rule ccontr)
assume succ M2 (last seq) (target (butlast seq \|| tr2b) (initial M2)) $\neq\{ \}$
then obtain $q 2 f$ where $q 2 f \in$ succ M2 (last seq) ? $q 2 b$
by blast
then have target $([$ last seq] \| [q2f]) ? $q 2 b=q 2 f$
path M2 ([last seq] || [q2f]) ?q2b
length $[q 2 f]=$ length $[$ last seq]
by auto
then have $q 2 f \in$ io-targets M2 ? $q 2 b$ [last seq]
by (metis io-target-from-path)
then have io-targets M2 ? $q 2 b[$ last seq] $=\{q 2 f\}$
using assms(4) by (meson observable-io-target-is-singleton)
have io-targets M2 (initial M2) (butlast seq @ [last seq]) $=\{q 2 f\}$
using observable-io-targets-append $[$ OF assms(4)〈io-targets M2 (initial M2) $? b l s=\{? q 2 b\}\rangle$
$\langle i o-$ targets M2 ?q2b [last seq] $=\{q 2 f\}\rangle$ by assumption
then have seq $\in L$ M2
using $<$ seq = butlast seq @ [last seq] >by auto
then show False
using «seq $\notin L$ M2 〉 by blast
qed
have ?pIO $\in L$ M1 ? $p I O \in L$ M2
using <path M1 (?pIO || ?p1) (initial M1)〉〈path M2 (?pIO || ?p2) (initial M2)> by auto
then have butlast $(? p I O @[$ last seq] $) \in L M 1 \cap L$ M2
by auto
have ?pIO@[last seq] $\in L$ M1
using observable-io-targets-append[OF assms(3) «io-targets M1 (initial M1) ?pIO = \{?q1b\}〉
$\langle$ io-targets M1 ? $q 1 b$ [last seq] $=\{q 1 s\}\rangle]$
by (metis all-not-in-conv insert-not-empty io-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-targets M2 (initial M2) (?pIO@[last seq]) $=\{q 2 f\}$
by (meson assms(4) io-targets-observable-singleton-ob)

```
have io-targets M2 ?q2b [last seq] = {q2f}
    using observable-io-targets-split[OF assms(4)
        <io-targets M2 (initial M2) (?pIO@[last seq]) = {q2f}>
        <io-targets M2 (initial M2) (map fst p)={?q2b}`] by assumption
    then have q2f \in succ M2 (last seq) ?q2b
    by (simp add: io-targets-succ)
    then show False
    using <succ M2 (last seq) ?q2b = {}` by auto
qed
ultimately have ?pIO@[last seq] \inL M1 - L M2
    by auto
have sequence-to-failure M1 M2 (?pIO@[last seq])
    using <butlast(?pIO@[last seq]) \inL M1 \cap L M2`<?pIO@[last seq] \inL M1 - L M2> by auto
have length (?pIO@[last seq])= Suc (length ?pIO)
    by auto
then have length(?pIO@[last seq]) \leq |?PM 
    using <length p< |?PM|> by auto
have card (nodes M2 }\times\mathrm{ nodes M1) }\leq|M2|*|M1
    by (simp add: card-cartesian-product)
have finite (nodes M2 }\times\mathrm{ 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<card (nodes M2 }\times\mathrm{ nodes M1) }\leq|M2|* |M1|〉<finite (nodes M2 × nodes M1)>
        card-mono dual-order.trans product-nodes)
    then have length(?pIO@[last seq])\leq |M2 | * MM1 
    using <length (?pIO@[last seq])\leq \?PM > by auto
    then have sequence-to-failure M1 M2 (?pIO@[last seq]) ^ length (?pIO@[last seq])\leq |M2|* |M1 |
    using <sequence-to-failure M1 M2 (?pIO@[last seq])> by auto
    then show ?thesis
    by blast
qed
```


### 1.11 Minimal sequence to failure extending

A minimal sequence to a failure extending some 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 \(v^{\prime}\) io \(=(\)
    \(v^{\prime} \in L_{i n} M 1 V \wedge\) sequence-to-failure M1 M2 ( \(v^{\prime}\) @ io)
            \(\wedge \neg\left(\exists i o^{\prime} . \exists w^{\prime} \in L_{i n} M 1 V\right.\). sequence-to-failure M1 M2 ( \(\left.w^{\prime} @ i o^{\prime}\right)\)
                                \(\wedge\) length io \({ }^{\prime}<\) 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 \prec 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.\existsv\mp@subsup{s}{}{\prime}\in\mp@subsup{L}{in}{}M1 V. 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 Lin 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 = arg-min length ( }\lambdaxs.xs\in\mathrm{ ?exts)
    have xsMin-def:?xsMin \in?exts
                \wedge (\forallxs\in ?exts.length ?xsMin \leq length xs)
        by (metis (no-types, lifting) <stf \in?exts` arg-min-nat-lemma)
    then obtain vs where vs \inL Lin M1 V
                                    ^ sequence-to-failure M1 M2 (vs @ ?xsMin)
        by blast
    moreover have }\neg(\existsxs.\exists\mathrm{ ws }\in\mp@subsup{L}{in}{}M1 V. sequence-to-failure M1 M2 (ws@xs
                        ^ length xs < 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 (map fst vs) \inV
proof -
    have vs \inL Lin 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 = inputs M1
    and }\quad|M1|\leq
shows \negM1\preceqM2 \longrightarrow\negM1\preceq\llbracket{xs. set xs\subseteq inputs M2 ^ length xs \leq |M2 * m}\rrbracketM2
    (is \negM1 \preceqM2\longrightarrow ᄀM1 \preceq\llbracket?TS\rrbracketM2)
proof
    assume \negM1 \preceq M2
    obtain stf where sequence-to-failure M1 M2 stf ^ 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 \leq |M2 | * |M1 
    by auto
then have stf }\inLM
    by auto
let ?xs = map fst stf
have set ?xs \subseteq inputs M1
    by (meson <stf \inL M1`assms(1) language-state-inputs)
then have set ?xs \subseteq inputs M2
    using assms(5) by auto
have length ?xs \leq |M2 | * |M1 
    using <length stf }\leq|M2|*|M1|> by aut
have length ?xs \leq |M2 | *m
proof -
    show ?thesis
        by (metis (no-types)<length (map fst stf) \leq |M2 | * |M1|><|M1 | mm>
            dual-order.trans mult.commute mult-le-mono1)
qed
have stf}\in\mp@subsup{L}{in}{}M1{?xs
    by (meson <stf \inL M1` insertI1 language-state-for-inputs-map-fst)
have ?xs \in ?TS
    using<set ?xs \subseteq inputs M2`<length ?xs }\leq|M2|*m> by blas
have stf \in Lin M1 ?TS
    by (metis (no-types, lifting)<map fst stf \in{xs. set xs \subseteq inputs M2 ^ length xs \leq |M2 | * m}>
            stf \inL M1> language-state-for-inputs-map-fst)
have stf # L M2
    using <sequence-to-failure M1 M2 stf〉 by auto
then have stf \not\inL Lin M2 ?TS
    by auto
show ᄀM1 \preceq\llbracket?TS\rrbracketM2
    using <stf \inLLin M1 ?TS〉<stf }\not\in\mp@subsup{L}{in}{}\mathrm{ 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 }\quad|M1|\leq
shows M1 \preceqM2 \longleftrightarrowM1 \preceq\llbracket{xs. set xs \subseteq inputs M2 ^ length xs \leq |M2 | * m}\rrbracketM2
    (is M1 \preceq M2 \longleftrightarrowM1 \preceq\llbracket?TS\rrbracketM2)
proof
    show M1 \preceqM2 \LongrightarrowM1 \preceq\llbracket?TS\rrbracketM2 - soundness holds trivially
        unfolding language-state-for-inputs.simps io-reduction.simps by blast
    show M1 \preceq\llbracket?TS\rrbracketM2 \Longrightarrow M1 \preceq 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 $A B$ is the product machine of A and B extended with fail state FAIL.

```
fun productF :: ('in, 'out, 'state1) FSM \(\Rightarrow\) ('in, 'out, 'state2) \(F S M \Rightarrow\) ('state1 \(\times\) 'state2)
    \(\Rightarrow\) ('in, 'out, 'state1 \(\times\) 'state2) \(F S M \Rightarrow\) bool where
    productF \(A B\) FAIL \(A B=(\)
        (inputs \(A=\) inputs \(B\) )
    \(\wedge(\) fst FAIL \(\notin\) nodes \(A)\)
    \(\wedge(\) snd \(F A I L \notin\) nodes \(B)\)
    \(\wedge A B=0\)
                succ \(=(\lambda a(p 1, p 2) \cdot(i f(p 1 \in\) nodes \(A \wedge p 2 \in\) nodes \(B \wedge(f s t a \in\) inputs \(A)\)
                            \(\wedge(\) snd \(a \in\) outputs \(A \cup\) outputs \(B))\)
                                    then (if (succ A a p1=\{\}^succ B a \(p 2 \neq\{ \}\) )
                                    then \(\{F A I L\}\)
                            else (succ A a p1 \(\times\) succ B a p2))
                            else \{\})),
            inputs \(=\) inputs \(A\),
            outputs \(=\) outputs \(A \cup\) outputs \(B\),
            initial \(=(\) initial \(A\), initial \(B)\)
            D)
lemma productF-simps[simp]:
    productF \(A\) B FAIL \(A B \Longrightarrow\) succ \(A B a(p 1, p 2)=(\) if \((p 1 \in\) nodes \(A \wedge p 2 \in\) nodes \(B\)
                    \(\wedge\left(f_{\text {st }} a \in\right.\) inputs \(\left.A\right) \wedge(\) snd \(a \in\) outputs \(A \cup\) outputs \(\left.B)\right)\)
                    then (if (succ A a \(p 1=\{ \} \wedge\) succ \(B\) a \(p \mathbf{2} \neq\{ \}\) )
                        then \(\{F A I L\}\)
                        else (succ A a p1 \(\times\) succ \(B\) a \(p 2\) ))
                else \{\})
    productF \(A B\) FAIL \(A B \Longrightarrow\) inputs \(A B=\) inputs \(A\)
    productF \(A B\) FAIL \(A B \Longrightarrow\) outputs \(A B=\) outputs \(A \cup\) outputs \(B\)
    productF A B FAIL \(A B \Longrightarrow\) initial \(A B=(\) 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 \(((f s t F A I L) \in\) 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) \(=\{ \} \wedge(f s t a \in\) 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 $P M \subseteq$ insert FAIL (nodes M2 $\times$ nodes M1)
proof
fix $q$ assume $q$-assm : $q \in$ nodes $P M$
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-a-split[simp]:p=(p1,p2)
\[
\begin{aligned}
& a=((x, y), q) \\
& q=(q 1, q 2)
\end{aligned}
\]
by (metis eq-snd-iff)
have subnodes : p1 \(\in\) nodes M2 \(\wedge p 2 \in\) nodes \(M 1 \wedge x \in\) inputs \(M 2 \wedge y \in\) outputs \(M 2 \cup\) outputs \(M 1\) proof (rule ccontr)
assume \(\neg(p 1 \in\) nodes M2 \(\wedge p 2 \in\) nodes \(M 1 \wedge x \in\) inputs \(M 2 \wedge y \in\) outputs M2 \(\cup\) outputs M1)
then have succ \(P M(x, y)(p 1, p 2)=\{ \}\)
using assms(3) by auto
then show False
using execute by auto
qed
show ?thesis proof (cases (succ M2 \((x, y) p 1=\{ \} \wedge \operatorname{succ} M 1(x, y) p 2 \neq\{ \}))\)
case True
then have \(q=\) FAIL
using subnodes assms(3) execute by auto
then show ?thesis
by auto
next
case False
then have succ \(P M(f s t a) p=\operatorname{succ} M 2(x, y) p 1 \times \operatorname{succ} M 1(x, y) p 2\) using subnodes assms(3) execute by auto
then have \(q \in(\operatorname{succ} M 2(x, y) p 1 \times\) succ M1 \((x, y) p\) 2) using execute by blast
then have \(q\)-succ : \((q 1, q 2) \in(\) succ M2 \((x, y) p 1 \times \operatorname{succ} M 1(x, y) p 2)\) by \(\operatorname{simp}\)
have \(q 1 \in \operatorname{succ}\) M2 \((x, y) p 1\) using \(q\)-succ by simp
then have \(q 1 \in\) successors M2 \(p 1\) by auto
then have \(q 1 \in\) reachable M2 \(p 1\) by blast
then have \(q 1 \in\) reachable M2 (initial M2) using subnodes by blast
then have nodes1 : q1 \(\in\) nodes M2 by blast
have \(q 2 \in \operatorname{succ} M 1(x, y) p 2\) using \(q\)-succ by simp
then have \(q 2 \in\) successors M1 p2 by auto
then have \(q 2 \in\) reachable M1 p2 by blast
then have \(q 2 \in\) 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 }\times\mathrm{ nodes M1))
        by simp
    moreover have nodes PM\subseteq insert FAIL (nodes M2 }\times\mathrm{ nodes M1)
    using nodes-productF assms by blast
    moreover have inputs PM = inputs M2 outputs PM = outputs M2 \cup 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 \cup outputs M1
        using assms by auto
    then show ( }\forall\mathrm{ s1 x y. x & inputs PM V y# outputs PM }\longrightarrow\operatorname{succ}PM(x,y) s1={}
                        ^ inputs PM\not={}^ outputs PM }\not={
        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 \forallt s.succ M1 t(fst s)={}\vee(\exists s2. succ M1 t (fst s)={s2})
        using assms by auto
    moreover have }\forallts.succ M2 t(snd s)={}\vee(\exists s2. succ M2 t (snd s)={s2})
        using assms by auto
    ultimately have sub-succs: }\forallts.\operatorname{succ}M2t(fst s)\times succ M1 t (snd s)={
                    \vee \mp@code { ( \exists ~ s 2 ~ . ~ s u c c ~ M 2 ~ t ~ ( f s t ~ s ) ~ × ~ s u c c ~ M 1 ~ t ~ ( s n d ~ s ) ~ = \{ s 2 \} ) }
        by fastforce
    moreover have succ-split: }\forallts.\mathrm{ succ PM t s={}
                    vucc PM t s}={FAIL
                    vucc PM t s = succ M2 t (fst s) }\times\mathrm{ succ M1 t (snd s)
        using assms by auto
    ultimately show }\forallt\mathrm{ s. succ PM ts={}}\vee(\exists s2. succ PM t s={s2}
        by metis
qed
```

lemma no-transition-after-FAIL :
assumes productF A B FAIL AB
shows succ $A B$ io FAIL $=\{ \}$
using assms by auto
lemma no-prefix-targets-FAIL :
assumes productF M2 M1 FAIL PM
and path $P M p q$
and $\quad k<$ length $p$
shows target (take $k$ p) $q \neq$ FAIL
proof
assume assm: target (take kp) $q=$ FAIL
have path PM (take kp@drop $k$ p) q using assms by auto
then have path PM (drop kp) (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 $k p) \neq 0$ using assms by auto
then obtain io $q$ where drop $k p=(i o, q) \#(d r o p(S u c k) p)$ by (metis Cons-nth-drop-Suc assms(3) prod-cases3)
then have succ $P M$ io FAIL $\neq\{ \}$
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 $r 1$ length $r 1=$ length $r 2$
and productF A B FAIL AB
and well-formed $A$
and well-formed $B$
and path $A(w \| r 1) p 1 \wedge$ path $B(w \| r 2) p 2$
and $\quad p 1 \in$ nodes $A$
and $\quad p 2 \in$ nodes $B$
shows path $(A B)(w\|r 1\| r 2)(p 1, p 2)$
using assms proof (induction wr1 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 $A([w] \|[r 1]) p 1 \wedge$ path $B([w] \|[r 2]) p 2$
by auto
then have succs : r1 $\in$ succ $A w p 1 \wedge r 2 \in \operatorname{succ} B w p 2$
by auto
then have succ $A$ w $p 1 \neq\{ \}$
by force
then have $w$-elem : fst $w \in$ inputs $A \wedge$ snd $w \in$ outputs $A$
using Cons by (metis assms(4) prod.collapse well-formed.elims(2))
then have $(r 1, r 2) \in \operatorname{succ} A B w(p 1, p 2)$
using Cons succs by auto
then have path-head : path $A B([w] \|[(r 1, r 2)])(p 1, p 2)$
by auto
have path $A(w s \| r 1 s) r 1 \wedge$ path $B(w s \| r 2 s) r 2$
using Cons by auto
moreover have r1 $\in$ nodes $A \wedge r 2 \in$ nodes $B$
using succs Cons.prems succ-nodes[of r1A wipl] succ-nodes[of r2 B w p2] by auto
ultimately have path $A B(w s\|r 1 s\| r 2 s)(r 1, r 2)$
using Cons by blast
then show? case
using path-head by auto
qed
lemma productF-path-forward:
assumes length $w=$ length $r 1$ length $r 1=$ length $r 2$
and productF A B FAIL AB
and well-formed $A$
and well-formed $B$
and $\quad($ path $A(w \| r 1) p 1 \wedge$ path $B(w \| r 2) p 2)$
$\vee(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2)=F A I L$
$\wedge$ length $w>0$
$\wedge$ path $A$ (butlast $(w \| r 1)) p 1$
$\wedge$ path B (butlast ( $w \| r 2$ ) $)$ p2
$\wedge$ succ $A($ last $w)($ target $($ butlast $(w \| r 1)) p 1)=\{ \}$
$\wedge \operatorname{succ} B($ last $w)($ target $($ butlast $(w \| r 2)) p 2) \neq\{ \})$
and $\quad p 1 \in$ nodes $A$
and $\quad p 2 \in$ nodes $B$
shows path $(A B)(w\|r 1\| r 2)(p 1, p 2)$
using assms proof (induction wr1 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,p\mathscr{2})=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.prems 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, p\mathscr{2})=FAIL ^
                0< length ([w])^
                path A [] p1^
                path B [] p2 ^
                succ A wp1={}^
                succ B w p2 \not={}
        using fail-prop by auto
    then have fst w\in inputs }B\wedge\mathrm{ snd w}\in\mathrm{ outputs }
        using Cons.prems by (metis prod.collapse well-formed.elims(2))
    then have inputs-0 : fst w\in inputs }A\wedge\mathrm{ snd w}\in\mathrm{ outputs B
        using Cons.prems 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.prems by auto
    then have path AB([w]|[r1]|[r2]) (p1,p\mathcal{Q})
        using Cons.prems 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]|[r\mathscr{L}) 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||2s)) 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 \in nodes A
```

using Cons.prems path-r1 by (metis FSM.path-cons-elim snd-conv succ-nodes zip-Cons-Cons)
moreover have $r 2 \in$ nodes $B$
using Cons.prems 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) $\neq\{ \}$
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 $A B(w s\|r 1 s\| r 2 s)(r 1, r 2)$
using Cons.IH Suc 〈succ B (last ws) (target (butlast (ws \| r2s)) r2) $\neq\{ \}\rangle$ assms(3) assms(4) assms(5) calculation(1-4) path-r1s path-r2s zero-less-Suc
by presburger
moreover have path $A B([w]\|[r 1]\|[r 2])(p 1, p 2)$
using path-r1 path-r2 productF-path-inclusion[of [w] [r1] [r2] A B FAIL AB p1 p2]
Cons.prems
by auto
ultimately show ?thesis
by auto
qed
qed
qed

```
lemma butlast-zip-cons: length ws \(=\) length \(r 1 s \Longrightarrow w s \neq[]\)
    \(\Longrightarrow\) butlast \((w \#\) ws \(\| r 1 \# r 1 s)=((w, r 1) \#(\) butlast \((w s \| r 1 s)))\)
proof -
assume a1: length \(w s=\) length \(r 1 s\)
assume a2: ws \(\neq[]\)
    have length \((w \# w s)=\) length \(r 1 s+\) Suc 0
        using a1 by (metis list.size(4))
    then have f3: length \((w \#\) ws \()=\) length ( \(r 1 \#\) r1s \()\)
        by (metis list.size(4))
    have \(f_{4}\) : ws @ \(w \# w s \neq w \# w s\)
        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 \(=\mathrm{w} \# \mathrm{ws} \| \mathrm{r} 1 \# \mathrm{r} 1 \mathrm{~s}\)
        using \(f_{4}\) f3 by (meson zip-eq)
    then show ?thesis
        using \(a 1\) by simp
qed
```

lemma productF-succ-fail-imp :
assumes productF $A$ B FAIL $A B$
and $\quad F A I L \in \operatorname{succ} A B w(p 1, p 2)$
and well-formed $A$
and well-formed $B$
shows $p 1 \in$ nodes $A \wedge p 2 \in$ nodes $B \wedge($ fst $w \in$ inputs $A) \wedge($ snd $w \in$ outputs $A \cup$ outputs $B)$
$\wedge$ succ $A B w(p 1, p 2)=\{F A I L\} \wedge$ succ $A w p 1=\{ \} \wedge$ succ $B w p 2 \neq\{ \}$
proof -
have path-head : path $A B([w] \|[F A I L])(p 1, p 2)$
using assms by auto
then have succ-nonempty: succ $A B w(p 1, p 2) \neq\{ \}$
by force
then have succ-if-1:p1 $\in$ nodes $A \wedge p 2 \in$ nodes $B \wedge($ fst $w \in$ inputs $A)$
$\wedge($ snd $w \in$ outputs $A \cup$ outputs $B)$
using assms by auto
then have $(p 1, p 2) \neq F A I L$
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 \(B w p 2 \subseteq\) nodes \(B\)
    using assms succ-if-1 by (simp add: subsetI succ-nodes)
    ultimately have FAIL \(\notin(\) succ \(A\) wp1 \(\times\) succ \(B\) w \(p\) ) \()\)
    using assms by auto
    then have succ-no-inclusion : succ \(A B w(p 1, p 2) \neq(\operatorname{succ} A w p 1 \times \operatorname{succ} B w p 2)\)
    using assms succ-if-1 by blast
    moreover have succ \(A B w(p 1, p 2)=\{ \} \vee \operatorname{succ} A B w(p 1, p 2)=\{F A I L\}\)
                \(\vee\) succ \(A B w(p 1, p 2)=(\operatorname{succ} A w p 1 \times \operatorname{succ} B w p 2)\)
    using assms by simp
    ultimately have succ-fail : succ \(A B w(p 1, p 2)=\{F A I L\}\)
    using succ-nonempty by simp
    have succ \(A\) w p1 \(=\{ \} \wedge\) succ \(B w p 2 \neq\{ \}\)
    proof (rule ccontr)
    assume \(\neg(\) succ \(A w p 1=\{ \} \wedge\) succ \(B w p 2 \neq\{ \})\)
    then have succ \(A B w(p 1, p 2)=(\) succ \(A w p 1 \times\) succ \(B w p 2)\)
        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 $r 1$ length $r 1=$ length $r 2$
and productF A B FAIL AB
and well-formed $A$
and well-formed $B$
and $\quad p a t h ~ A B(w\|r 1\| r 2)(p 1, p 2)$
and $\quad p 1 \in$ nodes $A$
and $\quad p 2 \in$ nodes $B$
shows (path $A(w \| r 1) p 1 \wedge$ path $B(w \| r 2) p 2)$
$\vee(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2)=F A I L$
$\wedge$ length $w>0$
$\wedge$ path $A($ butlast $(w \| r 1)) p 1$
$\wedge$ path $B$ (butlast $(w \| r 2)) p 2$
$\wedge$ succ $A($ last $w)($ target $($ butlast $(w \| r 1)) p 1)=\{ \}$
$\wedge$ succ $B($ last $w)($ target $($ butlast $(w \| r 2)) p 2) \neq\{ \})$
using assms proof (induction wr1 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 $A B([w] \|[(r 1, r 2)])(p 1, p 2)$ using Cons by auto
then have succ-nonempty: succ $A B w(p 1, p 2) \neq\{ \}$ by force
then have succ-if-1:p1 $\in$ nodes $A \wedge p 2 \in$ nodes $B \wedge($ fst $w \in$ inputs $A)$
$\wedge($ snd $w \in$ outputs $A \cup$ outputs $B)$
using Cons by fastforce
then have $(p 1, p 2) \neq F A I L$
using Cons by auto
have path-tail: path $A B(w s\|r 1 s\| r 2 s)(r 1, r 2)$
using path-head Cons by auto
show ?case
proof $($ cases $(r 1, r 2)=F A I L)$
case True
have $r 1 s=[]$
proof (rule ccontr)

```
    assume \(\neg(r 1 s=[])\)
    then have \((\neg(w s=[])) \wedge(\neg(r 1 s=[])) \wedge(\neg(r 2 s=[]))\)
        using Cons.hyps by auto
    moreover have path \(A B\) (ws || r1s || r2s) FAIL
    using True path-tail by simp
    ultimately have path \(A B([h d w s]\) @ \(t l w s \|[h d r 1 s]\) @ \(t l r 1 s \|[h d r 2 s]\) @ \(t l r 2 s) F A I L\)
    by simp
    then have path \(A B([h d w s]\|[h d r 1 s]\|[h d r 2 s])\) FAIL
        by auto
    then have succ \(A B\) (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 \(=[] \wedge r 1 s=[] \wedge r 2 s=[]\)
    using Cons.hyps by simp
    have succ-fail : FAIL \(\in\) succ \(A B w(p 1, p 2)\)
    using path-head True by auto
    then have succs: succ \(A\) wp1 \(=\{ \} \wedge\) succ \(B w p 2 \neq\{ \}\)
    using Cons.prems by (meson productF-succ-fail-imp)
    have \(\operatorname{target}(w \# w s\|r 1 \# r 1 s\| r 2 \# r 2 s)(p 1, p 2)=F A I L\)
    using True tail-nil by simp
    moreover have \(0<\) length ( \(w\) \# ws)
        by \(\operatorname{simp}\)
    moreover have path \(A\) (butlast ( \(w\) \# ws || r1 \# r1s)) p1
        using tail-nil by auto
    moreover have path B (butlast ( \(w\) \# ws || r2 \# r2s)) p2
        using tail-nil by auto
    moreover have succ \(A(\operatorname{last}(w \#\) ws \())(\operatorname{target}(\) butlast \((w \# w s \| r 1 \# r 1 s)) p 1)=\{ \}\)
        using succs tail-nil by simp
    moreover have succ \(B(\operatorname{last}(w \#\) ws \())(\operatorname{target}(\) butlast \((w \# w s \| r 2 \# r 2 s)) p 2) \neq\{ \}\)
        using succs tail-nil by simp
    ultimately show ?thesis
        by \(\operatorname{simp}\)
next
    case False
    have \((r 1, r 2) \in \operatorname{succ} A B w(p 1, p 2)\)
        using path-head by auto
    then have succ-not-fail: succ \(A B w(p 1, p 2) \neq\{F A I L\}\)
        using succ-nonempty False by auto
    have \(\neg(\) succ A wp1 \(=\{ \} \wedge\) succ \(B\) w p2 \(\neq\{ \})\)
    proof (rule ccontr)
    assume \(\neg \neg(\) succ \(A\) w p1 \(=\{ \} \wedge\) succ \(B w p 2 \neq\{ \})\)
    then have succ \(A B w(p 1, p 2)=\{\) FAIL \(\}\)
        using succ-if-1 Cons by auto
    then show False
        using succ-not-fail by simp
qed
then have succ \(A B w(p 1, p 2)=(\) succ \(A w p 1 \times s u c c B w p 2)\)
    using succ-if-1 Cons by auto
then have \((r 1, r 2) \in(s u c c A w p 1 \times s u c c B w p 2)\)
    using Cons by auto
then have succs-next : r1 \(\in \operatorname{succ} A w p 1 \wedge r 2 \in \operatorname{succ} B w p 2\)
        by auto
    then have nodes-next : r1 \(\in\) nodes \(A \wedge r 2 \in\) nodes \(B\)
        using Cons succ-nodes by metis
    moreover have path-tail : path \(A B(w s\|r 1 s\| r 2 s)(r 1, r 2)\)
        using Cons by auto
ultimately have prop-tail :
```

```
    path A (ws| r1s) r1 ^ path B (ws| r2s)r2 \vee
    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.prems 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)r\mathcal{Z})
    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]) p\mathscr{Z}
        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)) r\mathcal{Z})
        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)={}
                    \wedge 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 \(x s=\) length \(y s\)
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 \(r 1\) length \(r 1=\) length \(r 2\)
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
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)
                \vee (target }(w|r1|r2)(p1,p2)=FAI
                ^ length w>0
        ^ path A (butlast (w|r1)) p1
        \wedge path B (butlast (w|r2)) p2
        ^ succ A (last w) (target (butlast (w|r1)) p1)={}
        \wedge 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 \existsr1'. 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
                        \wedge 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 \(r 1\) length \(r 1=\) length \(r 2\)
    and productF A B FAIL AB
    and well-formed \(A\)
    and well-formed \(B\)
    and \(\quad p 1 \in\) nodes \(A\)
    and \(\quad p 2 \in\) nodes \(B\)
shows path \(A B(w\|r 1\| r 2)(p 1, p 2) \longleftrightarrow((p a t h ~ A(w \| r 1) p 1 \wedge\) path \(B(w \| r 2) p 2)\)
        \(\vee(\) target \((w\|r 1\| r 2)(p 1, p 2)=F A I L\)
            \(\wedge\) length \(w>0\)
            \(\wedge\) path A (butlast \((w|\mid r 1))\) p1
            \(\wedge\) path \(B\) (butlast \((w \| r 2)) p 2\)
            \(\wedge\) succ \(A(\) last \(w)(\) target \((\) butlast \((w \| r 1)) p 1)=\{ \}\)
```


## proof

assume ?path
then show ?paths using assms productF-path-reverse[of wr1 r2 A B FAIL AB p1 p2] by simp next
assume ?paths
then show ?path using assms productF-path-forward[of wr1 r2 A B FAIL AB p1 p2] by simp qed
lemma path-last-succ:
assumes path $A(w s \| r 1 s) p 1$
and length r1s $=$ length ws
and length ws $>0$
shows last r1s $\in \operatorname{succ} A($ last ws) $($ target $($ butlast $(w s \| r 1 s)) p 1)$
proof -
have path $A$ (butlast (ws \|r1s)) p1
$\wedge$ path $A[$ last (ws \| r1s)] (target (butlast (ws \| 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 snd $($ last $(w s \| r 1 s)) \in$
succ $A(f s t($ last $(w s \| r 1 s)))($ target $($ butlast $(w s \| r 1 s)) p 1)$
by auto
moreover have ws || r1s $\neq[]$
using $\operatorname{assms}(3) \operatorname{assms}(2)$ by (metis length-zip list.size(3) min.idem neq0-conv)
ultimately have last r1s $\in \operatorname{succ} A($ last ws) $($ target $($ butlast $(w s \| r 1 s)) p 1)$
by (simp add: assms(2))
then show ?thesis
by auto
qed
lemma zip-last :
assumes length $r 1>0$
and length $r 1=$ length $r 2$
shows last $(r 1 \| r 2)=($ last r1, 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 length $w=$ length $r 1$ length $r 1=$ length $r 2$
and productF A B FAIL AB
and well-formed $A$
and well-formed $B$
and $\quad$ path $A B(w\|r 1\| r 2)(p 1, p 2)$
and $\quad p 1 \in$ nodes $A$
and $\quad p 2 \in$ nodes $B$
and $\quad w \in$ language-state $A p 1$
and observable $A$
shows path $A(w \| r 1) p 1 \wedge$ length $w=$ length $r 1$ path $B(w \| r 2) p 2 \wedge$ length $w=$ length $r 2$
$\operatorname{target}(w \| r 1) p 1=f s t(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2))$
$\operatorname{target}(w \| r 2) p 2=\operatorname{snd}(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2))$
proof -
have (path $A(w \| r 1) p 1 \wedge$ path $B(w \| r 2) p 2)$ $\vee(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2)=F A I L$
$\wedge$ length $w>0$
$\wedge$ path $A($ butlast $(w \| r 1)) p 1$
$\wedge$ path B (butlast ( $w \| r 2$ ) $)$ p2
$\wedge \operatorname{succ} A($ last $w)($ target $($ butlast $(w \| r 1)) p 1)=\{ \}$
$\wedge$ succ $B($ last $w)($ target (butlast $(w \| r 2)) p 2) \neq\{ \})$
using productF-path[of wr1 r2 A B FAIL AB p1 p2] assms by blast
moreover have path $A(\operatorname{butlast}(w \| r 1)) p 1$
$\wedge \operatorname{succ} A($ last $w)($ target $($ butlast $(w \| r 1)) p 1)=\{ \}$
$\wedge$ length $w>0 \Longrightarrow$ False
proof -
assume assm : path $A$ (butlast $(w \| r 1)) p 1$
$\wedge$ succ $A($ last $w)($ target $($ butlast $(w \| r 1)) p 1)=\{ \}$
$\wedge$ length $w>0$
obtain $r 1^{\prime}$ where $r 1^{\prime}$-def : path $A\left(w \| r 1^{\prime}\right) p 1 \wedge$ length $r 1^{\prime}=$ length $w$
using assms (9) by auto
then have path $A\left(\right.$ butlast $\left.\left(w \| r 1^{\prime}\right)\right) p 1 \wedge$ length $\left(\right.$ butlast $\left.r 1^{\prime}\right)=$ length (butlast $\left.w\right)$
by (metis FSM.path-append-elim append-butlast-last-id butlast.simps(1) length-butlast)
moreover have path $A($ butlast $(w \| r 1)) p 1 \wedge$ length (butlast r1) $=$ length (butlast $w)$
using assm assms(1) by auto
ultimately have butlast $r 1=$ butlast $r 1^{\prime}$
by (metis assms(1) assms(10) butlast-zip language-state observable-path-unique r1'-def)
then have butlast $(w \| r 1)=$ butlast $\left(w \| r 1^{\prime}\right)$
using assms(1) r1'-def by simp
moreover have succ $A($ last $w)\left(\operatorname{target}\left(\right.\right.$ butlast $\left.\left.\left(w \| r 1^{\prime}\right)\right) p 1\right) \neq\{ \}$
by (metis (no-types) assm empty-iff path-last-succ r1'-def)
ultimately show False
using assm by auto
qed
ultimately have paths : (path $A(w \| r 1) p 1 \wedge$ path $B(w \| r \mathcal{Z}) p$ 2 $)$
by auto
show path $A(w \| r 1) p 1 \wedge$ length $w=$ length $r 1$
using assms(1) paths by simp
show path $B(w \| r 2) p \mathcal{Z} \wedge$ length $w=$ length $r 2$
using assms(1) assms(2) paths by simp
have length $w=0 \Longrightarrow \operatorname{target}(w\|r 1\| r 2)(p 1, p 2)=(p 1, p 2)$
by $\operatorname{simp}$
moreover have length $w>0 \Longrightarrow \operatorname{target}(w\|r 1\| r 2)(p 1, p 2)=\operatorname{last}(r 1 \| r 2)$
proof -
assume length $w>0$
moreover have length $w=$ length ( $r 1 \| r 2$ )
using $\operatorname{assms}(1) \operatorname{assms}(2)$ by $\operatorname{simp}$
ultimately show ?thesis
using target-alt-def(2)[of wr1 || r2 (p1,p2)] by simp
qed
ultimately have $\operatorname{target}(w \| r 1) p 1=f s t(\operatorname{target}(w\|r 1\| r \mathcal{Z})(p 1, p 2))$
$\wedge \operatorname{target}(w \| r 2) p 2=\operatorname{snd}(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2))$
proof (cases length $w$ )
case 0
then show ?thesis by simp
next
case (Suc nat)
then have length $w>0$ by $\operatorname{simp}$
have target $(w\|r 1\| r 2)(p 1, p 2)=$ last $(r 1 \| r 2)$
proof -
have length $w=$ length (r1\| r2)
using $\operatorname{assms}(1) \operatorname{assms}(2)$ by $\operatorname{simp}$
then show ?thesis
using <length $w>0$ 〉target-alt- $\operatorname{def}(2)[$ of $w r 1 \| r 2(p 1, p 2)]$ by $\operatorname{simp}$
qed
moreover have target $(w \| r 1) p 1=$ last $r 1$
using <length $w>0$ 〉target-alt-def(2)[of wr1p1] assms(1) by simp
moreover have target ( $w \|$ r2) p2 $=$ last r2
using 〈length $w>$ 0〉 target-alt-def(2)[of wr2 p2] assms(1) assms(2) by simp
moreover have last $(r 1 \|$ 2 $)=($ last r1, last r2)
using 〈length $w>0$ 〉assms(1) assms(2) zip-last[of r1 r2] by simp
ultimately show ?thesis
by $\operatorname{simp}$

## qed

then show target $(w \| r 1) p 1=f s t(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2))$

$$
\operatorname{target}(w \| r 2) p 2=\operatorname{snd}(\operatorname{target}(w\|r 1\| r 2)(p 1, p 2))
$$

by $\operatorname{simp}+$
qed

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

lemma productF-path-io-targets :
assumes productF A B FAIL AB
and io-targets $A B(q A, q B) w=\{(p A, p B)\}$
and $\quad w \in$ language-state $A q A$
and $\quad w \in$ language-state $B q B$
and observable $A$
and observable $B$
and well-formed $A$
and well-formed $B$
and $\quad q A \in$ nodes $A$
and $q B \in$ nodes $B$
shows $p A \in$ io-targets $A q A w p B \in$ io-targets $B q B w$
proof -
obtain $t r$ where $t r$-def : target $(w \| t r)(q A, q B)=(p A, p B)$
$\wedge$ path $A B(w \| t r)(q A, q B)$
$\wedge$ length $w=$ length $t r$ using assms(2)
by blast
have path- $A$ : path $A(w \|$ map fst tr) $q A \wedge$ length $w=$ length (map fst tr)
using productF-path-reverse-ob-2[of $w$ map fst tr map snd $\operatorname{tr} A$ B FAIL $A B q A q B]$
assms tr-def by auto
have path- $B$ : path $B(w \|$ map snd $t r) q B \wedge$ length $w=$ length (map snd tr)
using productF-path-reverse-ob-2[of $w$ map fst tr map snd $\operatorname{tr} A B$ FAIL $A B q A q B]$
assms $t r$-def by auto
have targets: target ( $w \|$ map fst tr) $q A=p A \wedge \operatorname{target}(w \|$ map snd tr) $q B=p B$
proof (cases tr)
case Nil
then have $q A=p A \wedge q B=p B$
using $t r-d e f$ by auto
then show?thesis
by (simp add: local.Nil)
next
case (Cons a list)
then have last tr $=(p A, p B)$
using tr-def by (simp add: tr-def FSM.target-alt-def states-alt-def)
moreover have target ( $w \|$ map fst tr) $q A=$ 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

```
    moreover have target ( }w||\mathrm{ 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\inio-targets A qA 
    using path- }A\mathrm{ targets by auto
show pB\inio-targets B qBw
    using path-B targets by auto
qed
```

lemma productF-path-io-targets-reverse :
assumes productF $A B$ FAIL $A B$
and $\quad p A \in$ io-targets $A q A w$
and $\quad p B \in$ io-targets $B q B w$
and $\quad w \in$ language-state $A q A$
and $\quad w \in$ language-state $B q B$
and observable $A$
and observable $B$
and well-formed $A$
and well-formed $B$
and $\quad q A \in$ nodes $A$
and $\quad q B \in$ nodes $B$
shows io-targets $A B(q A, q B) w=\{(p A, p B)\}$
proof -
obtain $\operatorname{tr} A$ where path $A(w \| \operatorname{tr} A) q A$
length $w=$ length $\operatorname{tr} A$ $\operatorname{target}(w \| \operatorname{tr} A) q A=p A$
using assms(2) by auto
obtain $\operatorname{tr} B$ where path $B(w \| \operatorname{tr} B) q B$
length $\operatorname{tr} A=$ length $\operatorname{tr} B$
$\operatorname{target}(w \| \operatorname{tr} B) q B=p B$
using <length $w=$ length $\operatorname{tr} A 〉$ assms(3) by auto

```
have path \(A B(w\|\operatorname{tr} A\| \operatorname{tr} B)(q A, q B)\)
    length \((\operatorname{tr} A \| \operatorname{tr} B)=\) length \(w\)
    using productF-path-inclusion
            \([O F 〈 l e n g t h w=l e n g t h \operatorname{tr} A\rangle\langle l e n g t h \operatorname{tr} A=l e n g t h \operatorname{trB} \operatorname{assms}(1) \operatorname{assms}(8,9)-\operatorname{assms}(10,11)]\)
    by \((\operatorname{simp} a d d:\langle l e n g t h \operatorname{tr} A=\) length \(\operatorname{tr} B\rangle\langle l e n g t h ~ w=\) length \(\operatorname{tr} A\rangle\langle p a t h A(w \| \operatorname{tr} A) q A\rangle\)
        \(\langle\) path \(B(w \| \operatorname{tr} B) q B\rangle)+\)
have target \((w\|\operatorname{tr} A\| \operatorname{tr} B)(q A, q B)=(p A, p B)\)
    by \((\operatorname{simp}\) add: <length \(\operatorname{tr} A=\) length \(\operatorname{tr} B\rangle\langle l e n g t h ~ w=\) length \(\operatorname{tr} A\rangle\langle\operatorname{target}(w \| \operatorname{tr} A) q A=p A\rangle\)
        \(\langle\operatorname{target}(w \| \operatorname{tr} B) q B=p B\rangle)\)
have \((p A, p B) \in\) io-targets \(A B(q A, q B) w\)
    by \((\) metis \(\langle l e n g t h ~(t r A \| \operatorname{tr} B)=\) length \(w\rangle\langle\) path \(A B(w\|\operatorname{tr} A\| \operatorname{tr} B)(q A, q B)\rangle\)
        <target \((w\|\operatorname{tr} A\| \operatorname{tr} B)(q A, q B)=(p A, p B)\rangle\) io-target-from-path \()\)
    have observable \(A B\)
    by (metis (no-types) assms(1) assms(6) assms(7) observable-productF)
show ?thesis
    by \((\) meson \(\langle(p A, p B) \in\) io-targets \(A B(q A, q B) w\rangle\langle o b s e r v a b l e ~ A B\rangle\)
        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 $P M(i o \| p)($ initial $P M) \wedge$ length $p=$ length io $\wedge \operatorname{target}(i o \| p)($ initial $P M)=F A I L$
proof -
have io $\neq$ [] using assms by auto
then obtain io-init io-last where $i o-s p l i t[s i m p]:$ io $=i o$-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)
using assms by auto
have io-init @ [io-last $] \in$ 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 tri-def by auto
then have succ-1 : succ M1 io-last (target (io-init || tr1-init) (initial M1)) $\neq\{ \}$ by auto
obtain tr2 where tr2-def : path M2 (io-init || tr2) (initial M2) $\wedge$ 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)) $\neq\{ \}$
then obtain tr2-last where tr2-last $\in$ 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] elanguage-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 $F A I L])=$ 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 \in nodes M2 ^ initial M1 \in 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 \(P M=(\) initial M2, initial M1 \()\)
    using \(\operatorname{assms}(4) \operatorname{productF-simps}(4)\) by blast
    ultimately have
path PM (io-init @ [io-last] || tr2 @ [fst FAIL] \| tr1-init @ [snd FAIL]) (initial PM)
$\wedge$ length (tr2 @ [fst FAIL] \| tr1-init @ [snd FAIL]) = length (io-init @ [io-last])
$\wedge$ 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 $\neg M 1 \preceq M 2$
and well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
shows FAIL $\in$ 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 $\neg M 1 \preceq M 2$
and well-formed M1
and well-formed M2
and observable M2
and productF M2 M1 FAIL PM
obtains $p$
where path PM $p($ initial $P M)$ target $p($ initial $P M)=F A I L$
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 $\quad$ FAIL $\in$ reachable PM (initial PM)
and observable M2
shows $\neg M 1 \preceq M 2$
proof -
obtain pathF where pathF-def : path PM pathF $($ initial $P M) \wedge$ target pathF $($ initial $P M)=F A I L$
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 $P M \neq F A I L$
using assms by auto
then have path $F \neq[]$
using pathF－def by auto
moreover have initial $P M=($ initial M2，initial M1 $)$
using assms by simp
ultimately have path M2（？io｜｜？tr2）（initial M2）$\wedge$ path M1（？io｜｜？tr1）（initial M1）$\vee$ target（？io｜｜？tr2｜｜？tr1）（initial M2，initial M1）＝FAIL $\wedge$
$0<$ length（？io）$\wedge$
path M2（butlast（？io｜｜？tr2））（initial M2）$\wedge$
path M1（butlast（？io｜｜？tr1））（initial M1）$\wedge$
succ M2（last（？io））（target（butlast（？io \｜？tr2））（initial M2））$=\{ \} \wedge$ succ M1（last（？io））（target（butlast（？io \｜？tr1））（initial M1））$\neq\{ \}$
using productF－path－reverse［of ？io ？tr2 ？tr1 M2 M1 FAIL PM initial M2 initial M1］
using assms pathF－def
proof－
have f1：path $P M$（？io｜｜？tr2｜｜？tr1）（initial M2，initial M1）
by（metis（no－types）〈initial $P M=$（initial M2，initial M1）＞pathF－def zip－map－fst－snd）
have f2：length $(? i o)=$ length path $F \longrightarrow$ length $(? i o)=$ length $(? t r 2)$
by auto
have length $(?$ io $)=$ length path $F \wedge$ length $(? t r 2)=$ length $(? t r 1)$
by auto
then show ？thesis
using f2 f1 〈productF M2 M1 FAIL PM〉〈well－formed M1〉〈well－formed M2〉 by blast qed
moreover have $\neg($ path M2（？io \｜？tr2）$($ initial M2）$\wedge$ path M1（？io \｜？tr1）（initial M1）$)$
proof（rule ccontr）
assume $\neg \neg($ path M2（？io \｜？tr2）（initial M2）$) \wedge$ path M1（？io｜｜？tr1）（initial M1））
then have path M2（？io \｜\｜？tr2）（initial M2）
by $\operatorname{simp}$
then have target（？io \｜｜？tr2）（initial M2）$\in$ nodes M2
by auto
then have target（？io \｜？？tr2）（initial M2）$\neq$ 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）$)$ $\langle$ path $P M$ pathF $($ initial $P M) \wedge$ target pathF $($ initial $P M)=F A I L\rangle z i p-m a p-f s t-s n d)$

## then show？thesis

using 〈target（map fst pathF \｜map fst（map snd pathF））（initial M2）$\neq f$ ft FAIL〉 by auto
qed
qed
ultimately have fail－prop ：

```
        target (?io || ?tr2 || ?tr1)(initial M2, initial M1) = FAIL ^
```

$0<$ length（？io）$\wedge$
path M2（butlast（？io \｜？tr2））（initial M2）$\wedge$
path M1（butlast（？io｜｜？tr1））（initial M1）$\wedge$
succ M2（last（？io））（target（butlast（？io \｜？tr2））（initial M2））$=\{ \} \wedge$
succ M1（last（？io））（target（butlast（？io \｜？tr1））（initial M1））$\neq\{ \}$
by auto
then have ？io $\in$ language－state $M 1$（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 }\forallcf.c\not= initial (f::('a, 'b, 'c) FSM) \vee c\in nodes 
    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 & language-state M2 (initial M2)
    proof (rule ccontr)
    assume ᄀ ?io \not\in language-state M2 (initial M2)
    then have assm : ?io \in language-state M2 (initial M2)
        by simp
    then obtain tr2' where tr2'-def : path M2 (?io||tr2') (initial M2)
                        ^length ?io = length tr2'
    by auto
    then obtain tr2'-init tr2'-last where tr2''-split : tr2 '' = tr2''-init @ [tr\mathscr{2}
        using fail-prop by (metis <pathF \not=[]> append-butlast-last-id length-0-conv map-is-Nil-conv)
    have butlast ?io \in language-state M2 (initial M2)
    using fail-prop by auto
    then have {t. path M2 (butlast ?io || t) (initial M2) }\wedge length (butlast ?io) = length t
                ={butlast?tr2}
        using assms(5) observable-path-unique[of butlast ?io M2 initial M2 butlast ?tr2]
                fail-prop by fastforce
    then have }\forallt\mathrm{ ts . path M2 ((butlast ?io) @ [last ?io] | ts @ [t]) (initial M2)
                                    ^length ((butlast ?io) @ [last ?io]) = length (ts @ [t])
                    \longrightarrow t s = \text { butlast ?tr2}
    by (metis (no-types, lifting) FSM.path-append-elim
        <butlast (map fst pathF) \in 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'-init = butlast ?tr2
        using tr2'-def tr2'-split <pathF # []> by auto
    then have path M2 ((butlast ?io) @ [last ?io]||(butlast ?tr2) @ [tr2'-last]) (initial M2)
                ^length ((butlast ?io) @ [last ?io]) = length ((butlast ?tr2) @ [tr2'-last])
        using tr2'-def fail-prop tr2'-split by auto
    then have path M2 ([last ?io] || [tr2'-last])
                                    (target (butlast ?io || butlast ?tr2) (initial M2))
                ^length [last ?io] = length [tr2'-last]
        by auto
    then have tr2'-last \in succ M2 (last (?io)) (target (butlast (?io || ?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 FAIL \in reachable PM (initial PM) \longleftrightarrow\negM1 \preceq M2
proof
    show FAIL \in reachable PM (initial PM) व \M1 \preceq M2
        using assms fail-reachable-reverse by blast
    show \negM1 \preceqM2 \Longrightarrow FAIL \in 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\in\mathrm{ reachable AB q1
    and q2 = FAIL
    and q1 \in nodes AB
shows \exists p . path AB p q1 ^ target pq1 = 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 \not=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 : \neg FAIL & set (butlast (q1 # states p q1))
            then obtain i}\mathrm{ 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 }\forallk.k< length p\longrightarrow target (take k p) q1 \not= 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) \subseteq 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)\subseteq nodes A }\times\mathrm{ nodes B
    using nodes-productF assms by blast
    have finite-nodes:finite (nodes A }\times\mathrm{ nodes B)
```

using assms(2) assms(3) by auto
have length $p \leq$ length (states $p$ q1)
by $\operatorname{simp}$
then have length $p<\operatorname{card}$ (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 $A B$
and well-formed $A$
and well-formed $B$
and $\quad q 2 \in$ reachable $A B q 1$
and $\quad q 1 \in$ nodes $A B$
shows $\exists p$. path $A B p q 1 \wedge$ target $p q 1=q 2 \wedge$ length $p \leq \operatorname{card}($ nodes $A) * \operatorname{card}($ nodes $B)$
proof $($ cases $q 2=$ FAIL $)$
case True
then have q2-def : $q 2=F A I L$
by simp
then show ?thesis
proof (cases q1 $=q 2$ )
case True
then show ?thesis by auto
next
case False
then obtain $p x$ where $p x$-def : path $A B p x q 1 \wedge$ target $p x q 1=q 2$
using assms by auto
then have $p x$-nonempty : $p x \neq[]$
using q2-def False by auto
let $? q x=$ target (butlast $p x$ ) $q 1$
have ? $q x \in$ reachable $A B q 1$
using $p x$-def $p x$-nonempty
by (metis FSM.path-append-elim FSM.reachable.reflexive FSM.reachable-target
append-butlast-last-id)
moreover have ? $q x \neq$ 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 $p x$-def $p x$-nonempty)
ultimately obtain $p x^{\prime}$ where $p x^{\prime}$-def : path $A B p x^{\prime} q 1$
$\wedge$ target $p x^{\prime} q 1=? q x$
$\wedge$ length $p x^{\prime}<\operatorname{card}$ (nodes $A$ ) * card (nodes B)
using assms reaching-path-length[of A B FAIL AB?qx q1] by blast
have $p x$-split : path $A B(($ butlast $p x)$ @ [last px]) q1
$\wedge \operatorname{target}(($ butlast $p x) @[$ last $p x]) q 1=q 2$
using $p x$-def $p x$-nonempty by auto
then have path $A B[$ last $p x] ? q x \wedge$ target [last $p x]$ ? $q x=q 2$
using $p x$-nonempty
proof -
have target [last px] (target (butlast px) q1) $=q 2$
using $p x$-split by force
then show ?thesis
using $p x$-split by blast
qed
then have path $A B\left(p x^{\prime}\right.$ @ $[$ last $\left.p x]\right) q 1 \wedge \operatorname{target}\left(p x^{\prime}\right.$ @ $[$ last $\left.p x]\right) q 1=q 2$
using $p x^{\prime}$-def by auto
moreover have length $\left(p x^{\prime} @[\right.$ last $\left.p x]\right) \leq \operatorname{card}($ nodes $A) * \operatorname{card}($ nodes $B)$
using $p x^{\prime}$-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 }\inLA\capL
shows io }\inLA
proof -
    obtain trA trB where tr-def : path A (io | trA) (initial A) ^ length io = length trA
                        path B (io| |rB) (initial B)^ length io = length tr }
        using assms by blast
    then have path AB(io| trA|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 \inL 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|tr) (q2,q1)
        length xs = length tr
proof -
    have vs @ xs \inLPM
        using productF-language[OF assms(2,4,6,1)] by simp
    then obtain trVX where path PM (vs@xs|trVX)(initial PM)^ length trVX = length(vs@xs)
        by auto
    then have tgt-VX : io-targets PM (initial PM) (vs@xs)={target (vs@xs|trVX)(initial PM)}
        by (metis assms(2) assms(3) assms(5) obs-target-is-io-targets observable-productF)
    have vs \inLPM using <vs@xs \inL PM >
        by (meson language-state-prefix)
    then obtain trV where path PM (vs|trV) (initial PM)^ length trV = length vs
        by auto
    then have tgt-V: io-targets PM (initial PM) vs ={target (vs| |rV) (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|trVX) (initial PM)}
        using observable-io-targets-split[OF <observable PM> tgt-VX tgt-V] by simp
    then have xs \in language-state PM (?q2,?q1)
        by auto
    then obtain tr where path PM (xs|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 $\in$ io-targets $P M($ initial $P M)$ io
proof -
obtain $p$ where path $P M$ (io \|p) (initial PM)
$\wedge$ length $p=$ length io
$\wedge \operatorname{target}($ io $\| p)($ initial $P M)=F A I L$
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 $P M($ initial $P M)$ io $=\{F A I L\}$
proof -
have FAIL $\in$ io-targets $P M$ (initial $P M$ ) io
using sequence-to-failure-reaches-FAIL[OF assms(1-4)] by assumption
have observable $P M$
by (meson assms(2) assms(3) assms(4) observable-productF)
show ?thesis
by (meson $\langle F A I L \in$ io-targets $P M($ initial $P M)$ 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 $P M$ (io \| $p$ ) (initial PM)
length $p=$ length io
target $($ io $\| p)($ initial $P M)=F A I L$
using assms(1) by (metis io-targets-elim singletonI)
have io $\neq$ []
proof
assume io $=[]$
then have io-targets $P M($ initial $P M)$ io $=\{$ initial $P M\}$
by auto
moreover have initial $P M \neq F A I L$
proof -
have initial $P M=($ initial M2, initial M1 $)$
using assms(4) by auto
then have initial $P M \in($ nodes M2 $\times$ nodes M1)
by (simp add: FSM.nodes.initial)
moreover have FAIL $\notin$ (nodes M2 $\times$ 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< length io
    by blast
have target (butlast (io|p)) (initial PM)}\not=F\mathrm{ FAIL
    using no-prefix-targets-FAIL[OF assms(4)<path PM (io | p) (initial PM)>,of (length io) - 1]
    by (metis (no-types, lifting) <0 < length io〉<length p = length io` 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
        <path PM (io|p)(initial PM)> append-butlast-last-id butlast.simps(1))
moreover have nodes PM\subseteq insert FAIL (nodes M2 }\times\mathrm{ 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\mathrm{ nodes M1)
    by blast
have target (butlast (io|p)) (initial PM) ( (nodes M2 × nodes M1)
    using <target (butlast (io|p)) (initial PM)\in insert FAIL (nodes M2 × nodes M1)>
        <target (butlast (io | p)) (initial PM) = FAIL>
    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: <length p = length io`)+
have path PM (butlast (io|p)) (initial PM)
    by (metis FSM.path-append-elim <path PM (io| | ) (initial PM)` append-butlast-last-id
        butlast.simps(1))
then have path PM ((butlast io)|(map fst (butlast p))|(map snd (butlast p)))
                    (initial M2, initial M1)
    using «length p = length io` assms(4) by auto
have target (butlast io | map fst (butlast p)| map snd (butlast p)) (initial M2, initial M1)
        \not= FAIL
    using <length p = length io><target (butlast (io| | ) ) (initial PM) f FAIL> assms(4)
    by auto
have path M2 (butlast io | map fst (butlast p)) (initial M2) ^
            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«length (butlast io) = length (map fst (butlast p))>
            <length (map fst (butlast p)) = length (map snd (butlast p))>
            assms(4) --
            <path PM ((butlast io) || (map fst (butlast p))| (map snd (butlast p)))
            (initial M2, initial M1)> - -]
    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 <target (butlast io | map fst (butlast p)| map snd (butlast p))
            (initial M2, initial M1) }=\mathrm{ FAIL>
    by auto
then have butlast io \inL M2 \cap L M1
    using «length (butlast io) = length (map fst (butlast p))` by auto
have path PM (io | map fst p| map snd p) (initial M2, initial M1)
    using <path PM (io| |)(initial PM)` assms(4) by auto
have length io = length (map fst p)
        length (map fst p) = length (map snd p)
```

by $(\operatorname{simp}$ add：«length $p=$ length $i o\rangle)+$
obtain $p 1^{\prime}$ where path $M 1\left(\right.$ io $\left.\| p 1^{\prime}\right)($ initial $M 1) \wedge$ length io $=$ length $p 1^{\prime}$
using productF－path－reverse－ob
$[O F<l e n g t h$ io $=$ length $($ map fst $p)$ 〉
〈length $($ map fst $p)=$ length $($ map snd $p)\rangle \operatorname{assms}(4)-$
〈path PM（io \｜map fst p\｜map snd p）（initial M2，initial M1）〉］
using assms（2）assms（3）by blast
then have $i o \in L M 1$
by auto
moreover have io $\notin L M 2$
proof
assume $i o \in L M 2$－only possible if io does not target FAIL
then obtain $p 2^{\prime}$ where path M2（io \｜p2＇）（initial M2）length io $=$ length p2＇ by auto
then have length $p 2^{\prime}=$ length $p 1^{\prime}$
using $\left\langle p a t h ~ M 1\left(i o \| p 1^{\prime}\right)(\right.$ initial $M 1) \wedge$ length io $=$ length $\left.p 1^{\prime}\right\rangle$
by auto
have path $P M$（io \｜p2＇\｜$p 1^{\prime}$ ）（initial M2，initial M1）
 of initial M2 initial M1］
$\left\langle\right.$ path M1（io \｜$\left.p 1^{\prime}\right)($ initial $M 1) \wedge$ length io $=$ length $\left.p 1^{\prime}\right\rangle$〈path M2（io｜｜p2＇）（initial M2）〉 assms（2）assms（3）
by blast
have target $\left(\right.$ io $\left.\left\|p 2^{\prime}\right\| p 1^{\prime}\right)($ initial M2，initial M1）$\in($ nodes M2 $\times$ nodes M1）
using＜length io＝length p2＇〉〈path M1（io \｜p1＇）（initial M1）$\wedge$ length io $=$ length $\left.p 1^{\prime}\right\rangle$〈path M2（io \｜｜p2＇）（initial M2）〉
by auto
moreover have $F A I L \notin($ nodes M2 $\times$ nodes M1）
using assms（4）by auto
ultimately have target（io\｜p2＇\｜p1＇）（initial M2，initial M1）$\neq F A I L$
by blast
have length io $=$ length $\left(p 2^{\prime} \| p 1^{\prime}\right)$
by（simp add：＜length io＝length p2＇〉〈length p2＇$=$ length $\left.p 1^{\prime}>\right)$
have target（io \｜p2＇\｜p1＇）（initial M2，initial M1） $\in$ io－targets $P M$（initial M2，initial M1）io
using＜path $P M\left(\right.$ io $\left.\left\|p 2^{\prime}\right\| p 1^{\prime}\right)\left(\right.$ initial M2，initial M1）〉〈length io $=$ length $\left.\left(p \mathcal{Z}^{\prime} \| p 1^{\prime}\right)\right\rangle$
unfolding io－targets．simps by blast
have io－targets $P M$（initial $P M)$ io $\neq\{F A I L\}$
using＜target（io \｜p2＇\｜p1＇）（initial M2，initial M1）
$\in$ io－targets $P M$（initial M2，initial M1）io＞
〈target（io \｜ $\mathrm{p}^{\prime} \| p 1^{\prime}$ ）（initial M2，initial M1）$\neq$ FAIL〉 $\operatorname{assms}(4)$
by auto
then show False
using assms（1）by blast
qed
ultimately have $i o \in L M 1-L$ M2
by blast
show sequence－to－failure M1 M2 io
using 〈butlast io $\in L$ M2 $\cap L M 1\rangle\langle i o \in L M 1-L M 2\rangle$ by auto
qed
end
theory $A T C$
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 \(\mid\) Node 'in 'out \(\Rightarrow\) ('in, 'out) ATC
inductive atc-reaction \(::\) ('in, 'out, 'state) \(F S M \Rightarrow\) 'state \(\Rightarrow\) ('in, 'out) ATC
                        \(\Rightarrow\) ('in \(\times\) 'out) list \(\Rightarrow\) bool
    where
    leaf[intro!]: atc-reaction M q1 Leaf [] |
    node[intro!]: q2 \(\in \operatorname{succ} M(x, y) q 1\)
            \(\Longrightarrow\) atc-reaction M q2 (fy) io
            \(\Longrightarrow\) atc-reaction M q1 (Node \(x f)((x, y) \# i o)\)
inductive-cases leaf-elim[elim!] : atc-reaction M q1 Leaf []
inductive-cases node-elim[elim!] : atc-reaction M q1 (Node xf) (( \(x, y\) )\#io)
```


### 3.1 Properties of ATC-reactions

lemma atc-reaction-empty[simp] :
assumes atc-reaction Mqt[]
shows $t=$ Leaf
using assms atc-reaction.simps by force
lemma atc-reaction-nonempty-no-leaf :
assumes atc-reaction Mqt (Cons a io)
shows $t \neq$ Leaf
using assms
proof -
have $\bigwedge f$ c a ps. $\neg$ atc-reaction $f\left(c::^{\prime} c\right)\left(a::\left({ }^{\prime} a, ' b\right) A T C\right) p s \vee a \neq$ Leaf $\vee a \neq$ Leaf $\vee p s=[]$
using atc-reaction.simps by fastforce
then show?thesis
using assms by blast
qed
lemma atc-reaction-nonempty[elim]:
assumes atc-reaction Mq1 $t$ (Cons $(x, y)$ io)
obtains $q 2 f$
where $t=$ Node $x f q 2 \in \operatorname{succ} M(x, y)$ q1 atc-reaction $M q 2(f y)$ io
proof -
obtain $x 2 f$ where $t=$ Node $x 2 f$
using assms by (metis ATC.exhaust atc-reaction-nonempty-no-leaf)
moreover have $x=x 2$
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 $M(i o \| t r) q 1 \wedge$ length io $=$ length $t r$
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 $q 2$ where $q 2$-def : q2 $\in \operatorname{succ} M(x, y)$ q1 atc-reaction $M$ q2 ( $f y$ ) io-tl
using Cons io-hd-def atc-reaction-nonempty by auto
then obtain $t r-t l$ where $t r-t l-d e f: p a t h ~ M(i o-t l \| t r-t l) q 2$ length io-tl $=$ length $t r-t l$ using Cons.IH [of q2 fy] 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? ?ase using tr-tl-def by fastforce
qed
lemma atc-reaction-path[elim]:
assumes atc-reaction M q1 tio
obtains $t r$
where path $M$ (io \|tr) q1 length io $=$ length $t r$
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\in range f\Longrightarrow subtest t(Node xf)
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\in 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 xf) |fxt.t\in range f}
lemma subtest-rel-altdef: subtest-rel = {(s,t)|st.subtest s t }
    by (auto simp: subtest-rel-def subtest.simps)
```

lemma subtest-relI [intro]: $t \in$ range $f \Longrightarrow(t$, Node $x f) \in$ subtest-rel
by (simp add: subtest-rel-def)
lemma subtest-relI ${ }^{\prime}[$ intro $]: t=f y \Longrightarrow(t$, Node $x f) \in$ 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-acc-iff
by auto
function inputs-atc :: ('a, 'b) ATC $\Rightarrow{ }^{\prime} a$ set where
inputs-atc Leaf $=\{ \} \mid$
inputs-atc $($ Node $x f)=$ insert $x(\cup($ image inputs-atc (range $f)))$
by pat-completeness auto
termination by (relation subtest-rel) auto
fun applicable :: ('in, 'out, 'state) FSM $\Rightarrow$ ('in, 'out) ATC $\Rightarrow$ bool where
applicable $M t=($ inputs-atc $t \subseteq$ inputs $M)$
fun applicable-set :: ('in, 'out, 'state) FSM $\Rightarrow$ ('in, 'out) ATC set $\Rightarrow$ bool where
applicable-set $M \Omega=(\forall t \in \Omega$. 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 => 'state => ('in, 'out) ATC => ('in × 'out) list set where
    IOMqt={tr.atc-reaction Mqttr }
fun IO-set :: ('in, 'out, 'state) FSM => 'state }=>\mathrm{ ('in, 'out) ATC set }=>\mathrm{ ('in }\times\mathrm{ 'out) list set
    where
    IO-set Mq\Omega=\bigcup{IOMqt|t.t\in\Omega}
lemma IO-language :IO M q t\subseteq language-state Mq
    by (metis atc-reaction-path IO.elims language-state mem-Collect-eq subsetI)
```

```
lemma \(I O\)-leaf \([\) simp \(]:\) IO \(M\) q Leaf \(=\{[]\}\)
proof
    show \(I O M\) Leaf \(\subseteq\{[]\}\)
    proof (rule ccontr)
        assume assm: ᄀIO M q Leaf \(\subseteq\{[]\}\)
        then obtain io-hd io-tl where elem-ex : Cons io-hd io-tl \(\in I O M q\) 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 \(\{[]\} \subseteq I O M q\) Leaf by auto
qed
```

lemma IO-applicable-nonempty
assumes applicable $M t$
and completely-specified $M$
and $\quad q 1 \in$ nodes $M$
shows $I O M$ q1 $t \neq\{ \}$
using assms proof (induction t arbitrary: q1)
case Leaf
then show? ?ase by auto
next
case (Node $x$ f)
then have $x \in$ inputs $M$ by auto
then obtain $y$ q2 where $x$-appl : q2 $\in \operatorname{succ} M(x, y) q 1$
using Node unfolding completely-specified.simps by blast
then have applicable $M$ ( $f y$ )
using applicable-subtest Node by metis
moreover have $q 2 \in$ nodes $M$

ultimately have $I O M q 2(f y) \neq\{ \}$
using Node by auto
then show? case unfolding IO.simps
using $x$-appl by blast
qed
lemma $I O$-in-language :
$I O M q t \subseteq L S M q$
unfolding IO.simps by blast
lemma IO-set-in-language :
$I O$-set $M q \Omega \subseteq L S M q$
using $I O$-in-language $[$ of $M$ $q$ ] unfolding $I O$-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) $F S M \Rightarrow$ ('in, 'out) $A T C \Rightarrow$ 'state $\Rightarrow$ 'state $\Rightarrow$ bool where $r$-dist $M$ t s1 s2 $=(t \neq$ Leaf $\wedge I O M s 1 t \cap I O M s 2 t=\{ \})$

```
fun r-dist-set :: ('in, 'out, 'state) FSM => ('in, 'out) ATC set => ' 'state }=>\mathrm{ 'state }=>\mathrm{ bool where
r-dist-set M T s1 s2 = (\existst\inT.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\mathrm{ nodes M
shows q1 f=q2
proof (rule ccontr)
    assume }\neg(q1\not=q2
    then have q1 = q2
        by simp
    then have IO M q1 t={}
        using assms by simp
    moreover have IO M q1 }t\not={
        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\mathrm{ nodes M
shows q1 # 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}\quad\forallt1\inT1.\forall t2 \inT2.r-dist-set M \Omega t1 t2
    and T1\subseteq nodes M
shows T1\capT2 = {}
```

    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) \(F S M \Rightarrow\) ('in * 'out) list \(\Rightarrow\) ('in, 'out) ATC set
    \(\Rightarrow\) ('in * 'out) list set where
    \(B M\) io \(\Omega=\bigcup(\) image \((\lambda s . I O\)-set \(M s \Omega)(\) io-targets \(M(\) initial \(M)\) io) \()\)
fun \(D::\) ('in, 'out, 'state) \(F S M \Rightarrow\) '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 \(\Omega)\left(L S_{\text {in }} M\right.\) (initial \(\left.M\right)\) ISeqs \()\)
fun append-io-B :: ('in, 'out, 'state) \(F S M \Rightarrow\) ('in * 'out) list \(\Rightarrow\) ('in, 'out) ATC set
    \(\Rightarrow\) ('in * 'out) list set where
    append-io- \(B M\) io \(\Omega=\{\) io@res \(\mid\) res.res \(\in B M\) io \(\Omega\}\)
```

lemma $B$-dist ${ }^{\prime}$ :
assumes $d f: B M$ io1 $\Omega \neq B M$ io2 $\Omega$
shows $\quad($ io-targets $M($ initial $M)$ io1) $\neq($ io-targets $M$ (initial $M)$ io2)
using assms by force
lemma $B$-dist :
assumes io-targets $M($ initial $M)$ io1 $=\{q 1\}$
and $\quad$ io-targets $M($ initial $M)$ io2 $=\{q 2\}$
and $\quad B M$ io1 $\Omega \neq B M$ io2 $\Omega$
shows $q 1 \neq q 2$
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) card (D M ISeqs \Omega) \leqcard (nodes M)
proof -
    have D M ISeqs \Omega\subseteq image ( }\lambda\textrm{s}.IO\mathrm{ -set M s }\Omega\mathrm{ ) (nodes M)
    proof
        fix RS assume RS-def : RS \inD M ISeqs \Omega
        then obtain xs ys where RS-tr:RS=BM(xs|ys) \Omega
                                    (xs \inISeqs }\wedge\mathrm{ length xs = length ys
                                    \wedge(xs|ys)\in language-state M (initial M))
        by auto
        then obtain qx where qx-def:io-targets M (initial M)(xs|ys)={qx }
            by (meson io-targets-observable-singleton-ex ob)
        then have RS =IO-set M qx \Omega
            using RS-tr by auto
    moreover have qx nodes M
            by (metis FSM.nodes.initial io-targets-nodes qx-def singletonI)
        ultimately show RS\inimage ( }\lambdas.IO-set Ms\Omega) (nodes M
            by auto
    qed
    moreover have finite (nodes M)
        using assms by auto
    ultimately show finite (D M ISeqs \Omega) card (D M ISeqs \Omega) \leqcard (nodes M)
        by (meson finite-imageI infinite-super surj-card-le)+
qed
```

lemma append-io-B-in-language :
append-io-B $M$ io $\Omega \subseteq L M$
proof
fix $x$ assume $x \in$ append-io- $B M$ io $\Omega$
then obtain res where $x=$ io@ res res $\in B$ io $\Omega$
unfolding append-io-B.simps by blast
then obtain $q$ where $q \in$ io-targets $M$ (initial $M$ ) io res $\in I O$-set $M q \Omega$
unfolding B.simps by blast
then have res $\in L S M q$
using IO-set-in-language $[$ of $M q \Omega$ ] by blast
obtain pIO where path $M$ (io \|pIO) (initial M)
length pIO $=$ length io target $($ io $\| p I O)($ initial $M)=q$
using $\langle q \in$ io-targets $M$ (initial $M$ ) io by auto
moreover obtain $p$ Res where path $M$ (res \| pRes) q length pRes $=$ length res
using 〈res $\in L S M q$ by auto
ultimately have io@res $\in L M$
using FSM.path-append[of M io\| $\|$ pIO initial $M$ res $\| p R e s]$
by (metis language-state length-append zip-append)
then show $x \in L M$
using $« x=i o @ r e s 〉$ by blast
qed
lemma append-io-B-nonempty :
assumes applicable-set $M \Omega$
and completely-specified $M$
and $\quad$ io $\in$ language-state $M($ initial $M)$
and $\Omega \neq\{ \}$
shows append-io-B $M$ io $\Omega \neq\{ \}$

```
proof -
    obtain t where t\in\Omega
        using assms(4) by blast
    then have applicable Mt
        using assms(1) by simp
    moreover obtain tr where path M (io|tr) (initial M)^ 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\not= {}
        using assms(2) IO-applicable-nonempty by simp
    then obtain io' where io' }\inIOM(target (io|tr) (initial M))
        by blast
    then have io' }\inIO\mathrm{ -set M (target (io| |r) (initial M)) S
        using <t \in\Omega\rangle unfolding IO-set.simps by blast
    moreover have (target (io| |r) (initial M)) \in io-targets M (initial M) io
        using <path M (io || tr) (initial M)^ length tr = length io> by auto
    ultimately have io' }\inBM\mathrm{ io }
        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\not={}
    shows io \inLM
proof -
    obtain res where io @ res \in append-io-B M io \Omega^ res \inB M io \Omega
        using assms by auto
    then have io-targets M(initial M) io }\not={
        by auto
    then obtain q}\mathrm{ where q}\in\mathrm{ 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 ~ l e n g t h ~ t r ~ = ~ l e n g t h ~ i o ~ b y ~ a u t o
    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) $F S M \Rightarrow$ ('in, 'out) ATC set $\Rightarrow$ bool where
characterizing-atc-set $M \Omega=$ (applicable-set $M \Omega \wedge(\forall$ s1 $\in($ nodes $M) . \forall$ s $2 \in($ nodes $M)$.
$(\exists t d . r$-dist $M$ td s1 s2) $\longrightarrow(\exists t t \in \Omega . r$-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 ATCreaction to it of the former state is also an ATC-reaction of the latter state.

```
fun atc-reduction :: ('in, 'out, 'state) \(F S M \Rightarrow\) '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 \(t \subseteq I O M 1\) 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 \Omega
    and rdist:r-dist-set M1 \Omega s1 s2
    and t1\in nodes M2
shows r-dist-set M2 \Omega t1 t2
proof -
    obtain td where td-def:td }\in\Omega\wedger\mathrm{ -dist M1 td s1 s2
        using rdist by auto
    then have IO M1 s1 td \capIO M1 s2 td = {}
        using td-def by simp
    moreover have IO M2 t1 td\subseteq \O M1 s1 td
        using red1 td-def by auto
    moreover have IO M2 t2 td \subseteqIO M1 s2 td
        using red2 td-def by auto
    ultimately have no-inter : IO M2 t1 td \cap IO M2 t2 td = {}
        by blast
    then have td\not= Leaf
        by auto
    then have IO M2 t1 td \not={}
        by (meson ap2 IO-applicable-nonempty applicable-set.elims(2) cs2 td-def assms(8))
    then have IO M2 t1 td \not= 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 \(\Rightarrow\) ('in, 'out, 'state2) FSM \(\Rightarrow\) ' in list
                \(\Rightarrow\) ('in, 'out) ATC set \(\Rightarrow\) bool where
    atc-io-reduction-on M1 M2 iseq \(\Omega=\left(L_{i n} M 1\{\right.\) iseq \(\} \subseteq L_{i n}\) M2 \{iseq\}
    \(\wedge\left(\forall\right.\) io \(\in L_{i n} M 1\{\) iseq\} . B M1 io \(\Omega \subseteq B\) M2 io \(\left.\Omega)\right)\)
fun atc-io-reduction-on-sets :: ('in, 'out, 'state1) FSM \(\Rightarrow\) 'in list set \(\Rightarrow\) ('in, 'out) ATC set
                        \(\Rightarrow\) ('in, 'out, 'state2) \(F S M \Rightarrow\) bool where
    atc-io-reduction-on-sets M1 TS \(\Omega\) M2 \(=(\forall\) iseq \(\in T S\). atc-io-reduction-on M1 M2 iseq \(\Omega)\)
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 $\Omega$ M2
and finite $T$
shows io-reduction-on M1 T M2
using $\operatorname{assms}(2,1)$ proof (induction $T$ )
case empty
then show? ?ase by auto
next
case (insert $t$ )
then have atc-io-reduction-on M1 M2 $t \Omega$
by auto
then have $L_{i n} M 1\{t\} \subseteq L_{i n} M 2\{t\}$
using atc-io-reduction-on.simps by blast
have $L_{i n} M 1 T \subseteq L_{i n} M 2 T$
using insert.IH
proof -
have atc-io-reduction-on-sets M1 T $\Omega$ M2
by (meson contra-subsetD insert.prems atc-io-reduction-on-sets.simps subset-insertI)
then show?thesis
using insert.IH by blast
qed
then have $L_{i n} M 1 T \subseteq L_{i n} M 2$ (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 $L_{i n} M 1\{t\} \subseteq L_{i n}$ M2 (insert $\left.t T\right)$
proof -
obtain pps :: (' $a \times$ 'b) list set $\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)$ list set $\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)$ list where $\forall x 0 x 1$. $(\exists v 2 . v 2 \in x 1 \wedge v 2 \notin x 0)=(p p s x 0 x 1 \in x 1 \wedge p p s x 0 x 1 \notin x 0)$
by moura
then have $\forall P$ Pa. pps $P a P \in P \wedge p p s P a P \notin P a \vee P \subseteq P a$ by blast

## moreover

\{ assume map fst $\left(\right.$ pps $\left(L_{i n} M 2(\right.$ insert $\left.\left.t T)\right)\left(L_{i n} M 1\{t\}\right)\right) \notin$ insert $t T$ then have pps $\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1\{t\}\right) \notin L_{i n} M 1\{t\}$
$\vee$ pps $\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1\{t\}\right) \in L_{i n} M 2($ insert $t T)$
by (metis (no-types) insertI1 language-state-for-inputs-map-fst-contained singletonD) \}
ultimately show ?thesis
by (meson «Lin $M 1\{t\} \subseteq L_{i n} M 2\{t\}$ language-state-for-inputs-in-language-state language-state-for-inputs-map-fst set-rev-mp)
qed
ultimately show ?case
proof -
have f1: $\forall p$ s $P$ Pa. (ps::('a $\times$ 'b) list $) \notin P \vee \neg P \subseteq P a \vee p s \in P a$ by blast
obtain pps :: (' $a \times$ 'b) list set $\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)$ list set $\Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)$ list where $\forall x 0 x 1 .(\exists v 2 . v 2 \in x 1 \wedge v 2 \notin x 0)=($ pps $x 0 x 1 \in x 1 \wedge p p s x 0 x 1 \notin x 0)$
by moura
moreover
\{ assume pps $\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1(\right.$ insert $\left.t T)\right)$ $\notin L_{i n} M 1\{t\}$
moreover
\{ assume map fst $\left(p p s\left(L_{i n} M 2(\right.\right.$ insert $\left.t T)\right)\left(L_{i n} M 1(\right.$ insert $\left.\left.t T)\right)\right)$ $\notin\{t\}$
then have map fst (pps ( $L_{\text {in }}$ M2 (insert $\left.t T\right)$ )
$\left(L_{\text {in }} M 1(\right.$ insert $\left.\left.t T)\right)\right) \neq t$
by blast
then have pps $\left(L_{\text {in }} M 2(\right.$ insert $\left.t T)\right)\left(L_{\text {in }} M 1\right.$ (insert $\left.\left.t T\right)\right)$
$\notin L_{i n} M 1($ insert $t T)$
$\vee$ pps $\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1(\right.$ insert $\left.t T)\right)$ $\in L_{i n} M 2($ insert $t T)$
using $f 1$ by (meson 〈Lin M1 $T \subseteq L_{i n}$ M2 (insert $t T$ ) >
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 $t ~ T) ~ M 2 ~$
$\vee \operatorname{pps}\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1(\right.$ insert $\left.t T)\right)$ $\notin L_{i n} M 1($ insert $t T)$
$\vee$ pps $\left(L_{i n} M 2(\right.$ insert $\left.t T)\right)\left(L_{i n} M 1(\right.$ insert $\left.t T)\right)$ $\in L_{i n} M 2($ insert $t T)$
using $f 1$ by (meson language-state-for-inputs-in-language-state language-state-for-inputs-map-fst) \}
ultimately show ?thesis
using $f 1$ by (meson $\left\langle L_{i n} M 1\{t\} \subseteq L_{i n}\right.$ M2 (insert $\left.t T\right)$ ) subsetI)
qed
qed
lemma atc-io-reduction-on-subset :
assumes atc-io-reduction-on-sets M1 T $\Omega$ M2
and $\quad T^{\prime} \subseteq T$
shows atc-io-reduction-on-sets M1 T' $\Omega$ 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 (fy) io-tl
        using Node.prems(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 fio-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 ~ l e n g t h ~ [ ( x , y ) ] = ~ l e n g t h ~ c s \}
        using f1 pt1 by blast
    qed
    ultimately have tgt1: io-targets M1 q1 [(x,y)]={q1x}
        using Node.prems 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.prems(1) ls1 by auto
    then obtain q2x where q2x-def : q2x E 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 f 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
                        ^ length [(x,y)] = length cs}
        using f1 pt2 by blast
    qed
    then have tgt2: io-targets M2 q2 [(x,y)]={q2x}
        using Node.prems io-targets-observable-singleton-ex ls2 q2x-def
        by (metis (no-types, lifting) singletonD)
```

then have language-state M1 $q 1 x \subseteq$ language-state M2 $q 2 x$
using language-state-inclusion-of-state-reached-by-same-sequence
[of M1 q1 M2 q2 $[(x, y)] q 1 x q 2 x]$
tgt1 tgt2 Node.prems by auto
moreover have $q 1 x \in$ nodes M1
using q1x-def(1) Node.prems(2) by (metis insertI1 io-targets-nodes tgt1)
moreover have $q 2 x \in$ nodes M2
using $q 2 x-\operatorname{def}(1)$ Node.prems(3) by (metis insertI1 io-targets-nodes tgt2)
ultimately have $q 2 x \in$ succ M2 $(x, y) q 2 \wedge$ 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 $I O$-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 $t \subseteq I O M 2 q 2 t$
using assms atc-reaction-reduction unfolding IO.simps by auto
lemma IO-set-reduction :
assumes $l s$ : language-state M1 q1 $\subseteq$ language-state M2 q2
and el1: q1 $\in$ nodes M1
and el2 : $q 2 \in$ nodes M2
and ob1: observable M1
and ob2: observable M2
shows $I O$-set M1 q1 $\Omega \subseteq$ IO-set M2 q2 $\Omega$
proof -
have $\forall t \in \Omega$. IO M1 q1 $t \subseteq I O$ M2 q2 $t$ using assms IO-reduction by metis
then show ?thesis
unfolding $I O$-set.simps by blast
qed
lemma $B$-reduction :
assumes red: $M 1 \preceq$ M2
and ob1: observable M1
and ob2 : observable M2
shows $B$ M1 io $\Omega \subseteq B$ M2 io $\Omega$
proof
fix $x y$ assume $x y$-assm : $x y \in B$ M1 io $\Omega$
then obtain $q 1 x$ where $q 1 x$-def : q1x $\in($ io-targets M1 (initial M1) io) $\wedge x y \in I O$-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 $q 1 x$-ob: io-targets M1 (initial M1) io $=\{q 1 x\}$
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) ^ length io = length tr2 by auto
then obtain $q 2 x$ where $q 2 x$-def : $q 2 x \in$ (io-targets M2 (initial M2) io) by auto

```
    then have q2x-ob : io-targets M2 (initial M2) io = {q2x}
    using tr2-def assms
    by (metis io-targets-observable-singleton-ex language-state singleton-iff)
    then have language-state M1 q1x \subseteq language-state M2 q2x
    using q1x-ob assms unfolding io-reduction.simps
    by (simp add: language-state-inclusion-of-state-reached-by-same-sequence)
    then have IO-set M1 q1x \Omega\subseteqIO-set M2 q2x \Omega
    using assms IO-set-reduction by (metis FSM.nodes.initial io-targets-nodes q1x-def q2x-def)
    moreover have B M1 io \Omega=IO-set M1 q1x \Omega
    using q1x-ob by auto
    moreover have B M2 io \Omega=IO-set M2 q2x \Omega
    using q2x-ob by auto
    ultimately have B M1 io \Omega\subseteqBM2 io \Omega
    by simp
    then show xy\inB M2 io \Omega using xy-assm
    by blast
qed
lemma append-io-B-reduction :
    assumes red:M1 \preceq M2
    and ob1 : observable M1
    and ob2 : observable M2
shows append-io-B M1 io \Omega\subseteq append-io-B M2 io \Omega
proof
    fix ioR assume ioR-assm : ioR \in append-io-B M1 io \Omega
    then obtain res where res-def : ioR=io @ res res }\inBM1 io 
        by auto
    then have res }\inB\mathrm{ M2 io }
        using assms B-reduction by (metis (no-types, opaque-lifting) subset-iff)
    then show ioR \in append-io-B M2 io \Omega
        using ioR-assm res-def by auto
qed
```

lemma atc-io-reduction-on-reduction[intro]:
assumes red: M1 $\prec ~ M 2 ~$
and ob1 : observable M1
and ob2 : observable M2
shows atc-io-reduction-on M1 M2 iseq $\Omega$
unfolding atc-io-reduction-on.simps proof
show $L_{i n} M 1\{i s e q\} \subseteq L_{i n} M 2\{i s e q\}$
using red by auto
next
show $\forall i o \in L_{i n} M 1\{$ iseq\}. $B M 1$ io $\Omega \subseteq B$ M2 io $\Omega$
using $B$-reduction assms by blast
qed
lemma atc-io-reduction-on-sets-reduction[intro] :
assumes red : M1 $\preceq ~ M 2$
and ob1 : observable M1
and ob2 : observable M2
shows atc-io-reduction-on-sets M1 TS $\Omega$ M2
using assms atc-io-reduction-on-reduction by (metis atc-io-reduction-on-sets.elims(3))
lemma atc-io-reduction-on-sets-via-LS ${ }_{i n}$ :
assumes atc-io-reduction-on-sets M1 TS $\Omega$ M2
shows $\left(L_{i n} M 1 T S \cup\left(\bigcup i o \in L_{i n} M 1 T S . B M 1\right.\right.$ io $\left.\left.\Omega\right)\right)$
$\subseteq\left(L_{i n} M 2 T S \cup\left(\bigcup i o \in L_{i n} M 2\right.\right.$ TS. B M2 io $\left.\left.\Omega\right)\right)$
proof -
have $\forall$ iseq $\in T S .\left(L_{i n} M 1\{i s e q\} \subseteq L_{i n} M 2\{i s e q\}\right.$
$\wedge\left(\forall\right.$ io $\in L_{\text {in }} M 1\{i s e q\}$. B M1 io $\Omega \subseteq B$ M2 io $\left.\left.\Omega\right)\right)$

```
    using assms by auto
then have }\forall\mathrm{ iseq }\inTS.(\bigcup\mathrm{ iofLin M1 {iseq}. B M1 io S)
                    \subseteq ( \bigcup \text { iofLin M2 \{iseq\}. B M2 io ת)}
    by blast
moreover have }\forall\mathrm{ iseq }\inTS.(\bigcup\mathrm{ iofLin M2 {iseq}. B M2 io ת)
                        \subseteq(\bigcupio\inLin M2 TS. B M2 io \Omega)
    unfolding language-state-for-inputs.simps by blast
ultimately have elem-subset:}\forall\mathrm{ iseq }\inTS\mathrm{ .
                                    (\bigcupio\inLin M1 {iseq}. B M1 io \Omega)
                                    \subseteq ( \bigcup i o \in L _ { i n } ~ M 2 ~ T S . ~ B ~ M 2 ~ i o ~ \Omega )
    by blast
    show ?thesis
    proof
    fix x assume x f Lin M1 TS \cup(\bigcupio\inLin M1 TS. B M1 io \Omega)
    then show }x\in\mp@subsup{L}{in}{}M2TS\cup(\bigcupio\in\mp@subsup{L}{in}{}M2 TS. B M2 io \Omega
    proof (cases x }\in\mp@subsup{L}{in}{}M1TS
        case True
        then obtain iseq where iseq \inTS x\in Lin M1 {iseq}
            unfolding language-state-for-inputs.simps by blast
        then have atc-io-reduction-on M1 M2 iseq \Omega
            using assms by auto
        then have Lin M1 {iseq}}\subseteq\mp@subsup{L}{in}{}M2{iseq
            by auto
        then have x\in Lin M2 TS
            by (metis (no-types, lifting) UN-I
                \thesis. (\bigwedgeiseq. \llbracketiseq G TS; x \in Lin M1 {iseq}\rrbracket\Longrightarrow thesis)\Longrightarrow thesis`
                \foralliseq\inTS.L Lin M1 {iseq}\subseteq L Lin M2 {iseq} ^(\forallio\inLin M1 {iseq}. B M1 io \Omega\subseteqB M2 io \Omega)>
                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(\bigcupio\in\mp@subsup{L}{in}{}M1 TS. B M1 io \Omega
            using <x\in Lin M1 TS \cup (\bigcup io\inLin M1 TS. B M1 io \Omega)> by blast
        then obtain io where io }\in\mp@subsup{L}{in}{}M1TSx\inB M1 io 
            by blast
        then obtain iseq where iseq \inTS io\inLin M1 {iseq}
            unfolding language-state-for-inputs.simps by blast
        have }x\in(\bigcup\mathrm{ iofLin M1 {iseq}. B M1 io ת)
            using <io \in Lin M1 {iseq}〉 <x\inB M1 io \Omega\rangle by blast
        then have }x\in(\bigcupio\in\mp@subsup{L}{in}{}M2 TS. B M2 io \Omega
            using <iseq }\in\mathrm{ TS` 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) \(F S M \Rightarrow\) ('in \(\times\) 'out) list set set where
    Perm \(V M=\{\) image \(f V \mid f . \forall v \in V . f v \in\) language-state-for-input \(M\) (initial \(M\) ) \(v\}\)
lemma perm-empty :
    assumes is-det-state-cover M2 \(V\)
    and \(V^{\prime \prime} \in \operatorname{Perm} V M 1\)
shows []\(\in V^{\prime \prime}\)
proof -
    have init-seq : [] \(\in V\) using det-state-cover-empty assms by simp
    obtain \(f\) where \(f\)-def : \(V^{\prime \prime}=\) image \(f V\)
                                    \(\wedge(\forall v \in V . f v \in\) language-state-for-input M1 (initial M1) v)
        using assms by auto
    then have \(f[]=[]\)
        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 \(\quad V^{\prime \prime} \in \operatorname{Perm} V M 1\)
    shows finite \(V^{\prime \prime}\)
proof -
    obtain \(f\) where is-det-state-cover-ass M2 \(f \wedge V=f\) ' d-reachable M2 (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 \(\operatorname{simp}\)
        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 \(f^{\prime \prime}\) where \(V^{\prime \prime}=\) image \(f^{\prime \prime} V\)
                                    \(\wedge\left(\forall v \in V \cdot f^{\prime \prime} v \in\right.\) language-state-for-input M1 (initial M1) \(\left.v\right)\)
        using assms(3) by auto
    ultimately show ?thesis
        by \(\operatorname{simp}\)
qed
lemma perm-inputs :
    assumes \(V^{\prime \prime} \in\) Perm \(V\) M
    and \(\quad v s \in V^{\prime \prime}\)
shows map fst vs \(\in V\)
proof -
    obtain \(f\) where \(f\)-def : \(V^{\prime \prime}=\) image \(f V\)
                                    \(\wedge(\forall v \in V . f v \in\) language-state-for-input \(M(\) initial \(M) v)\)
    using assms by auto
    then obtain \(v\) where \(v\)-def \(: v \in V \wedge f v=v s\)
        using assms by auto
    then have \(v s \in\) language-state-for-input \(M(\) 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\mathrm{ Perm V M
    and vs1\inV'
    and vs2\in V'
    and vs1 f=vs2
shows map fst vs1 }\not=\mathrm{ map fst vs2
proof -
    obtain f}\mathrm{ where f-def : }\mp@subsup{V}{}{\prime\prime}=\mathrm{ image f V
                                    \wedge ( \forall v \in V . f v \in l a n g u a g e - s t a t e - f o r - i n p u t ~ M ( i n i t i a l ~ M ) ~ v )
        using assms by auto
    then obtain v1 v2 where v-def:v1\inV\wedgefv1=vs1^v2\inV \fv2=vs2
        using assms by auto
    then have vs1 \in language-state-for-input M (initial M) v1
                vs2 \in language-state-for-input M (initial M) v2
        using f
    moreover have v1 f v2
        using v-def assms(4) by blast
    ultimately show ?thesis
        by auto
qed
lemma perm-language :
    assumes }\mp@subsup{V}{}{\prime\prime}\in\operatorname{Perm V M
    and vs\inV}\mp@subsup{V}{}{\prime\prime
shows vs \inLM
proof -
    obtain f}\mathrm{ where f-def : image f V = V'
                        \wedge(\forallv\inV.fv\inlanguage-state-for-input M(initial M)v)
        using assms(1) by auto
    then have }\existsv.fv=vs\wedgefv\inlanguage-state-for-input M(initial M)
        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 $m \equiv$ (inputs M2 $=$ inputs $M 1 \wedge$ card $($ nodes M1) $\leq m)$
lemma asc-fault-domain-props[elim!] :
assumes asc-fault-domain M2 M1 m
shows inputs M2 $=$ inputs M1 card (nodes M1) $\leq m$ using assms by auto

## abbreviation

test-tools M2 M1 FAIL PM V $\Omega \equiv$ ( productF M2 M1 FAIL PM
$\wedge$ is-det-state-cover M2 V
$\wedge$ applicable-set M2 $\Omega$
)
lemma test-tools-props[elim] :
assumes test-tools M2 M1 FAIL PMV $\Omega$
and asc-fault-domain M2 M1 m
shows productF M2 M1 FAIL PM
is-det-state-cover M2 V
applicable-set M2 $\Omega$
applicable-set M1 $\Omega$
proof -
show productF M2 M1 FAIL PM using assms(1) by blast
show is-det-state-cover M2 V using assms(1) by blast
show applicable-set M2 $\Omega$ using assms(1) by blast
then show 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 \not={}
proof -
    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 assms(1)]
        by (metis assms(1) finite-imageI infinite-super is-det-state-cover.elims(2))
    have [] \inV
        using assms(1) det-state-cover-empty by blast
    have }\bigwedgeVS.VS\subseteqV\wedgeVS\not={}\LongrightarrowPerm VS M1 \not={
    proof -
    fix VS assume VS\subseteqV^VS\not={}
    then have finite VS using <finite V>
        using infinite-subset by auto
    then show Perm VS M1 \not={}
        using〈VS\subseteqV^VS\not={}><finite VS>
    proof (induction VS)
            case empty
            then show ?case by auto
    next
            case (insert vs F)
            then have vs \inV by blast
            obtain q2 where d-reaches M2 (initial M2) vs q2
                using det-state-cover-d-reachable[OF assms(1)\langlevs\inV`] by blast
            then obtain vs' vsP where io-path: length vs = length vs'
                \ength vs = length vsP
                    ^(path M2 ((vs |vs')|vsP) (initial M2))
                            ^ target ((vs |vs')|vsP)(initial M2) = q2
```

            by auto
        have well-formed M2
                using assms by auto
            have map fst (map fst \(\left.\left(\left(v s \| v s^{\prime}\right) \| v s P\right)\right)=v s\)
            proof -
            have length ( \(v s \| v s^{\prime}\) ) \(=\) length \(v s P\)
                using io-path by simp
            then show ?thesis
                    using io-path by auto
            qed
            moreover have set (map fst (map fst \(\left.\left.\left(\left(v s \| v s^{\prime}\right) \| v s P\right)\right)\right) \subseteq\) inputs M2
                using path-input-containment[OF «well-formed M2〉, of (vs \| vs') || vsP initial M2]
                    io-path
            by linarith
            ultimately have set vs \(\subseteq\) inputs M2
            by presburger
            then have set \(v s \subseteq\) inputs M1
    ```
            using assms by auto
        then have Lin M1 {vs}\not={}
            using assms(2) language-state-for-inputs-nonempty
            by (metis FSM.nodes.initial)
        then have language-state-for-input M1 (initial M1) vs \not={}
            by auto
        then obtain vs' where vs}\mp@subsup{}{}{\prime}\inl=language-state-for-input M1 (initial M1) v
            by blast
        show ?case
        proof (cases F={})
            case True
            moreover obtain f}\mathrm{ where fvs=vs'
                by moura
            ultimately have image f(insert vs F)\inPerm (insert vs F)M1
            using Perm.simps }\langlev\mp@subsup{s}{}{\prime}\inlanguage-state-for-input M1 (initial M1) vs` by blas
            then show ?thesis by blast
        next
            case False
            then obtain F}\mp@subsup{F}{}{\prime\prime}\mathrm{ where }\mp@subsup{F}{}{\prime\prime}\in\operatorname{Perm F M1
                using insert.IH insert.hyps(1) insert.prems(1) by blast
            then obtain f}\mathrm{ where }\mp@subsup{F}{}{\prime\prime}=\mathrm{ image f F
                                    (\forallv\inF.fv\inlanguage-state-for-input M1 (initial M1) v)
            by auto
            let ?f =f(vs:=vs')
            have }\forallv\in(\mathrm{ insert vs F). ?f v}\inlanguage-state-for-input M1 (initial M1) v
            proof
            fix v}\mathrm{ assume v}\in\mathrm{ insert vs F
            then show ?f v\inlanguage-state-for-input M1 (initial M1) v
            proof (cases v=vs)
                    case True
                    then show ?thesis
                    using <vs' \in language-state-for-input M1 (initial M1) vs> by auto
                    next
                    case False
                then have v}\in
                    using }\langlev\in\mathrm{ insert vs F> by blast
            then show ?thesis
                using False }\langle\forallv\inF.fv\inlanguage-state-for-input M1 (initial M1) v\rangle by aut
            qed
            qed
            then have image ?f (insert vs F) \in Perm (insert vs F) M1
                using Perm.simps by blast
            then show ?thesis
                by blast
            qed
        qed
    qed
    then show ?thesis
    using <[] G V> by blast
qed
lemma perm-elem :
    assumes is-det-state-cover M2 V
    and OFSM M1
    and OFSM M2
    and inputs M1 = inputs M2
    and vs \inV
    and vs'}\inlanguage-state-for-input M1 (initial M1) v
obtains }\mp@subsup{V}{}{\prime\prime
where }\mp@subsup{V}{}{\prime\prime}\inP\operatorname{Perm}VM1vs'\in\mp@subsup{V}{}{\prime\prime
proof -
```

```
    obtain }\mp@subsup{V}{}{\prime\prime}\mathrm{ where V V' }\in\mathrm{ Perm V M1
    using perm-nonempty[OF assms(1-4)] by blast
    then obtain f}\mathrm{ where }\mp@subsup{V}{}{\prime\prime}=\mathrm{ image f V
                (\forallv\inV.fv\inlanguage-state-for-input M1 (initial M1) v)
    by auto
    let ?f =f(vs := vs')
    have }\forallv\inV.(?f v)\in(language-state-for-input M1 (initial M1) v
    using }\forallv\inV.(fv)\in(language-state-for-input M1 (initial M1) v)\rangle assms(6) by fastforc
    then have (image ?f V) \in Perm V M1
    unfolding Perm.simps by blast
    moreover have vs'}\in\mathrm{ image ?f }
    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) FSM \(\Rightarrow\) 'state \(\Rightarrow\) ('in \(\times\) 'out) list
            \(\Rightarrow\) ('in \(\times\) 'out) list \(\Rightarrow\) ('in \(\times\) 'out) list set
    where
    \(R\) M s vs \(x s=\left\{v s @ x s^{\prime} \mid x s^{\prime} . x s^{\prime} \neq[]\right.\)
                    \(\wedge\) prefix \(x s^{\prime}\) xs
                            \(\wedge s \in\) io-targets \(M(\) initial \(\left.M)\left(v s @ x s^{\prime}\right)\right\}\)
lemma finite- \(R\) : finite ( \(R\) Ms vs xs)
proof -
    have \(R M\) s vs xs \(\subseteq\left\{v s @ x s^{\prime} \mid x s^{\prime} . p r e f i x x s^{\prime} x s\right\}\)
        by auto
    then have \(R M\) s vs \(x s \subseteq\) image \(\left(\lambda x s^{\prime} . v s @ x s^{\prime}\right)\left\{x s^{\prime}\right.\).prefix \(\left.x s^{\prime} x s\right\}\)
        by auto
    moreover have \(\left\{x s^{\prime}\right.\). prefix \(\left.x s^{\prime} x s\right\}=\{\) take \(n x s \mid n \cdot n \leq\) length \(x s\}\)
    proof
        show \(\left\{x s^{\prime}\right.\). prefix \(\left.x s^{\prime} x s\right\} \subseteq\{\) take \(n x s \mid n . n \leq\) length \(x s\}\)
        proof
            fix \(x s^{\prime}\) assume \(x s^{\prime} \in\left\{x s^{\prime}\right.\). prefix \(\left.x s^{\prime} x s\right\}\)
            then obtain \(z s^{\prime}\) where \(x s^{\prime} @ z s^{\prime}=x s\)
                by (metis (full-types) mem-Collect-eq prefixE)
            then obtain \(i\) where \(x s^{\prime}=\) take \(i x s \wedge i \leq\) length \(x s\)
                by (metis (full-types) append-eq-conv-conj le-cases take-all)
            then show \(x s^{\prime} \in\{\) take \(n x s \mid n . n \leq\) length \(x s\}\)
                by auto
    qed
    show \(\{\) take \(n\) xs \(\mid n . n \leq\) length \(x s\} \subseteq\left\{x s^{\prime}\right.\). prefix \(\left.x s^{\prime} x s\right\}\)
        using take-is-prefix by force
    qed
    moreover have finite \(\{\) take \(n x s \mid n . n \leq\) length \(x s\}\)
        by auto
    ultimately show?thesis
        by auto
qed
```

lemma card-union-of-singletons :
assumes $\forall S \in S S .(\exists t . S=\{t\})$
shows card $(\bigcup S S)=\operatorname{card} S S$
proof -
let $? f=\lambda x .\{x\}$
have bij-betw ?f $(\bigcup S S) S S$
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 $\forall S 1 \in S S . \forall S 2 \in S S . S 1=S 2 \vee f S 1 \cap f S 2=\{ \}$
and finite $S S$
and $\quad \forall S \in S S . f S \neq\{ \}$
shows card (image f SS) $=$ card $S S$
proof -
from $\operatorname{assms(2)}$ have $\forall S 1 \in S S . \forall S 2 \in S S . S 1=S 2 \vee f S 1 \cap f S 2=\{ \}$

$$
\Longrightarrow \forall S \in S S . f S \neq\{ \} \Longrightarrow \text { ?thesis }
$$

proof (induction $S S$ )
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 $f x \notin$ image $f F$
by auto
then have card $($ image $f($ insert $x F))=\operatorname{Suc}(\operatorname{card}(\operatorname{image} f F))$
using insert by auto
moreover have card ( $\left.f^{\prime} F\right)=\operatorname{card} F$
using insert by auto
moreover have card (insert $x F)=$ Suc (card F)
using insert by auto
ultimately show ?case
by $\operatorname{simp}$
qed
then show? ?thesis
using assms by simp
qed
lemma $R$-count :
assumes (vs@xs) $\in L M 1 \cap L M 2$
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 $P M($ initial $P M)$ vs $=\{(q 2, q 1)\}$
and path $P M(x s \| t r)(q 2, q 1)$
and length $x s=$ length $t r$
and distinct (states ( $x s \| t r$ ) (q2,q1))
shows card $(\bigcup$ (image (io-targets M1 (initial M1)) (R M2 s vs xs) ) ) $=\operatorname{card}(R$ 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\mathrm{ io }\in(R M2 s vs xs). io-targets M2 (initial M2) io ={s
    proof
    fix io assume io }\inR\mathrm{ M2 s vs xs
    then have s\inio-targets M2 (initial M2) io
        by auto
    moreover have io \inlanguage-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
```

```
moreover have }\forall io\inR M2 s vs xs . io-targets PM (initial PM) i
                = io-targets M2 (initial M2) io }\times\mathrm{ io-targets M1 (initial M1) io
proof
    fix io assume io-assm:io \inR M2 s vs xs
    then have io-prefix:prefix io (vs @ xs)
        by auto
    then have io-lang-subs:io }\inLM1\wedge io\inL M2
        using assms(1) unfolding prefix-def by (metis IntE language-state language-state-split)
    then have io-lang-inter:io }\inLM1\capL M
        by simp
    then have io-lang-pm:io\inLPM
        using productF-language assms by blast
    moreover obtain p2 p1 where (p2,p1) \in io-targets PM (initial PM) io
        by (metis assms(2) assms(3) assms(7) 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| |rP) (initial PM) = ( p2,p1)
                    \wedge path PM (io || trP) (initial PM)
                            \wedge ~ l e n g t h ~ i o ~ = ~ l e n g t h ~ t r ~ P ~
        proof -
            assume a1: \trP. target (io |trP) (initial PM)=(p2, p1)
                                    \wedge path PM (io || trP) (initial PM)
                                    \wedge ~ l e n g t h ~ i o ~ = ~ l e n g t h ~ t r P \Longrightarrow ~ t h e s i s
            have \existsps.target (io| |s)(initial PM) = (p2,p1)
                    ^ path PM (io|ps) (initial PM)^ length io = length ps
            using <(p2, p1) \in io-targets PM (initial PM) io> 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)^ length io = length ps} ={pps}
                \vee \neg \text { observable PM}
            by (metis (no-types)<\thesis.\llbracketobservable PM; io \inL PM; \tr.
                                    {t. path PM (io|t) (initial PM)
                                    ^length io = length t } = {tr} \Longrightarrow thesis\rrbracket\Longrightarrow thesis>
                io-lang-pm)
            have f2: observable PM
            by (meson «observable M1` <observable M2` <productF M2 M1 FAIL PM` 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-subs 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-subs assms(2) by blast
    have path PM (io || trIO2 || trIO1) (initial M2, initial M1)
        \wedge ~ l e n g t h ~ i o ~ = ~ l e n g t h ~ t r I O 2 ~
        ^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 $f 1$ by presburger
then show ?thesis
using f2 f1 assms(4) assms(5) assms(7) by blast
qed
then have $\operatorname{trP}$-split : path PM (io \| trIO2 || trIO1) (initial PM)
$\wedge$ length io $=$ length trIO2
$\wedge$ length trIO2 $=$ length trIO1
using assms(7) by auto
then have $\operatorname{tr} P-z i p: \operatorname{trIO} 2 \| \operatorname{trIO1}=\operatorname{tr} P$
using $\operatorname{trP}$-def trP-unique using length-zip by fastforce
have target (io \|trIO2) (initial M2) $=p 2$
$\wedge$ path M2 (io || trIO2) (initial M2)
$\wedge$ length io $=$ length trIO2
using $\operatorname{tr} P$-zip $\operatorname{tr} P$-split assms(7) $\operatorname{tr} P$-def $\operatorname{trIO} 2$-def by auto
then have $p 2 \in$ io-targets M2 (initial M2) io
by auto
then have targets-2 : io-targets M2 (initial M2) io $=\{p 2\}$
by (metis state-component-2 io-assm singletonD)
have target (io $\|$ trIO1) $($ initial M1) $=p 1$ $\wedge$ path M1 (io || trIO1) (initial M1) $\wedge$ length io $=$ length trIO1
using trP-zip trP-split assms(7) trP-def trIO1-def by auto
then have $p 1 \in$ io-targets M1 (initial M1) io
by auto
then have targets-1: io-targets M1 (initial M1) io $=\{p 1\}$
by (metis io-lang-subs assms(2) io-targets-observable-singleton-ex singletonD)
have io-targets M2 (initial M2) io $\times$ io-targets M1 (initial M1) io $=\{(p 2, p 1)\}$
using targets- 2 targets- 1 by simp
then show io-targets PM (initial PM) io
$=$ io-targets M2 (initial M2) io $\times$ io-targets M1 (initial M1) io
using targets-pm by simp
qed
ultimately have state-components $: \forall$ io $\in R$ M2 s vs xs. io-targets $P M$ (initial PM) io $=\{s\} \times$ io-targets M1 (initial M1) io
by auto
then have $\bigcup($ image (io-targets $P M($ initial $P M))(R$ M2 $s$ vs $x s))$
$=\bigcup($ image $(\lambda$ io $\cdot\{s\} \times$ io-targets M1 (initial M1) io) $(R$ M2 s vs xs $))$
by auto
then have $\bigcup($ image (io-targets PM (initial PM)) (R M2 s vs xs)) $=\{s\} \times \bigcup($ image $($ io-targets M1 (initial M1) $)(R$ M2 s vs xs $))$
by auto
then have card $(\bigcup$ (image (io-targets PM (initial PM)) (R M2 s vs xs $)$ ))
$=\operatorname{card}(\bigcup$ (image (io-targets M1 (initial M1)) $(R$ M2 s vs xs)))
by (metis (no-types) card-cartesian-product-singleton)
moreover have card $(\bigcup$ (image (io-targets PM (initial PM)) (R M2 s vs xs))) $=\operatorname{card}($ R M2 s vs xs)
proof (rule ccontr)
assume assm : card $(\bigcup$ (io-targets $P M($ initial PM)' $R$ M2 s vs xs $)) \neq \operatorname{card}(R$ M2 s vs xs)
have $\forall$ io $\in R$ M2 s vs xs. io $\in L P M$
proof
fix $i o$ assume $i o$-assm : io $\in R$ M2 $s$ vs $x s$
then have prefix io (vs @ xs)
by auto
then have io $\in L M 1 \wedge i o \in L M 2$
using assms(1) unfolding prefix-def by (metis IntE language-state language-state-split)
then show io $\in L P M$
using productF-language assms by blast
qed
then have singletons: $\forall i o \in R$ M2 s vs xs. $(\exists t$. io-targets $P M$ (initial $P M)$ io $=\{t\})$
using io-targets-observable-singleton-ex observable-productF assms by metis
then have card-targets : card $(\bigcup$ (io-targets PM (initial PM)'R M2 s vs xs))

$$
=\operatorname{card}(\text { image }(\text { io-targets } P M(\text { initial PM }))(R \text { M2 } s \text { vs xs }))
$$

using finite- $R$ card-union-of-singletons
[of image (io-targets PM (initial PM)) (R M2 s vs xs)]
by $\operatorname{simp}$
moreover have card (image (io-targets PM (initial PM)) (R M2 s vs xs $)$ ) $\operatorname{card}(R$ M2 s vs xs $)$ using finite- $R$ by (metis card-image-le)
ultimately have card-le : card ( $\bigcup$ (io-targets $P M$ (initial PM)' $R$ M2 s vs xs)) $<\operatorname{card}(R$ M2 $s$ vs $x s)$
using assm by linarith
have $\exists i o 1 \in(R$ M2 s vs xs). $\exists i o 2 \in(R$ M2 s vs xs). io1 $\neq i o 2$ $\wedge$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M($ initial $P M)$ io2 $\neq\{ \}$
proof (rule ccontr)
assume $\neg(\exists i o 1 \in R$ M2 s vs $x s . \exists i o 2 \in R$ M2 s vs xs. io1 $\neq i o 2$

$$
\wedge \text { io-targets } P M(\text { initial } P M) \text { io1 } \cap \text { io-targets } P M(\text { initial } P M) \text { io } 2 \neq\{ \})
$$

then have $\forall i o 1 \in R$ M2 s vs $x s . \forall i o 2 \in R$ M2 s vs xs. io1 $=i o 2$
$\vee$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M($ initial $P M)$ io2 $=\{ \}$
by blast
moreover have $\forall i o \in R$ M2 s vs xs. io-targets $P M($ initial $P M)$ io $\neq\{ \}$
by (metis insert-not-empty singletons)
ultimately have card (image (io-targets PM (initial PM)) (R M2 s vs xs))

$$
=\operatorname{card}(R M 2 s v s x s)
$$

using finite- $R$ [of M2 s vs xs] card-union-of-distinct
[of R M2 s vs xs (io-targets PM (initial PM))]
by blast
then show False
using card-le card-targets by linarith
qed
then have $\exists$ io1 io2. io1 $\in(R$ M2 s vs $x s)$
$\wedge i o 2 \in(R$ M2 s vs xs)
$\wedge$ io1 $\neq$ io2
$\wedge$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M($ initial $P M)$ io2 $\neq\{ \}$
by blast
moreover have $\forall$ io1 io2. $($ io1 $\in(R$ M2 s vs $x s) \wedge i o 2 \in(R$ M2 s vs $x s) \wedge i o 1 \neq i o 2)$
$\longrightarrow$ length io1 $\neq$ length io2
proof (rule ccontr)
assume $\neg(\forall$ io1 io2. io1 $\in R$ M2 s vs $x s \wedge i o 2 \in R$ M2 s vs $x s \wedge i o 1 \neq i o 2$ $\longrightarrow$ length io1 $\neq$ length io2)
then obtain io1 io2 where io-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) ^ prefixio2 (vs @ xs)
by auto
then have $i o 1=$ take $($ length io1 $)(v s @ x s) \wedge$ io2 $=$ take (length io2) $(v s @ x s)$
by (metis append-eq-conv-conj prefixE)
then show False
using io-def by auto
qed
ultimately obtain io1 io2 where rep-ios-def :
io1 $\in(R$ M2 s vs $x s)$
$\wedge i o 2 \in(R$ M2 s vs $x s)$
$\wedge$ length io1 < length io2
$\wedge$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M($ initial $P M)$ io2 $\neq\{ \}$
by (metis inf-sup-aci(1) linorder-neqE-nat)
obtain rep where $(s, r e p) \in$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M$ (initial PM) io2 proof -
assume $a 1: \bigwedge$ rep. $(s$, rep $) \in$ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M$ (initial $P M)$ io2 $\Longrightarrow$ thesis
have $\exists f$. Sigma $\{s\} f \cap($ io-targets $P M($ initial $P M)$ io1 $\cap$ io-targets $P M$ (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-state : io-targets $P M($ initial $P M)$ io1 $=\{(s, r e p)\}$

$$
\wedge \text { io-targets } P M(\text { initial } P M) \text { io2 }=\{(s, r e p)\}
$$

by (metis Int-iff rep-ios-def singletonD singletons)
obtain io1X io2X where rep-ios-split : io1 = vs@ io1X
$\wedge$ prefix io1X xs
$\wedge i o 2=v s @ i o 2 X$
$\wedge$ 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 $v s @ x s \in L P M$
by (metis (no-types) assms(1) assms(4) $\operatorname{assms(5)} \operatorname{assms}(7)$ inf-commute productF-language)
then have $v s \in L P M$
by (meson language-state-prefix)
then obtain $\operatorname{tr} V$ where $\operatorname{tr} V$-def : \{tr. path $P M(v s \| t r)($ initial $P M) \wedge$ length $v s=$ length tr $\}$

$$
=\{\operatorname{tr} V\}
$$

using observable-path-unique-ex[of PM vs initial PM]
$\operatorname{assms}(2) \operatorname{assms}(3) \operatorname{assms}(7)$ observable-productF
by blast
let ? $q v=\operatorname{target}(v s \| \operatorname{tr} V)($ initial $P M)$
have ? $q v \in$ io-targets $P M$ (initial $P M$ ) vs using $\operatorname{tr} V$-def by auto
then have $q v$-simp $[\operatorname{simp}]: ? q v=(q 2, q 1)$
using singletons assms by blast
then have ? $q v \in$ nodes $P M$
using trV-def assms by blast
— get a path using io1X from the state reached by vs in PM
obtain tr1X-all where tr1X-all-def : path PM (vs @ io1X || tr1X-all) (initial PM)
$\wedge$ length $(v s @ i o 1 X)=$ length $\operatorname{tr} 1 X$-all
using rep-ios-def rep-ios-split by auto
let ? $\operatorname{tr} 1 X=$ drop (length vs) tr1X-all
have take (length vs) tr1X-all $=\operatorname{tr} V$
proof -
have path $P M$ (vs \| take (length vs) tr1X-all) (initial PM)
$\wedge$ length $v s=$ length (take (length vs) tr1X-all)
using tr1X-all-def trV-def
by (metis (no-types, lifting) FSM.path-append-elim append-eq-conv-conj
length-take zip-append1)
then show take (length vs) tr1X-all $=\operatorname{tr} V$
using $\operatorname{tr} V$-def by blast
qed
then have $\operatorname{tr} 1 X$-def : path $P M($ io1X $\|$ ?tr1X $)$ ?qv $\wedge$ length io1X $=$ length ? $\operatorname{tr} 1 X$
proof -
have length tr1X-all $=$ length $v s+$ length io1X
using tr1X-all-def by auto
then have length io1X = length tr1X-all - length vs by presburger
then show ?thesis by (metis (no-types) FSM.path-append-elim 〈take (length vs) tr1X-all $=\operatorname{tr} V\rangle$

```
    length-drop tr1X-all-def zip-append1)
qed
then have io1X-lang:io1X \in language-state PM ?qv
    by auto
then obtain tr1\mp@subsup{X}{}{\prime}}\mathrm{ where tr1X'-def : {tr . path PM (io1X|tr) ?qv ^ length io1X = length tr}
                    ={\operatorname{tr}1\mp@subsup{X}{}{\prime}}
    using observable-path-unique-ex[of PM io1X ?qv
        assms(2) assms(3) assms(7) observable-productF
    by blast
moreover have ?tr1X \in{tr . path PM (io1X|tr) ?qv ^ length io1X = length tr }
    using tr1X-def by auto
ultimately have tr1x-unique : tr 1X' =? ?r1X
    by simp
－get a path using io2X from the state reached by vs in PM
```

```
obtain tr2X-all where tr2X-all-def : path PM (vs @ io2X || tr2X-all) (initial PM)
```

obtain tr2X-all where tr2X-all-def : path PM (vs @ io2X || tr2X-all) (initial PM)
$\wedge$ length $(v s @ i o 2 X)=$ length tr2X-all
$\wedge$ length $(v s @ i o 2 X)=$ length tr2X-all
using rep-ios-def rep-ios-split by auto
using rep-ios-def rep-ios-split by auto
let ?tr2 $X=$ drop (length vs) tr2X-all
let ?tr2 $X=$ drop (length vs) tr2X-all
have take (length vs) tr2X-all $=\operatorname{tr} V$
have take (length vs) tr2X-all $=\operatorname{tr} V$
proof -
proof -
have path PM (vs || take (length vs) tr2X-all) (initial PM)
have path PM (vs || take (length vs) tr2X-all) (initial PM)
$\wedge$ length vs $=$ length (take (length vs) tr2X-all)
$\wedge$ length vs $=$ length (take (length vs) tr2X-all)
using tr2X-all-def trV-def
using tr2X-all-def trV-def
by (metis (no-types, lifting) FSM.path-append-elim append-eq-conv-conj
by (metis (no-types, lifting) FSM.path-append-elim append-eq-conv-conj
length-take zip-append1)
length-take zip-append1)
then show take (length vs) tr2X-all $=t r V$
then show take (length vs) tr2X-all $=t r V$
using $t r V$-def by blast
using $t r V$-def by blast
qed
qed
then have tr2X-def : path PM (io2X || ?tr2X) ?qv $\wedge ~ l e n g t h ~ i o 2 X ~=~ l e n g t h ~ ? t r 2 X ~$
then have tr2X-def : path PM (io2X || ?tr2X) ?qv $\wedge ~ l e n g t h ~ i o 2 X ~=~ l e n g t h ~ ? t r 2 X ~$
proof -
proof -
have length tr2X-all $=$ length $v s+$ length io2 $X$
have length tr2X-all $=$ length $v s+$ length io2 $X$
using tr2X-all-def by auto
using tr2X-all-def by auto
then have length io2 $X=$ length tr2 $X$-all - length $v s$
then have length io2 $X=$ length tr2 $X$-all - length $v s$
by presburger
by presburger
then show ?thesis
then show ?thesis
by (metis (no-types) FSM.path-append-elim 〈take (length vs) tr2X-all $=\operatorname{trV}\rangle$
by (metis (no-types) FSM.path-append-elim 〈take (length vs) tr2X-all $=\operatorname{trV}\rangle$
length-drop tr2X-all-def zip-append1)
length-drop tr2X-all-def zip-append1)
qed
qed
then have io2X-lang : io2X $\in$ language-state PM ?qv by auto
then have io2X-lang : io2X $\in$ language-state PM ?qv by auto
then obtain tr2 $X^{\prime}$ where $\operatorname{tr2} X^{\prime}$-def : $\{$ tr . path $P M($ io2 $X \| t r)$ ?qv $\wedge$ length io2 $X=$ length tr $\}$
then obtain tr2 $X^{\prime}$ where $\operatorname{tr2} X^{\prime}$-def : $\{$ tr . path $P M($ io2 $X \| t r)$ ?qv $\wedge$ length io2 $X=$ length tr $\}$
$=\left\{\operatorname{tr} 2 X^{\prime}\right\}$
$=\left\{\operatorname{tr} 2 X^{\prime}\right\}$
using observable-path-unique-ex[of PM io2X ?qv] assms(2) assms(3) assms(7) observable-productF
using observable-path-unique-ex[of PM io2X ?qv] assms(2) assms(3) assms(7) observable-productF
by blast
by blast
moreover have ?tr2X $\in\{$ tr . path $P M($ io2 $X \| t r)$ ?qv $\wedge$ length io2 $X=$ length tr $\}$
moreover have ?tr2X $\in\{$ tr . path $P M($ io2 $X \| t r)$ ?qv $\wedge$ length io2 $X=$ length tr $\}$
using tr2X-def by auto
using tr2X-def by auto
ultimately have $\operatorname{tr2x}$-unique : $\operatorname{tr} 2 X^{\prime}=? \operatorname{tr} 2 X$
ultimately have $\operatorname{tr2x}$-unique : $\operatorname{tr} 2 X^{\prime}=? \operatorname{tr} 2 X$
by $\operatorname{simp}$

```
    by \(\operatorname{simp}\)
```

- both paths reach the same state
have io-targets $P M($ initial $P M)(v s @ i o 1 X)=\{(s$, rep $)\}$
using rep-state rep-ios-split by auto
moreover have io-targets $P M($ initial $P M)$ vs $=\{? q v\}$
using assms (8) by auto
ultimately have rep-via-1 : io-targets $P M$ ? $q v$ io $1 X=\{(s$, rep $)\}$
by (meson obs-PM observable-io-targets-split)
then have rep-tgt-1 : target (io1X\|tr1X') ?qv $=(s$, rep $)$
using obs-PM observable-io-target-unique-target[of PM ?qv io1X (s,rep)] tr1X'-def by blast
have length-1 : length (io1X $\left.\| \operatorname{tr} 1 X^{\prime}\right)>0$
using 〈length vs < length io1〉 rep-ios-split tr1X-def tr1x-unique by auto
have $\operatorname{tr} 1 X$-alt-def : tr1 $X^{\prime}=$ take (length io $1 X$ ) tr
by (metis (no-types) assms(10) assms(9) obs-PM observable-path-prefix qv-simp
rep-ios-split $\operatorname{tr} 1 X$-def $\operatorname{tr} 1 x$-unique)

```
moreover have io \(1 X=\) take (length io1X) xs
    using rep-ios-split by (metis append-eq-conv-conj prefixE)
ultimately have \(\left(\right.\) io1X \(\left.\| \operatorname{tr} 1 X^{\prime}\right)=\) take (length io1X) \((x s \| t r)\)
    by (metis take-zip)
moreover have length ( \(x s \| \operatorname{tr}\) ) \(\geq\) length (io1X \(\|\) tr1X')
    by (metis (no-types) \(\langle i o 1 X=\) take (length io1X) xs〉 assms(10) length-take length-zip
        nat-le-linear take-all tr1X-def tr1x-unique)
ultimately have rep-idx-1: (states \((x s \| t r)\) ?qv) ! ((length io1X) - 1\()=(s, r e p)\)
    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 \(\operatorname{tr1X-def} \operatorname{tr} 1 x\)-unique)
```

    have io-targets PM (initial PM) (vs @ io2X) \(=\{(s\), rep \()\}\)
    using rep-state rep-ios-split by auto
    moreover have io-targets \(P M(\) initial \(P M) v s=\{? q v\}\)
    using assms(8) by auto
    ultimately have rep-via-2 : io-targets \(P M\) ? \(q v\) io \(2 X=\{(s\), rep \()\}\)
        by (meson obs-PM observable-io-targets-split)
    then have rep-tgt-2: target (io2X || tr2X') ?qv = (s,rep)
        using obs-PM observable-io-target-unique-target \([\) of \(P M\) ?qv io2X ( \(s\), rep \()]\) tr2X'-def by blast
    moreover have length-2 : length (io2X \(\left.\| \operatorname{tr2} X^{\prime}\right)>0\)
        by (metis 〈length vs < length io1〉 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 \({ }^{\prime}=\) 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 io2 \(X=\) take (length io2X) xs
    using rep-ios-split by (metis append-eq-conv-conj prefixE)
ultimately have (io2X \|tr2X') = take (length io2X) (xs \|tr)
        by (metis take-zip)
moreover have length (xs \|tr) \(\geq\) length (io2X || tr2X')
    using calculation by auto
ultimately have rep-idx-2: (states \((x s \| t r)\) ?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 \(\operatorname{tr2X}\)-def tr2x-unique)
```

－thus the distinctness assumption is violated

```
    have length io1X F length io2X
    by (metis }\langleio1X=\mathrm{ take (length io1X) xs`<io2X = take (length io2X) xs` less-irrefl
                rep-ios-def rep-ios-split)
    moreover have (states (xs | tr) ?qv)! ((length io1X) - 1)
                        =(states (xs | tr)?qv)! ((length io2X) - 1)
        using rep-idx-1 rep-idx-2 by simp
    ultimately have }\neg(\mathrm{ distinct (states (xs | tr) ?qv))
        by (metis Suc-less-eq <io1X = take (length io1X) xs`
        <io1X | tr1\mp@subsup{X}{}{\prime}= take (length io1X) (xs | tr)\rangle\langleio2X = take (length io2X) xs>
        <io2X | tr2X' = take (length io2X) (xs | tr)
        <length (io1X | tr1X') \leq length (xs | tr)><length (io2X | tr2X') \leq length (xs | tr)>
        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 $x s)$
and observable M2
shows io-targets M2 (initial M2) io $=\{s\}$

```
proof -
    have s io-targets M2 (initial M2) io
        using assms(1) 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(2) io-targets-observable-singleton-ex by (metis singletonD)
qed
```

lemma $R$-union-card-is-suffix-length :
assumes OFSM M2
and $\quad i o @ x s \in L M 2$
shows $\operatorname{sum}(\lambda q$. card ( $R$ M2 $q$ io $x s)$ ) (nodes M2) $=$ length $x s$
using assms proof (induction xs rule: rev-induct)
case Nil
show ?case
by (simp add: sum.neutral)
next
case (snoc $x x s$ )
have finite (nodes M2)
using assms by auto
have R-update : $\bigwedge q$. R M2 $q$ io $(x s @[x])=($ if $(q \in$ io-targets M2 (initial M2) (io @ xs @ $[x]))$
then insert (io@xs@[x]) (R M2 q io xs)
else $R$ M2 $q$ io $x s$ )
by auto
obtain $q$ where io-targets M2 (initial M2) (io @ xs @ $[x]$ ) $=\{q\}$
by (meson assms(1) io-targets-observable-singleton-ex snoc.prems(2))
then have $R$ M2 q io $(x s @[x])=$ insert $($ io@xs@ $[x])(R M 2$ q io xs $)$
using $R$-update by auto
moreover have (io@xs@[x]) $\notin(R$ M2 q io $x s)$
by auto
ultimately have $\operatorname{card}(R$ M2 q io $(x s @[x]))=\operatorname{Suc}(\operatorname{card}(R M 2 q$ io $x s))$
by (metis card-insert-disjoint finite- $R$ )
have $q \in$ nodes M2
by (metis (full-types) FSM.nodes.initial 〈io-targets M2 (initial M2) (io@xs @ $[x]$ ) = \{q\}〉
insertI1 io-targets-nodes)
have $\forall q^{\prime} . q^{\prime} \neq q \longrightarrow R$ M2 $q^{\prime}$ io $(x s @[x])=R$ M2 $q^{\prime}$ io $x s$
using〈io-targets M2 (initial M2) (io@xs@ $[x])=\{q\}$ >R-update
by auto
then have $\forall q^{\prime} . q^{\prime} \neq q \longrightarrow \operatorname{card}\left(R\right.$ M2 $q^{\prime}$ io $\left.(x s @[x])\right)=\operatorname{card}\left(R M 2 q^{\prime}\right.$ io $\left.x s\right)$
by auto
then have $\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 q io $\left.(x s @[x]))\right)$
$=\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 $q$ io $\left.x s)\right)$
by auto
moreover have ( $\sum q \in$ nodes M2. card ( $R$ M2 q io $(x s @[x])$ ))
$=\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 $q$ io $\left.(x s @[x]))\right)+(\operatorname{card}(R$ M2 qio $(x s @[x])))$
( $\sum q \in$ nodes M2. card ( $R$ M2 q io $x s$ ) )
$=\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 $q$ io $\left.x s)\right)+(\operatorname{card}(R$ M2 q io $x s))$
proof -
have $\forall C c f .\left(\right.$ infinite $\left.C \vee\left(c::^{\prime} c\right) \notin C\right) \vee \operatorname{sum} f C=(f c:: n a t)+\operatorname{sum} f(C-\{c\})$
by (meson sum.remove)
then show ( $\sum q \in$ nodes M2. card ( $R$ M2 q io $(x s @[x])$ )
$=\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 q io $\left.(x s @[x]))\right)+(\operatorname{card}(R$ M2 q io $(x s @[x])))$
( $\sum q \in$ nodes M2. card ( $R$ M2 q io $x s$ ) )
$=\left(\sum q \in(\right.$ nodes M2 $-\{q\}) . \operatorname{card}(R$ M2 $q$ io $\left.x s)\right)+(\operatorname{card}(R$ M2 $q$ io $x s))$
using 〈finite (nodes M2)〉 $\langle q \in$ nodes M2〉 by presburger +
qed
ultimately have ( $\sum q \in$ nodes M2. card ( $R$ M2 q io $(x s @[x])$ ))

```
    = Suc (\sumq\innodes M2.card (R M2 q io xs))
    using <card (R M2 q io (xs@[x])) = Suc (card (R M2 q io xs))>
    by presburger
    have (\sumq\innodes M2. card (R M2 q io xs)) = length xs
    using snoc.IH snoc.prems language-state-prefix[of io@xs [x] M2 initial M2]
    proof -
    show ?thesis
        by (metis (no-types)«(io @ xs) @ [x]\inL M2 \Longrightarrowio @ xs \inL M2`
            <OFSM M2〉<io @ xs @ [x] \inL M2> append.assoc snoc.IH)
    qed
    show ?case
    proof -
        show ?thesis
        by (metis (no-types)
                <(\sumq\innodes M2.card (R M2 q io (xs @ [x]))) = Suc (\sumq\innodes M2.card (R M2 q io xs))>
                <(\sumq\in\mathrm{ nodes M2.card (R M2 q io xs)) = length xs> length-append-singleton)}
    qed
qed
lemma R-state-repetition-via-long-sequence :
    assumes OFSM M
    and card (nodes M)\leqm
    and Suc (m*m)\leqlength xs
    and vs@xs\inLM
shows \exists q| nodes M.card (R Mqvs xs)>m
proof (rule ccontr)
    assume}\neg(\existsq\in\mathrm{ nodes M.m<card (R M q vs xs))
    then have }\forallq\in\mathrm{ nodes M.card (R M q vs xs) }\leq
        by auto
    then have sum (\lambdaq.card (RMqvs xs))(nodes M)\leqsum (\lambdaq.m)(nodes M)
        by (meson sum-mono)
    moreover have sum (\lambda q.m)(nodes M)\leqm*m
        using assms(2) by auto
    ultimately have sum ( }\lambdaq.\operatorname{card}(RMq\mathrm{ vs xs)) (nodes M) <m*m
        by presburger
    moreover have Suc (m*m)\leqsum ( }\lambdaq.\operatorname{card}(RMq\mathrm{ vs xs)) (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 Suc (card (nodes M)*m)\leq length xs
    and vs@xs\inLM
shows }\existsq|\mathrm{ nodes M.card (R M q vs xs) > m
proof (rule ccontr)
    assume }\neg(\existsq\in\mathrm{ nodes M. m< card (R M q vs xs))
    then have }\forallq\in\mathrm{ nodes M.card (RM q vs xs) }\leq
        by auto
    then have sum (\lambdaq.card (RMqvs xs))(nodes M)\leq\operatorname{sum}(\lambdaq.m)(nodes M)
        by (meson sum-mono)
    moreover have sum (\lambda q.m) (nodes M)\leqcard (nodes M)*m
    using assms(2) by auto
    ultimately have sum (\lambdaq.card (R M q vs xs)) (nodes M) \leqcard (nodes M)*m
        by presburger
    moreover have Suc (card (nodes M)*m) \leq sum ( }\lambdaq.\operatorname{card}(RMq\mathrm{ vs xs)) (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 \(R P::\) ('in, 'out, 'state) \(F S M \Rightarrow\) 'state \(\Rightarrow\) ('in \(\times\) 'out) list
    \(\Rightarrow\) ('in \(\times\) 'out) list \(\Rightarrow\) ('in \(\times\) 'out) list set
    \(\Rightarrow\) ('in \(\times\) 'out) list set
```

    where
    \(R P M s\) vs \(x s V^{\prime \prime}=R M\) s vs \(x s\)
                            \(\cup\left\{v s^{\prime} \in V^{\prime \prime}\right.\). io-targets \(M(\) initial \(\left.M) v s^{\prime}=\{s\}\right\}\)
    lemma $R P-$ from- $R$ :
assumes is-det-state-cover M2 $V$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
shows $R P$ M2 s vs xs $V^{\prime \prime}=R$ M2 s vs xs
$\vee\left(\exists v s^{\prime} \in V^{\prime \prime} \cdot v s^{\prime} \notin R\right.$ M2 $s$ vs $x s \wedge R P$ M2 $s$ vs $x s V^{\prime \prime}=$ insert $v s^{\prime}(R M 2 s$ vs $\left.x s)\right)$
proof (rule ccontr)
assume assm : $\neg\left(R P\right.$ M2 s vs xs $V^{\prime \prime}=R$ M2 s vs xs $V$
$\left(\exists v s^{\prime} \in V^{\prime \prime} . v s^{\prime} \notin R\right.$ M2 $s$ vs $x s \wedge R P$ M2 $s$ vs $x s V^{\prime \prime}=$ insert vs ${ }^{\prime}(R$ M2 $s$ vs $\left.\left.x s)\right)\right)$
moreover have $R$ M2 s vs $x s \subseteq R P$ M2 s vs $x s V^{\prime \prime}$
by $\operatorname{simp}$
moreover have $R P$ M2 $s$ vs $x s V^{\prime \prime} \subseteq R$ M2 s vs $x s \cup V^{\prime \prime}$
by auto
ultimately obtain vs1 vs2 where vs-def :
$v s 1 \neq v s 2 \wedge v s 1 \in V^{\prime \prime} \wedge v s 2 \in V^{\prime \prime}$
$\wedge$ vs $1 \notin R$ M2 s vs $x s \wedge$ vs2 $\notin R$ M2 s vs xs
$\wedge$ vs $1 \in R P$ M2 $s$ vs $x s V^{\prime \prime} \wedge v s 2 \in R P$ M2 $s$ vs $x s V^{\prime \prime}$
by blast
then have io-targets M2 (initial M2) vs1 $=\{s\} \wedge$ 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 $\operatorname{simp}$
obtain $f$ where $f$-def: is-det-state-cover-ass M2 $f \wedge V=f$ ' $d$-reachable M2 (initial M2)
using assms by auto
moreover have $V=\operatorname{image} f$ ( $d$-reachable M2 (initial M2))
using $f$-def by blast
moreover have map fst vs $1 \in V \wedge$ map fst vs2 $\in V$
using assms(2) perm-inputs vs-def by blast
ultimately obtain r1 r2 where $r$-def :
$f r 1=$ map fst vs1 $\wedge r 1 \in d$-reachable M2 (initial M2)
$f r 2=$ map fst vs2 $\wedge r 2 \in d$-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) vs $1 \subseteq\{r 1\}$
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) vs $2 \subseteq\{r 2\}$
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 $r 1=r 2$
using «io-targets M2 (initial M2) vs1 $=\{s\} \wedge$ io-targets M2 (initial M2) vs2 $=\{s\}\rangle$ by auto
have map fst vs $1 \neq$ map fst vs2
using assms(2) perm-inputs-diff vs-def by blast
then have $r 1 \neq r 2$
using $r$ - $\operatorname{def}(1) r-\operatorname{def}(2)$ by force
then show False
using $\langle r 1=r 2\rangle$ by auto
qed

## lemma finite- $R P$ :

assumes is-det-state-cover M2 $V$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
shows finite ( $R P$ M2 s vs xs $V^{\prime \prime}$ )
using assms $R P$-from- $R$ finite- $R$ by (metis finite-insert)

```
lemma RP-count :
    assumes (vs@ xs)\inLM1\capL M2
    and observable M1
    and observable M2
    and well-formed M1
    and well-formed M2
    and }s\in\mathrm{ 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 }\mp@subsup{V}{}{\prime\prime}\in\mathrm{ Perm V M1
    and }\forall\mp@subsup{s}{}{\prime}\in\mathrm{ set (states (xs| map fst tr) q2). ᄀ(ヨ v | V . d-reaches M2 (initial M2) v s')
shows card (U (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 calcualted 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 \(R P\)-cases: \(R P\) M2 s vs xs \(V^{\prime \prime}=R\) M2 \(s\) vs xs
    \(\vee\left(\exists v s^{\prime} \in V^{\prime \prime} . v s^{\prime} \notin R M 2\right.\) s vs \(x s\)
        \(\wedge R P\) M2 \(s\) vs \(x s V^{\prime \prime}=\operatorname{insert} v s^{\prime}(R\) M2 \(s\) vs \(\left.x s)\right)\)
    using \(R P\)-from- \(R\) assms by metis
show ?thesis
proof (cases RP M2 s vs xs \(V^{\prime \prime}=R\) M2 \(s\) vs \(x s\) )
    case True
    then show ?thesis using \(R\)-count assms by metis
next
    case False
    then obtain \(v s^{\prime}\) where \(v s^{\prime}-d e f: v s^{\prime} \in V^{\prime \prime}\)
                        \(\wedge v s^{\prime} \notin R\) M2 s vs xs
                        \(\wedge R P\) M2 \(s\) vs \(x s V^{\prime \prime}=\) insert vs \({ }^{\prime}(R\) M2 \(s\) vs \(x s)\)
        using \(R P\)-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) \(i o=\{s\}\)
    proof
        fix io assume \(i o \in R\) 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) \(i o=\{s\}\)
            using assms(3) io-targets-observable-singleton-ex by (metis singletonD)
    qed
    have \(v s^{\prime} \in L\) M1
        using assms(13) perm-language \(v s^{\prime}\)-def by blast
    then obtain \(s^{\prime}\) where \(s^{\prime}\)-def: io-targets M1 (initial M1) vs \({ }^{\prime}=\left\{s^{\prime}\right\}\)
```

by（meson assms（2）io－targets－observable－singleton－ob）

```
moreover have \(s^{\prime} \notin \bigcup\) (image (io-targets M1 (initial M1)) (R M2 s vs xs))
proof (rule ccontr)
assume \(\neg s^{\prime} \notin \bigcup(\) io-targets M1 (initial M1) ' R M2 s vs xs)
then obtain \(x s^{\prime}\) where \(x s^{\prime}\)-def : vs @ xs \(s^{\prime} \in R\) M2 svs xs \(\wedge s^{\prime} \in\) io-targets M1 (initial M1) (vs @ xs')
proof -
    assume a1: \(\bigwedge x s^{\prime}\). vs @ \(x s^{\prime} \in R\) M2 s vs \(x s \wedge s^{\prime} \in\) io-targets M1 (initial M1) (vs @ xs \({ }^{\prime}\) )
                \(\Longrightarrow\) thesis
    obtain pps :: ('a×'b) list set \(\Rightarrow\left(\left(^{\prime} a \times{ }^{\prime} b\right)\right.\) list \(\Rightarrow{ }^{\prime} c\) set \() \Rightarrow{ }^{\prime} c \Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)\) list
        where
        \(\forall x 0 x 1 x 2 .(\exists v 3 . v 3 \in x 0 \wedge x 2 \in x 1 v 3)=(p p s x 0 x 1 x 2 \in x 0 \wedge x 2 \in x 1(p p s x 0 x 1 x 2))\)
        by moura
    then have f2: pps ( \(R\) M2 s vs xs) (io-targets M1 (initial M1)) \(s^{\prime} \in R\) M2 s vs xs
                \(\wedge s^{\prime} \in\) io-targets M1 (initial M1) (pps ( \(R\) M2 s vs xs)
                            (io-targets M1 (initial M1)) \(s^{\prime}\) )
        using \(\left\langle\neg s^{\prime} \notin \bigcup(\right.\) io-targets M1 (initial M1) ' \(R\) M2 s vs xs) 〉 by blast
    then have \(\exists \mathrm{ps}\). pps ( \(R\) M2 svs xs) (io-targets M1 (initial M1)) \(s^{\prime}=v s @ p s\)
                \(\wedge p s \neq[] \wedge\) prefix ps xs \(\wedge s \in\) io-targets M2 (initial M2) (vs@ \(p s\) )
        by \(\operatorname{simp}\)
    then show ?thesis
        using f2 a1 by (metis (no-types))
qed
then obtain \(t r^{\prime}\) where \(t r^{\prime}\)-def : path M2 (vs @ \(x s^{\prime} \| t r^{\prime}\) ) (initial M2)
                \(\wedge\) length tr \({ }^{\prime}=\) length (vs @ \(x s^{\prime}\) )
    by auto
```

then obtain $\operatorname{tr} V^{\prime} \operatorname{tr} X^{\prime}$ where $t r^{\prime}$-split : tr $V^{\prime}=$ take (length vs) $t r^{\prime}$
tr $X^{\prime}=$ drop (length vs) $t r^{\prime}$
$t r^{\prime}=t r V^{\prime} @ t r X^{\prime}$
by fastforce
then have path M2 (vs \|trV ) (initial M2) $\wedge$ length $\operatorname{tr} V^{\prime}=$ length vs
by (metis (no-types) FSM.path-append-elim 〈trV'= take (length $v s$ ) tr'〉
append-eq-conv-conj length-take tr'-def zip-append1)
have initial $P M=($ initial M2, initial M1 $)$
using assms(7) by simp
moreover have vs $\in L$ M2 vs $\in L M 1$
using assms(1) language-state-prefix by auto
ultimately have io-targets M1 (initial M1) vs $=\{q 1\}$
io-targets M2 (initial M2) vs $=\{q 2\}$
using productF-path-io-targets[of M2 M1 FAIL PM initial M2 initial M1 vs q2 q1]
by (metis FSM.nodes.initial assms(7) assms(8) assms(2) assms(3) assms(4) assms(5)
io-targets-observable-singleton-ex singletonD)+
then have target $\left(v s \| t r V^{\prime}\right)($ initial M2) $)=q 2$
using «path M2 (vs \|trV') (initial M2) ^ length tr $V^{\prime}=$ length vs〉 io-target-target
by metis
then have path-xs ${ }^{\prime}$ : path M2 $\left(x s^{\prime} \| \operatorname{tr} X^{\prime}\right) ~ q 2 \wedge$ length tr $X^{\prime}=$ length $x s^{\prime}$
by (metis (no-types) FSM.path-append-elim
<path M2 (vs \| trV$\left.V^{\prime}\right)\left(\right.$ initial M2) $\wedge$ length $\operatorname{tr} V^{\prime}=$ length vs〉
$\langle$ target (vs \| trV') (initial M2) $=q 2\rangle$ append-eq-conv-conj length-drop tr'-def
tr'-split (1) $t r^{\prime}$-split(2) zip-append2)
have io-targets M2 (initial M2) (vs @ $\left.x s^{\prime}\right)=\{s\}$
using state-component-2 xs'-def by blast
then have io-targets M2 q2 xs ${ }^{\prime}=\{s\}$
by (meson assms(3) observable-io-targets-split 〈io-targets M2 (initial M2) vs $=\{q 2\}\rangle$ )
then have target-xs' : target $\left(x s^{\prime} \| \operatorname{tr} X^{\prime}\right) q 2=s$
using io-target-target path-xs' by metis
moreover have length $x s^{\prime}>0$
using $x s^{\prime}-d e f$ by auto

```
ultimately have last (states \(\left.\left(x s^{\prime} \| \operatorname{tr} X^{\prime}\right) q 2\right)=s\)
    using path-xs' target-in-states by metis
moreover have length (states \(\left.\left(x s^{\prime} \| t r X^{\prime}\right) q 2\right)>0\)
    using \(\left\langle 0<\right.\) length \(\left.x s^{\prime}\right\rangle\) path-xs' by auto
ultimately have states-xs \(: s \in\) set (states \(\left.\left(x s^{\prime} \| \operatorname{tr} X^{\prime}\right) q 2\right)\)
    using last-in-set by blast
have vs @ \(x s \in L M 2\)
    using assms by simp
then obtain \(q^{\prime}\) where io-targets M2 (initial M2) (vs@xs) \(=\left\{q^{\prime}\right\}\)
    using io-targets-observable-singleton-ob[of M2 vs@xs initial M2] assms(3) by auto
then have \(x s \in\) language-state M2 q2
    using assms(3)〈io-targets M2 (initial M2) vs \(=\{q 2\}\) 〉
        observable-io-targets-split[of M2 initial M2 vs xs \(q^{\prime}\) q2]
    by auto
moreover have path \(P M\) ( \(x s \|\) map fst tr \| map snd tr) (q2,q1)
            \(\wedge\) length \(x s=\) length (map fst tr)
    using assms(7) assms(9) assms(10) productF-path-unzip by simp
moreover have \(x s \in\) language-state \(P M(q 2, q 1)\)
    using assms(9) assms(10) by auto
moreover have \(q 2 \in\) nodes M2
    using 〈io-targets M2 (initial M2) vs \(=\{q 2\}\) 〉io-targets-nodes
    by (metis FSM.nodes.initial insertI1)
moreover have q1 \(\in\) nodes M1
    using 〈io-targets M1 (initial M1) vs \(=\{q 1\}\) 〉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]
        \(\operatorname{assms}(2,3,4,5,7)\)
    by \(\operatorname{simp}\)
```

```
moreover have prefix \(x s^{\prime}\) xs
    using \(x s^{\prime}\)-def by auto
ultimately have \(\operatorname{tr} X^{\prime}=\) take (length \(x s^{\prime}\) ) ( map fst tr)
    using <path \(P M(x s \|\) map fst tr \| map snd tr) (q2, q1) \(\wedge\) length \(x s=\) length (map fst tr) 〉
        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 $f V$ where $f V$-def : is-det-state-cover-ass M2 $f V$
$\wedge V=f V$ ' $d$-reachable M2 (initial M2)
using assms(12) by auto
moreover have $V=$ image $f V$ (d-reachable M2 (initial M2))
using $f V$-def by blast
moreover have map fst vs' $\in V$
using perm-inputs vs'-def assms(13) by metis
ultimately obtain $q v$ where $q v$-def $: f V q v=$ map $f s t v s^{\prime} \wedge q v \in d$-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 ${ }^{\prime} \subseteq\{q v\}$
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 $v s^{\prime}$-def by (metis (mono-tags, lifting) RP.simps Un-iff insertI1 mem-Collect-eq)
ultimately have $q v=s$
by $\operatorname{simp}$
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 \(image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs)))
            = insert s'(U (image (io-targets M1 (initial M1)) (R M2 s vs xs)))
    using }\mp@subsup{s}{}{\prime}\mathrm{ -def by simp
    moreover have finite (U (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 \bigwedgea.a\inR M2 s vs xs \Longrightarrow finite (io-targets M1 (initial M1) a)
    proof -
            fix a assume }a\inR\mathrm{ M2 s vs xs
            then have prefix a (vs@xs)
            by auto
        then have a\inL 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 (U (image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs))))
                    = Suc (card (U (image (io-targets M1 (initial M1)) (R M2 s vs xs))))
        by (metis (no-types) card-insert-disjoint)
    moreover have card (U (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 (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''t)))
                        = Suc (card (U(image (io-targets M1 (initial M1)) (R M2 s vs xs))))
        by linarith
    then have card (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'I')))
            =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(RPM2 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@ @s)\inL M1\capL 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 $V^{\prime \prime} \in$ Perm V M1
and $i o \in R P$ M2 $s$ vs $x s V^{\prime \prime}$
shows io－targets $P M$（initial PM）io
$=$ io－targets M2（initial M2）io $\times$ io－targets M1（initial M1）io
proof－
have $R P$－cases ：$R P$ M2 $s$ vs $x s V^{\prime \prime}=R$ M2 $s$ vs xs

$$
\begin{aligned}
& \vee\left(\exists v s^{\prime} \in V^{\prime \prime} \cdot v s^{\prime} \notin R\right. \text { M2 s vs xs } \\
& \left.\wedge R P \text { M2 s vs xs } V^{\prime \prime}=\text { insert vs' }(R \text { M2 s vs xs })\right)
\end{aligned}
$$

using $R P$－from－$R$ assms by metis
show ？thesis
proof（cases io $\in R$ M2 s vs xs）
case True
then have io－prefix：prefix io（vs＠xs）
by auto
then have io－lang－subs ：io $\in L M 1 \wedge i o \in L M 2$
using assms（1）unfolding prefix－def by（metis IntE language－state language－state－split）
then have io－lang－inter ：io $\in L M 1 \cap L$ M2
by $\operatorname{simp}$
then have io－lang－pm：io $\in L P M$
using productF－language assms by blast
moreover obtain $p 2$ p1 where $(p 2, p 1) \in$ io－targets $P M($ initial $P M)$ 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 $P M($ initial $P M)$ io $=\{(p 2, p 1)\}$
using assms io－targets－observable－singleton－ex singletonD
by（metis observable－productF）
then obtain $\operatorname{tr} P$ where $\operatorname{trP}$－def ：target $($ io $\| \operatorname{tr} P)($ initial $P M)=(p 2, p 1)$
$\wedge$ path $P M($ io $\| \operatorname{tr} P)($ initial $P M) \wedge$ length io $=$ length $\operatorname{tr} P$
proof－
assume a1：$\bigwedge$ tr $P$ ．target $($ io $\| \operatorname{tr} P)($ initial $P M)=(p 2, p 1)$
$\wedge$ path $P M($ io $\| \operatorname{tr} P)($ initial $P M) \wedge$ length io $=$ length $\operatorname{tr} P \Longrightarrow$ thesis
have $\exists \mathrm{ps}$ ．target $($ io $\| p s)($ initial $P M)=(p 2, p 1) \wedge$ path $P M($ io $\| p s)($ initial $P M)$ $\wedge$ length io $=$ length $p s$
using $\langle(p 2, p 1) \in$ io－targets $P M($ initial $P M)$ io $>$ by auto
then show ？thesis
using a1 by blast
qed
then have $\operatorname{tr} P$－unique ：$\{$ tr ．path $P M($ io $\| \operatorname{tr})($ initial $P M) \wedge$ length io $=$ length $\operatorname{tr}\}=\{\operatorname{tr} P\}$
using observable－productF observable－path－unique－ex［of PM io initial PM］
io－lang－pm assms（2）assms（3）assms（7）
proof－
obtain pps ：：（ $\left.{ }^{\prime} d \times{ }^{\prime} c\right)$ list where
f1：$\{$ ps．path $P M($ io $\| p s)($ initial $P M) \wedge$ length io $=$ length $p s\}=\{p p s\}$
$\vee \neg$ observable $P M$
by（metis（no－types）＜$\backslash$ thesis．$\llbracket$ observable $P M ;$ io $\in L P M ; \bigwedge$ tr．
$\{t$ ．path $P M($ io $\| t)($ initial $P M) \wedge$ length io $=$ length $t\}=\{t r\}$
$\Longrightarrow$ thesis $\Longrightarrow$ thesis
io－lang－pm）
have f2：observable PM
by（meson 〈observable M1〉＜observable M2〉 〈productF M2 M1 FAIL PM〉 observable－productF）
then have $\operatorname{tr} P \in\{p p s\}$
using $f 1$ tr $P$－def by blast
then show ？thesis
using f2 f1 by force
qed
obtain trIO2 where trIO2－def ：\｛tr ．path M2（io \｜tr）（initial M2）$\wedge$ length io $=$ length $\operatorname{tr}\}$ $=\{$ trIO2 $\}$
using observable－path－unique－ex［of M2 io initial M2］io－lang－subs assms（3）by blast
obtain trIO1 where trIO1－def ：\｛tr ．path M1（io \｜tr）（initial M1）$\wedge$ length io $=$ length $\operatorname{tr}\}$ $=\{\operatorname{trIO} 1\}$
using observable－path－unique－ex［of M1 io initial M1］io－lang－subs assms（2）by blast

```
have path PM (io || trIO2 || trIO1) (initial M2, initial M1)
        \wedge ~ l e n g t h ~ i o ~ = ~ l e n g t h ~ t r I O 2 ~ \wedge ~ l e n g t h ~ t r I O 2 ~ = ~ l e n g t h ~ t r I O 1 ~
    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 \in 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\in io-targets M1 (initial M1) io
    by auto
then have targets-1 : io-targets M1 (initial M1) io = {p1}
    by (metis io-lang-subs assms(2) io-targets-observable-singleton-ex singletonD)
have io-targets M2 (initial M2) io }\times\mathrm{ 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 }\times\mathrm{ io-targets M1 (initial M1) io
    using targets-pm by simp
```


## next

    case False
    then have \(i o \notin R\) M2 s vs \(x s \wedge R P\) M2 s vs \(x s V^{\prime \prime}=\) insert io ( \(R\) M2 s vs \(x s\) )
        using \(R P\)-cases assms( 9 ) by (metis insertE)
    have io \(\in L\) M1 using assms(8) perm-language assms(9)
        using False by auto
    then obtain \(s^{\prime}\) where \(s^{\prime}\)-def : io-targets M1 (initial M1) io \(=\left\{s^{\prime}\right\}\)
        by (meson assms(2) io-targets-observable-singleton-ob)
    then obtain tr1 where tr1-def : target \((\) io \(\| \operatorname{tr} 1)(\) initial M1) \()=s^{\prime}\)
                                    \(\wedge\) path M1 (io \|tr1) (initial M1) \(\wedge\) length tr1 = length io
        by (metis io-targets-elim singletonI)
    have io-targets M2 (initial M2) io \(=\{s\}\)
    using \(\operatorname{assms}(9) \operatorname{assms}(3) R P\)-state-component-2 by simp
    then obtain tr2 where tr2-def : target (io \|tr2) (initial M2) \(=s\)
                    \(\wedge\) path M2 (io \|tr2) (initial M2) \(\wedge\) length tr2 \(=\) length io
        by (metis io-targets-elim singletonI)
    then have paths : path M2 (io \|| tr2) (initial M2) \(\wedge\) 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,\mp@subsup{s}{}{\prime})\in\mathrm{ io-targets }PM(\mathrm{ 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,\mp@subsup{s}{}{\prime})}
    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@ @s)\inLM1\capLM2
    and observable M1
    and is-det-state-cover M2 V
    and V}\mp@subsup{V}{}{\prime\prime}\inP\mathrm{ Perm V M1
shows finite (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'')))
proof
    show finite (RP M2 s vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ ) using finite-RP assms(3) assms(4) by simp
    show \a. a \in RP M2 s vs xs V'" \Longrightarrow finite (io-targets M1 (initial M1) a)
    proof -
        fix a assume a\inRP M2 s vs xs V'
        have RP-cases:RP M2 s vs xs }\mp@subsup{V}{}{\prime\prime}=R\mathrm{ M2 s vs xs
                        \vee (\existsvs'}\in\mp@subsup{|}{}{\prime\prime}.v\mp@subsup{s}{}{\prime}\not\inRM2 s vs xs
                        \wedge RP M2 s vs xs }\mp@subsup{V}{}{\prime\prime}=insert vs'( (R M2 s vs xs)) (
        using RP-from-R assms by metis
        have a\inL M1
        proof (cases a\inR M2 s vs xs)
            case True
            then have prefix a(vs@xs)
            by auto
            then show }a\inLM
            using language-state-prefix by (metis IntD1 assms(1) prefix-def)
        next
            case False
            then have a\in V'^}\wedgeRP M2 s vs xs V V'=insert a (R M2 s vs xs
            using RP-cases }\langlea\inRPM2 s vs xs V V'> by (metis insertE)
            then show }a\inLM
            by (meson assms(4) perm-language)
    qed
    then obtain p}\mathrm{ 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
lemma RP-io-targets-finite-PM :
    assumes (vs @ xs) \inL 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 }\mp@subsup{V}{}{\prime\prime}\in\mathrm{ Perm V M1
shows finite (U (image (io-targets PM (initial PM)) (RP M2 s vs xs V }\mp@subsup{V}{}{\prime\prime}))
proof -
    have }\forallio\inRPM2 s vs xs V' ' io-targets PM (initial PM) i
                                    ={s}\times io-targets M1 (initial M1) io
    proof
    fix io assume io }\inRP\mathrm{ M2 s vs xs }\mp@subsup{V}{}{\prime\prime
    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'l io s] by simp
    moreover have io-targets M2 (initial M2) io = {s}
        using <io \inRP M2 s vs xs V V'〉 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\mathrm{ io-targets M1 (initial M1) io
        by auto
    qed
    then have U(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
```



```
                ={s}\timesU(image (\lambda io . io-targets M1 (initial M1) io) (RP M2 s vs xs V''))
        by blast
    ultimately have U(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} }\\bigcup\(\mathrm{ (image (io-targets M1 (initial M1)) (RP M2 s vs xs V }\mp@subsup{V}{}{\prime\prime
        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 $R P$-union-card-is-suffix-length :
assumes OFSM M2
and $\quad i o @ x s \in L M 2$
and is-det-state-cover M2 $V$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
shows $\Lambda q \cdot \operatorname{card}(R$ M2 $q$ io xs $) \leq \operatorname{card}\left(R P\right.$ M2 $q$ io xs $\left.V^{\prime \prime}\right)$
$\operatorname{sum}\left(\lambda q\right.$.card $\left(R P\right.$ M2 $q$ io $\left.\left.x s V^{\prime \prime}\right)\right)($ nodes M2) $\geq$ length $x s$
proof -
have $\operatorname{sum}(\lambda q . \operatorname{card}(R$ M2 $q$ io xs)) (nodes M2) $=$ length $x s$
using $R$-union-card-is-suffix-length $[$ OF $\operatorname{assms}(1,2)]$ by assumption
show $\wedge q \cdot \operatorname{card}(R$ M2 $q$ io $x s) \leq \operatorname{card}\left(R P M 2 q\right.$ io $\left.x s V^{\prime \prime}\right)$
by (metis RP-from-R assms(3) assms(4) card-insert-le eq-iff finite-R)
show $\operatorname{sum}\left(\lambda q . \operatorname{card}\left(R P\right.\right.$ M2 $q$ io $\left.\left.x s V^{\prime \prime}\right)\right)($ nodes M2) $\geq$ length $x s$
by (metis (no-types, lifting) $\left\langle\left(\sum q \in\right.\right.$ nodes M2. card $(R$ M2 $q$ io $\left.x s)\right)=$ length $\left.x s\right\rangle$
« $\backslash q$. $\operatorname{card}(R$ M2 $q$ io $x s) \leq \operatorname{card}\left(R P\right.$ M2 $q$ io $\left.x s V^{\prime \prime}\right)$ 〉sum-mono)
qed
lemma $R P$-state-repetition-distribution-productF :
assumes OFSM M2
and $O F S M M 1$
and $\quad($ card $($ nodes M2) $) * m) \leq$ length $x s$
and $\quad \operatorname{card}($ nodes M1) $\leq m$
and $\quad v s @ x s \in L M 2 \cap L M 1$
and is－det－state－cover M2 $V$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
shows $\exists q \in$ nodes M2 ．card（RP M2 $q$ vs xs $\left.V^{\prime \prime}\right)>m$
proof－
have finite（nodes M1）
finite（nodes M2）
using $\operatorname{assms}(1,2)$ by auto
then have card（nodes M2 $\times$ nodes M1）$=\operatorname{card}$（nodes M2）$* \operatorname{card}$（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）〈card（nodes M2 $\times$ nodes M1）$=|M 2| *|M 1|\rangle\langle$ finite（nodes M1）$\rangle$ $\langle$ finite（nodes M2）〉〈nodes（product M2 M1）$\subseteq$ nodes M2 $\times$ nodes M1〉 card－mono finite－cartesian－product）

```
have \((\forall q \in\) nodes M2 . card \((R\) M2 \(q\) vs \(x s)=m) \vee(\exists q \in\) nodes M2. card \((R\) M2 \(q\) vs \(x s)>m)\)
proof (rule ccontr)
    assume \(\neg((\forall q \in\) nodes M2. card \((R\) M2 \(q\) vs \(x s)=m) \vee(\exists q \in\) nodes M2. \(m<\operatorname{card}(R\) M2 \(q\) vs \(x s)))\)
    then have \(\forall q \in\) nodes M2 . card \((R\) M2 \(q\) vs \(x s) \leq m\)
        by auto
    moreover obtain \(q^{\prime}\) where \(q^{\prime} \in\) nodes M2 card \(\left(R\right.\) M2 \(q^{\prime}\) vs xs) \(<m\)
        using \(\prec \neg((\forall q \in\) nodes M2. card \((R\) M2 \(q\) vs \(x s)=m) \vee(\exists q \in\) nodes M2. \(m<\operatorname{card}(R\) M2 \(q\) vs \(x s)))\rangle\)
            nat-neq-iff
```

        by blast
    have \(\operatorname{sum}(\lambda q . \operatorname{card}(R\) M2 \(q\) vs xs)) (nodes M2)
        \(=\operatorname{sum}(\lambda q \cdot \operatorname{card}(R\) M2 \(q\) vs xs \())\left(\right.\) nodes M2 \(\left.-\left\{q^{\prime}\right\}\right)\)
        \(+\operatorname{sum}(\lambda q \cdot \operatorname{card}(R\) M2 \(q\) vs xs \())\left\{q^{\prime}\right\}\)
    using 〈 \(q^{\prime} \in\) nodes M2〉
    by (meson〈finite (nodes M2)〉 empty-subsetI insert-subset sum.subset-diff)
    moreover have sum ( \(\lambda\) q. card ( \(R\) M2 \(q\) vs xs)) (nodes M2 - \(\left\{q^{\prime}\right\}\) )
                \(\leq \operatorname{sum}(\lambda q \cdot m)\left(\right.\) nodes M2 \(\left.-\left\{q^{\prime}\right\}\right)\)
    using \(\langle\forall q \in\) nodes M2 . card (R M2 \(q\) vs \(x s) \leq m\rangle\)
    by (meson sum-mono DiffD1)
    moreover have sum ( \(\lambda q\). card ( \(R\) M2 \(q\) vs \(x s\) )) \(\left\{q^{\prime}\right\}<m\)
    using <card \(\left(R\right.\) M2 \(q^{\prime}\) vs \(\left.x s\right)<m\) 〉 by auto
    ultimately have \(\operatorname{sum}(\lambda q \cdot \operatorname{card}(R\) M2 \(q\) vs xs) \()(\) nodes M2) \(<\operatorname{sum}(\lambda q \cdot m)(\) nodes M2)
    proof -
    have \(\forall C c f\). infinite \(C \vee\left(c::^{\prime} c\right) \notin C \vee \operatorname{sum} f C=(f c::\) nat \()+\operatorname{sum} f(C-\{c\})\)
        by (meson sum.remove)
    then have \(\left(\sum c \in\right.\) nodes M2. \(\left.m\right)=m+\left(\sum c \in\right.\) nodes M2 \(\left.-\left\{q^{\prime}\right\} . m\right)\)
        using 〈finite (nodes M2)〉〈 \(q^{\prime} \in\) nodes M2〉 by blast
    then show ?thesis
        using \(\left\langle\left(\sum q \in\right.\right.\) nodes M2 \(-\left\{q^{\prime}\right\}\). card \((R\) M2 q vs \(\left.x s)\right) \leq\left(\sum q \in\right.\) nodes M2 \(\left.\left.-\left\{q^{\prime}\right\} . m\right)\right\rangle\)
                    \(\left\langle\left(\sum q \in\right.\right.\) nodes M2. card ( \(R\) M2 \(q\) vs \(\left.x s\right)\) ) \(=\left(\sum q \in\right.\) nodes M2 \(-\left\{q^{\prime}\right\}\). card ( \(R\) M2 \(q\) vs \(\left.x s\right)\) )
                    \(+\left(\sum q \in\left\{q^{\prime}\right\} . \operatorname{card}(R\right.\) M2 \(q\) vs \(\left.x s)\right)\) )
                \(\left\langle\left(\sum q \in\left\{q^{\prime}\right\} . \operatorname{card}(R\right.\right.\) M2 \(q\) vs \(\left.x s)\right)<m>\)
        by linarith
    qed
    moreover have \(\operatorname{sum}(\lambda q \cdot m)(\) nodes M2) \(\leq \operatorname{card}\) (nodes M2) \(* m\)
    using assms(2) by auto
    ultimately have sum \((\lambda q . \operatorname{card}(R\) M2 \(q\) vs xs \()\) ) (nodes M2) \(<\operatorname{card}(\) nodes M2) \(* m\)
    by presburger
    moreover have Suc（card（nodes M2）$* m$ ）$\leq \operatorname{sum}(\lambda q \cdot \operatorname{card}(R$ 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 $\left\langle v s @ x s \in L\right.$ M2 $\Longrightarrow\left(\sum q \in\right.$ nodes M2．card $(R$ M2 q vs xs $\left.)\right)=$ length xs $\langle v s @ x s \in L$ M2 $\cap L M 1\rangle\langle | M 2 \mid * m \leq$ length $x s\rangle$ calculation less－eq－Suc－le not－less－eq－eq）

```
    ultimately show False by simp
qed
then show ?thesis
proof
    let ? q = initial M2
    assume }\forallq\in\mathrm{ nodes M2.card (R M2 q vs xs) =m
    then have card (R M2 ? q vs xs) = m
        by auto
    have [] \in V ''
        by (meson assms(6) assms(7) perm-empty)
    then have [] \in RP M2 ?q vs xs V''
        by auto
    have []\not\inR M2 ?q vs xs
        by auto
    have card (RP M2 ?q vs xs V}\mp@subsup{V}{}{\prime\prime})\geq\operatorname{card (R M2 ?q vs 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 card (RP M2 ?q vs xs V') > card (R M2 ?q vs xs)
    proof -
        have f1:\foralln na. (\neg (n::nat) \leqna\vee n=na)\veen<na
        by (meson le-neq-trans)
    have RP M2 (initial M2) vs xs }\mp@subsup{V}{}{\prime\prime}\not=R M2 (initial M2) vs x
        using<[] < RP M2 (initial M2) vs xs V''>< <] # R M2 (initial M2) vs xs> by blast
    then show ?thesis
        using f1 by (metis (no-types) RP-from-R
                        <card (R M2 (initial M2) vs xs) \leq card (RP M2 (initial M2) vs xs V')}\mp@subsup{V}{}{\prime\prime
                        assms(6) assms(7) card-insert-disjoint finite-R le-simps(2))
    qed
    then show ?thesis
        using <card (R M2 ?q vs xs) = m>
        by blast
next
    assume \existsq\innodes M2. m<card (R M2 q vs xs)
    then obtain q}\mathrm{ where qGnodes M2 m<card (R M2 q vs xs)
        by blast
    moreover have card (RP M2 q vs xs }\mp@subsup{V}{}{\prime\prime})\geq\operatorname{card}(R M2 q vs 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<card (RP M2 q vs xs V'')
        by simp
    show ?thesis
    using <q\in nodes M2` <m<card (RP M2 q vs xs V'')> 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 => ('in, 'out, 'state2) FSM => ('in × 'out) list
    #('in }\times\mathrm{ 'out) list }=>\mathrm{ ''in list set }=>\mathrm{ ''state1 set }=>\mathrm{ (' 'in, 'out) ATC set
    ('in }\times\mathrm{ 'out) list set }=>\mathrm{ bool
where
```

Prereq M2 M1 vs xs $T S \Omega V^{\prime \prime}=($
(finite $T$ )
$\wedge(v s @ x s) \in L M 2 \cap L M 1$
$\wedge S \subseteq$ nodes M2
$\wedge(\forall s 1 \in S . \forall s \mathcal{Z} \in S . s 1 \neq s \mathcal{Z}$
$\longrightarrow\left(\forall\right.$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall i o 2 \in R P$ M2 s2 vs $x s V^{\prime \prime}$.

```
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 \(x s \wedge\) xs1 \(\neq\) xs2
    \(\wedge(\exists\) s2 . io-targets M2 (initial M2) \((v s @ x s 1)=\{s 2\}\)
        \(\wedge\) io-targets M2 (initial M2) (vs @ xs2) \(=\{s 2\})\)
    \(\wedge(\exists\) s1 . io-targets M1 (initial M1) \((v s @ x s 1)=\{s 1\}\)
        \(\wedge\) io-targets M1 (initial M1) \((v s @ x s 2)=\{s 1\}))\)
```

fun Rep-Cov :: ('in, 'out, 'state1) FSM $\Rightarrow$ ('in, 'out, 'state2) $F S M \Rightarrow$ ('in $\times$ 'out) list set
$\Rightarrow$ ('in $\times$ 'out) list $\Rightarrow$ ('in $\times$ 'out $)$ list $\Rightarrow$ bool where
Rep-Cov M2 M1 $V^{\prime \prime}$ vs $x s=\left(\exists x s^{\prime}\right.$ vs $s^{\prime} . x s^{\prime} \neq[] \wedge$ prefix $x s^{\prime} x s \wedge v s^{\prime} \in V^{\prime \prime}$
$\wedge\left(\exists\right.$ s2 . io-targets M2 (initial M2) $\left(v s @ x s^{\prime}\right)=\{s 2\}$
$\wedge$ io-targets M2 (initial M2) $\left.\left(v s^{\prime}\right)=\{s 2\}\right)$
$\wedge\left(\exists\right.$ s1 . io-targets M1 (initial M1) $\left(v s @ x s^{\prime}\right)=\{s 1\}$
$\wedge$ io-targets M1 (initial M1) $\left.\left.\left(v s^{\prime}\right)=\{s 1\}\right)\right)$
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 $P M($ initial $P M)$ vs $=\{(q 2, q 1)\}$
and path $P M(x s \| t r)(q 2, q 1)$
and length $x s=$ length $t r$
and $(v s @ x s) \in L M 1 \cap L M 2$
and well-formed M1
and well-formed M2
shows distinct (states (xs \|tr) (q2, q1))
proof (rule ccontr)
assume assm : ᄀ distinct (states (xs \|tr) (q2, q1))
then obtain $i 1$ i2 where index-def :
$i 1 \neq 0$
$\wedge i 1 \neq i 2$
$\wedge i 1<\operatorname{length}($ states $(x s \| t r)(q 2, q 1))$
$\wedge i 2<\operatorname{length}($ states $(x s \| t r)(q 2, q 1))$
$\wedge($ states $(x s \| t r)(q 2, q 1))!i 1=($ states $(x s \| t r)(q 2, q 1))!i 2$
by (metis distinct-conv-nth)
then have length $x s>0$ by auto
let ? ${ }^{\text {xs1 }}=$ take $($ Suc i1) $) x s$
let ? $x s 2=$ take $($ Suc i2 $) ~ x s$
let $?$ tr1 $=$ take $($ Suc i1) $) t r$
let ? tr2 $=$ take $($ Suc i2) tr
let ?st $=($ states $(x s \| t r)(q 2, q 1))!i 1$
have obs-PM : observable PM
using observable-productF assms(2) assms(3) assms(4) by blast
have initial $P M=($ 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 $=\{q 1\}$ io-targets M2 (initial M2) vs $=\{q 2\}$
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 $(x s \| t r)(q 2, q 1))!i 1 \in$ io-targets $P M(q 2, q 1)$ ? $x s 1$
by（metis $\langle 0<$ length $x s\rangle \operatorname{assms}(6) \operatorname{assms}(7)$ index－def map－snd－zip states－alt－def states－index－io－target）
then have io－targets $P M(q 2, q 1)$ ？$x s 1=\{? s t\}$
using obs－PM by（meson observable－io－target－is－singleton）
have path $P M(? x s 1|\mid$ ？tr1）$(q 2, q 1)$
by（metis FSM．path－append－elim append－take－drop－id assms（6）assms（7）length－take zip－append）
then have path $P M$（？xs1｜｜map fst ？tr1｜｜map snd ？tr1）（q2，q1）
by auto
have vs＠？$x s 1 \in L M 2$
by（metis（no－types）IntD2 append－assoc append－take－drop－id assms（8）language－state－prefix）
then obtain $q 2^{\prime}$ where io－targets M2（initial M2）（vs＠？xs1）$=\left\{q^{2}{ }^{\prime}\right\}$
using io－targets－observable－singleton－ob［of M2 vs＠？xs1 initial M2］assms（4）by auto
then have $q 2^{\prime} \in$ io－targets M2 $q 2$ ？$x s 1$
using $\operatorname{assms}(4)\langle i o$－targets M2（initial M2）$v s=\{q 2\}>$ observable－io－targets－split［of M2 initial M2 vs ？xs1 q2＇q2］
by $\operatorname{simp}$
then have ？xs1 $\in$ 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 $q 2 \in$ nodes M2
using 〈io－targets M2（initial M2）vs $=\{q 2\}>$ io－targets－nodes
by（metis FSM．nodes．initial insertI1）
moreover have $q 1 \in$ 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（？$x s 1 \| \operatorname{map} f s t$ ？tr1）q2
target（？xs1｜｜map snd ？tr1）q1＝snd（target（？ $\mathrm{ms} 1\|\operatorname{map} f s t ? \operatorname{tr} 1\| \operatorname{map}$ snd ？tr1）$(q 2, q 1))$
target（？xs1 \｜map fst ？tr1）q2＝fst（target（？xs1 \｜map fst ？tr1 \｜map snd ？tr1）（q2，q1））
using $\operatorname{assms}(2) \operatorname{assms}(9) \operatorname{assms}(10)$ 〈path $P M(? x s 1 \|$ map fst ？tr1 \｜map snd ？tr1）（q2，q1）〉 $\operatorname{assms}(4)$ productF－path－reverse－ob－2［of ？xs1 map fst？tr1 map snd？tr1 M2 M1 FAIL PM q2 q1］
by $\operatorname{simp}+$
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）$q 2=$ fst ？st
by $\operatorname{simp}+$
— paths for ？xs2
have（states $(x s \| t r)(q 2, q 1))!i 2 \in$ io－targets $P M(q 2, q 1)$ ？$x s 2$
by（metis $\langle 0<l e n g t h ~ x s\rangle \operatorname{assms}(6) \operatorname{assms}(7)$ index－def map－snd－zip states－alt－def states－index－io－target）
then have io－targets $P M(q 2, q 1)$ ？xs $2=\{? s t\}$
using obs－PM by（metis index－def observable－io－target－is－singleton）
have path $P M$（？xs2｜｜？tr2）$(q 2, q 1)$
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 $\in L$ M2
by（metis（no－types）IntD2 append－assoc append－take－drop－id assms（8）language－state－prefix）
then obtain $q 2^{\prime \prime}$ where io－targets M2（initial M2）（vs＠？xs2）$=\left\{q 2^{\prime \prime}\right\}$
using io－targets－observable－singleton－ob［of M2 vs＠？xs2 initial M2］assms（4）
by auto
then have $q 2^{\prime \prime} \in$ io－targets M2 $q 2$ ？$x s 2$
using $\operatorname{assms}(4)$ 〈io－targets M2（initial M2）vs $=\{q 2\} 〉$ observable－io－targets－split［of M2 initial M2 vs ？xs2 q2＂q2］
by $\operatorname{simp}$
then have ？xs $2 \in$ language－state M2 $q 2$
by auto
moreover have length ？ $\mathrm{xs2}=$ length（ map snd ？tr2）using assms（7）
by auto
moreover have length（map fst ？tr2）$=$ length（map snd ？tr2） by auto
moreover have $q 2 \in$ nodes M2
using «io－targets M2（initial M2）vs $=\{q 2\}$ 〉io－targets－nodes
by（metis FSM．nodes．initial insertI1）
moreover have q1 $\in$ nodes M1
using 〈io－targets M1（initial M1）vs $=\{q 1\}$ 〉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）$q 1=\operatorname{snd}(\operatorname{target}(? x s 2 \|$ map fst ？tr2｜｜map snd ？tr2）$(q 2, q 1))$

using assms（2）assms（9）assms（10）«path PM（？xs2｜｜map fst ？tr2｜｜map snd ？tr2）（q2，q1）〉 $\operatorname{assms}$（4）
productF－path－reverse－ob－2［of ？xs2 map fst ？tr2 map snd ？tr2 M2 M1 FAIL PM q2 q1］
by $\operatorname{simp}+$
moreover have target（？xs2 \｜map fst ？tr2 \｜map snd ？tr2）$(q 2, q 1)=$ ？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 $\operatorname{simp}+$

```
have io-targets M1 q1 ? \(x s 1=\{\) snd ? \(s t\}\)
    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 \(\operatorname{simp}\)
then have tgt-1-1: io-targets M1 (initial M1) (vs @ ?xs1) \(=\{\) snd ? st \(\}\)
    by (meson 〈io-targets M1 (initial M1) vs \(=\{q 1\}\) 〉assms(3) observable-io-targets-append)
have io-targets M2 q2 ? \(x s 1=\{f s t\) ?st \(\}\)
    using 〈path M2 (?xs1 || map fst ?tr1) q2〉〈target (?xs1 || map fst ?tr1) q2 = fst ?st〉
        〈length ? \(x s 1=\) length (map snd ?tr1) 〉assms(4)
        obs-target-is-io-targets[of M2 ?xs1 map fst ?tr1 q2]
    by \(\operatorname{simp}\)
then have tgt-1-2 : io-targets M2 (initial M2) (vs @ ?xs1) \(=\{f s t\) ?st \(\}\)
    by (meson 〈io-targets M2 (initial M2) vs \(=\{q 2\}\) 〉assms(4) observable-io-targets-append)
have io-targets M1 q1 ? xs2 \(=\{\) snd ? \(s t\}\)
    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 \(\operatorname{simp}\)
then have tgt-2-1: io-targets M1 (initial M1) (vs @ ?xs2) \(=\{\) snd ?st \(\}\)
    by (meson〈io-targets M1 (initial M1) vs \(=\{q 1\}\) 〉assms(3) observable-io-targets-append)
have io-targets M2 q2 ? \(x s 2=\{f s t\) ?st \(\}\)
    using 〈path M2 (?xs2 || map fst ?tr2) q2〉〈target (?xs2 || map fst ?tr2) q2 = fst ?st〉
        〈length ? \(x s 2=\) length (map snd ?tr2) > assms(4)
        obs-target-is-io-targets[of M2 ?xs2 map fst ?tr2 q2]
    by \(\operatorname{simp}\)
then have tgt-2-2 : io-targets M2 (initial M2) (vs @ ?xs2) \(=\{f s t\) ?st \(\}\)
    by (meson 〈io-targets M2 (initial M2) vs \(=\{q 2\}\) 〉assms(4) observable-io-targets-append)
have ? \(x s 1 \neq[]\) using \(\langle 0<\) length \(x s\rangle\)
    by auto
have prefix ? xs 1 xs
```

using take－is－prefix by blast
have prefix ？xs2 xs
using take－is－prefix by blast
have ？$x s 1 \neq$ ？$x s 2$
proof－
have f1：$\forall n n a . \neg n<n a \vee$ Suc $n \leq n a$
by presburger
have f2：Suc i1 $\leq$ length xs
using index－def by force
have Suc i2 $\leq$ length $x s$
using $f 1$ 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 $\langle p r e f i x$（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）〉
$\langle p r e f i x ~(t a k e ~(S u c ~ i 2) ~ x s) ~ x s 〉<t a k e ~(S u c ~ i 1) ~ x s ~ \neq t a k e ~(S u c ~ i 2) ~ x s 〉 ~$ tgt－1－1 tgt－1－2 tgt－2－1 tgt－2－2）
next
case False
moreover have length ？xs $1 \neq$ length ？$x s 2$
by（metis（no－types）＜take（Suc i1）xs $\neq$ take（Suc i2）xs〉 append－eq－conv－conj append－take－drop－id）
ultimately have length ？xs2＜length ？xs 1
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 $\neq$ 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)\inL M1\capL M2
    and observable M1
    and observable M2
    and well-formed M1
    and well-formed M2
    and s}\in\mathrm{ 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 }\mp@subsup{V}{}{\prime\prime}\inP\mathrm{ Perm V M1
    and }\neg\mathrm{ Rep-Cov M2 M1 V' vs xs
shows card (U(image (io-targets M1 (initial M1)) (RP M2 s vs xs V V'}))=\operatorname{card}(RP M2 s vs xs V V')
proof -
```

have $R P$-cases: RP M2 s vs xs $V^{\prime \prime}=R$ M2 s vs xs

$$
\begin{aligned}
& \vee\left(\exists v s^{\prime} \in V^{\prime \prime} \cdot v s^{\prime} \notin R\right. \text { M2 s vs xs } \\
&\left.\wedge R P \text { M2 s vs xs } V^{\prime \prime}=\text { insert vs' }(R \text { M2 s vs xs })\right)
\end{aligned}
$$

using $R P$-from- $R$ assms by metis
show ?thesis
proof (cases RP M2 s vs xs $V^{\prime \prime}=R$ M2 $s$ vs $x s$ )
case True
then show ?thesis
using $R$-count assms by metis

## next

case False
then obtain $v s^{\prime}$ where $v s^{\prime}$-def : $v s^{\prime} \in V^{\prime \prime}$ $\wedge v s^{\prime} \notin R$ M2 s vs xs $\wedge R P M 2 s$ vs xs $V^{\prime \prime}=$ insert vs ${ }^{\prime}(R M 2 s v s x s)$
using $R P$-cases by auto
have state-component-2 : $\forall$ io $\in(R$ M2 s vs xs). io-targets M2 (initial M2) $i o=\{s\}$
proof
fix io assume $i o \in R$ 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) $i o=\{s\}$
using assms(3) io-targets-observable-singleton-ex by (metis singletonD)
qed
have $v s^{\prime} \in L$ M1
using assms(13) perm-language $v s^{\prime}$-def by blast
then obtain $s^{\prime}$ where $s^{\prime}$-def : io-targets M1 (initial M1) vs' $=\left\{s^{\prime}\right\}$
by (meson assms(2) io-targets-observable-singleton-ob)
moreover have $s^{\prime} \notin \bigcup$ (image (io-targets M1 (initial M1)) (R M2 s vs xs))
proof (rule ccontr)
assume $\neg s^{\prime} \notin \bigcup^{\prime}($ io-targets M1 (initial M1) ' R M2 s vs xs)
then obtain $x s^{\prime}$ where $x s^{\prime}-$ def : vs @ $x s^{\prime} \in R$ M2 s vs xs
$\wedge s^{\prime} \in$ io-targets $M 1$ (initial M1) (vs @ $\left.x s^{\prime}\right)$
proof -
assume a1: $\bigwedge x s^{\prime}$. vs @ $x s^{\prime} \in R$ M2 svs xs
$\wedge s^{\prime} \in$ io-targets M1 (initial M1) (vs @ xs') $\Longrightarrow$ thesis
obtain pps :: (' $a \times{ }^{\prime} b$ ) list set $\Rightarrow\left(\left(^{\prime} a \times{ }^{\prime} b\right)\right.$ list $\Rightarrow{ }^{\prime} c$ set $) \Rightarrow{ }^{\prime} c \Rightarrow\left({ }^{\prime} a \times{ }^{\prime} b\right)$ list where $\forall x 0 x 1 x 2 .(\exists v 3 . v 3 \in x 0 \wedge x 2 \in x 1 v 3)=(p p s x 0 x 1 x 2 \in x 0 \wedge x 2 \in x 1(p p s x 0 x 1 x 2))$ by moura
then have f2: pps ( $R$ M2 s vs xs) (io-targets M1 (initial M1)) $s^{\prime} \in R$ M2 s vs xs
$\wedge s^{\prime} \in$ io-targets M1 (initial M1)
(pps ( $R$ M2 s vs xs) (io-targets M1 (initial M1)) $s^{\prime}$ )
using $\prec \neg s^{\prime} \notin \bigcup($ io-targets M1 (initial M1)' $R$ M2 s vs xs) 〉 by blast
then have $\exists$ ps. pps ( $R$ M2 $s$ vs xs) (io-targets M1 (initial M1)) $s^{\prime}=v s @ p s \wedge p s \neq[]$
$\wedge$ prefix ps xs $\wedge s \in$ io-targets M2 (initial M2) (vs @ ps)
by $\operatorname{simp}$
then show ?thesis
using f2 a1 by (metis (no-types))
qed
have vs @ $x s^{\prime} \in L M 1$ using $x s^{\prime}$-def by blast
then have io-targets M1 (initial M1) (vs@xs') $=\left\{s^{\prime}\right\}$
by (metis assms(2) io-targets-observable-singleton-ob singletonD $x s^{\prime}$-def)
moreover have io-targets M1 (initial M1) (vs') $=\left\{s^{\prime}\right\}$ using $s^{\prime}$-def by blast
moreover have io-targets M2 (initial M2) (vs @ $\left.x s^{\prime}\right)=\{s\}$
using state-component-2 xs'-def by blast
moreover have io-targets M2 (initial M2) ( $\left.v s^{\prime}\right)=\{s\}$
by (metis (mono-tags, lifting) RP.simps Un-iff insertI1 mem-Collect-eq vs'-def)
moreover have $x s^{\prime} \neq[]$

```
    using xs'-def by simp
    moreover have prefix xs' xs
    using x\mp@subsup{s}{}{\prime}-def by simp
    moreover have vs'}\in\mp@subsup{V}{}{\prime\prime
    using vs'-def by simp
    ultimately have Rep-Cov M2 M1 V'l vs xs
    by auto
    then show False
    using assms(14) by simp
qed
moreover have U(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}\mp@subsup{s}{}{\prime}\mathrm{ -def by simp
    moreover have finite (U (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 \bigwedgea.a\inR M2 s vs xs \Longrightarrow finite (io-targets M1 (initial M1) a)
    proof -
            fix a assume }a\inRM2 s vs x
            then have prefix a(vs@xs)
                by auto
            then have a\inL 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 (U (image (io-targets M1 (initial M1)) (insert vs' (R M2 s vs xs))))
                = Suc (card (U (image (io-targets M1 (initial M1)) (R M2 s vs xs))))
        by (metis (no-types) card-insert-disjoint)
    moreover have card (U (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 (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''\))
                        = Suc (card (U (image (io-targets M1 (initial M1)) (R M2 s vs xs))))
        by linarith
    then have card (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'I')))
                =Suc (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')}=\operatorname{Suc}(\operatorname{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 \((v s @ x s) \in L M 1 \cap L M 2\)
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 $P M($ initial $P M)$ vs $=\{(q 2, q 1)\}$
and path $P M(x s \| t r)(q 2, q 1)$
and length $x s=$ length $t r$
and $\neg$ Rep-Pre M2 M1 vs xs
and is-det-state-cover M2 V
and $V^{\prime \prime} \in \operatorname{Perm} V$ M1
and $\neg$ Rep-Cov M2 M1 $V^{\prime \prime}$ vs xs
shows card $(\bigcup($ image (io-targets M1 (initial M1 $))\left(R P M 2 s\right.$ vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)=\operatorname{card}\left(R P M 2 s\right.$ vs $\left.x s V^{\prime \prime}\right)$
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 $\left.V V^{\prime \prime}\right]$
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 }\times\mathrm{ 'out) list }=>\mathrm{ ('in }\times\mathrm{ ''out) list }=>\mathrm{ ' 'in list set
    => 'state1 set }=>\mathrm{ ('in, 'out) ATC set
    (''n }\times\mathrm{ 'out) list set }=>\mathrm{ nat
    where
    LB M2 M1 vs xs T S \Omega V'=
        (sum (\lambda s.card (RP M2 s vs xs V}\mp@subsup{V}{}{\prime\prime}))S
        +card ((D M1 T \Omega) -
            {B M1 xs' \Omega|x\mp@subsup{s}{}{\prime}\mp@subsup{s}{}{\prime}.\mp@subsup{s}{}{\prime}\inS\wedgex\mp@subsup{s}{}{\prime}\inRP M2 s' vs xs }\mp@subsup{V}{}{\prime\prime}}
```

```
lemma LB-count-helper-RP-disjoint-and-cards:
    assumes (vs@ xs)\inLM1\capL 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 }\mp@subsup{V}{}{\prime\prime}\inP\mathrm{ Perm V M1
    and s1 # s2
shows U(image (io-targets PM (initial PM)) (RP M2 s1 vs xs V'|}\mp@subsup{V}{}{\prime\prime}
            \cap\bigcup(image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'}\mp@subsup{V}{}{\prime\prime}))={
        card (U (image (io-targets PM (initial PM)) (RP M2 s1 vs xs V V'}))
            = card (U (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V'`')))
        card (U (image (io-targets PM (initial PM)) (RP M2 s2 vs xs V'\)))
            = card (U (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V'`)))
proof -
    have }\forall\mathrm{ io }\inRP M2 s1 vs xs V'". io-targets PM (initial PM) io
                        ={s1}\times io-targets M1 (initial M1) io
    proof
        fix io assume io \inRP M2 s1 vs xs V'/
        then have io-targets PM (initial PM) io
                = io-targets M2 (initial M2) io }\times\mathrm{ 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 \in RP M2 s1 vs xs V''> assms(3) RP-state-component-2[of io M2 s1 vs xs V V']
        by blast
        ultimately show io-targets PM (initial PM) io ={s1} }\times\mathrm{ io-targets M1 (initial M1) io
        by auto
    qed
```

```
then have \bigcup(image (io-targets PM (initial PM)) (RP M2 s1 vs xs V'))
    = \bigcup(image ( }\lambda\mathrm{ io . {s1} × io-targets M1 (initial M1) io) (RP M2 s1 vs xs V't})
    by simp
moreover have U(image ( }\\mathrm{ io . {s1} > io-targets M1 (initial M1) io) (RP M2 s1 vs xs V}\mp@subsup{V}{}{\prime\prime})
                ={s1}\times\bigcup(image (\lambda io . io-targets M1 (initial M1) io) (RP M2 s1 vs xs V'\prime))
    by blast
ultimately have image-split-1 :
    U(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 (U(image (io-targets PM (initial PM)) (RP M2 s1 vs xs V}\mp@subsup{V}{}{\prime\prime}))
                = 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 $\in R P$ M2 s2 vs xs $V^{\prime \prime}$. io-targets $P M$ (initial $\left.P M\right)$ io
$=\{s \mathscr{2}\} \times$ io-targets M1 (initial M1) io
proof
fix io assume io $\in R P$ M2 s2 vs xs $V^{\prime \prime}$
then have io-targets $P M$ (initial $P M)$ 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 $V^{\prime \prime}$ io s2] by simp
moreover have io-targets M2 (initial M2) io $=\{s \mathscr{2}\}$
using 〈io $\in R P$ M2 s2 vs xs $\left.\left.V^{\prime \prime}\right\rangle \operatorname{assms(3)~RP-state-component-2[of~io~M2~s2~vs~xs~} V^{\prime \prime}\right]$
by blast
ultimately show io-targets $P M($ initial $P M)$ io $=\{s \mathcal{Z}\} \times$ io-targets $M 1$ (initial M1) io
by auto
qed
then have $\bigcup($ image (io-targets $P M($ initial $P M))\left(R P\right.$ M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$=\bigcup\left(\right.$ image $\left(\lambda\right.$ io $.\{s 2\} \times$ io-targets M1 (initial M1) io) $\left(R P\right.$ M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
by $\operatorname{simp}$
moreover have $\bigcup\left(\right.$ image $\left(\lambda\right.$ io $.\{s 2\} \times$ io-targets M1 (initial M1) io) (RPM2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$=\{s \mathcal{2}\} \times \bigcup\left(\right.$ image $\left(\lambda\right.$ io . io-targets M1 (initial M1) io) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
by blast
ultimately have image-split-2 :
$\bigcup\left(\right.$ image (io-targets PM (initial PM)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$=\{s 2\} \times \bigcup\left(\right.$ image $($ io-targets M1 (initial M1) $)\left(R P M 2\right.$ s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ by simp
then show card $\left(\bigcup\right.$ (image (io-targets PM (initial PM)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ )
$=\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ )
by (metis (no-types) card-cartesian-product-singleton)
have $\bigcup($ image (io-targets $P M($ initial $P M))\left(R P\right.$ M2 s1 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$\cap \bigcup\left(\right.$ image $($ io-targets $P M($ initial $P M))\left(R P\right.$ M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$=\{s 1\} \times \bigcup\left(\right.$ image $($ io-targets M1 (initial M1) $)\left(R P M 2\right.$ s1 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$\cap\{s 2\} \times \bigcup\left(\right.$ image $($ io-targets $M 1$ (initial M1) $)\left(R P M 2\right.$ s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
using image-split-1 image-split-2 by blast
moreover have $\{s 1\} \times \bigcup$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.V^{\prime \prime}\right)$ )
$\cap\{s 2\} \times \bigcup\left(\right.$ image $\left(\right.$ io-targets M1 (initial M1)) $\left(R P\right.$ M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)=\{ \}$
using assms(9) by auto
ultimately show $\bigcup\left(\right.$ image $($ io-targets $P M($ initial $P M))\left(R P\right.$ M2 s1 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$\cap \bigcup\left(\right.$ image $($ io-targets $P M($ initial $P M))\left(R P M 2\right.$ s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)=\{ \}$
by presburger
qed
lemma LB-count-helper-RP-disjoint-card-M1 :
assumes $(v s @ x s) \in L M 1 \cap L M 2$
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 $V^{\prime \prime} \in \operatorname{Perm} V M 1$
and $s 1 \neq s 2$
shows card $\left(\bigcup\right.$ (image (io-targets PM (initial PM)) (RP M2 s1 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$\cup \bigcup\left(\right.$ image (io-targets PM (initial PM)) (RP M2 s2 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$
$=\operatorname{card}\left(\bigcup\left(\right.\right.$ image (io-targets M1 (initial M1)) $\left(R P\right.$ M2 s1 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$ $+\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ )
proof -
have finite $(\bigcup$ (image (io-targets $P M($ initial $P M))\left(R P M 2\right.$ s1 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$ using $R P$-io-targets-finite- $P M[O F \operatorname{assms}(1-8)]$ by simp
moreover have finite $\left(\bigcup\right.$ (image (io-targets PM (initial PM)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ )
using $R P$-io-targets-finite- $P M[O F \operatorname{assms}(1-8)]$ by simp
ultimately show ?thesis
using LB-count-helper-RP-disjoint-and-cards[OF assms]
by (metis (no-types) card-Un-disjoint)
qed
lemma LB-count-helper-RP-disjoint-M1-pair :
assumes (vs @ xs) $\in L M 1 \cap L M 2$
and observable M1
and observable M2
and well-formed M1
and well-formed M2
and productF M2 M1 FAIL PM
and io-targets $P M($ initial $P M)$ vs $=\{(q 2, q 1)\}$
and path $P M(x s \| t r)(q 2, q 1)$
and length $x s=$ length $t r$
and $\neg$ Rep-Pre M2 M1 vs xs
and is-det-state-cover M2 V
and $V^{\prime \prime} \in$ Perm $V$ M1
and $\neg$ Rep-Cov M2 M1 $V^{\prime \prime}$ vs xs
and Prereq M2 M1 vs xs TS $\Omega V^{\prime \prime}$
and $s 1 \neq s 2$
and $s 1 \in S$
and $s 2 \in S$
and applicable-set M1 $\Omega$
and completely-specified M1
shows card (RP M2 s1 vs xs $\left.V^{\prime \prime}\right)+\operatorname{card}\left(R P\right.$ M2 s2 vs $\left.x s V^{\prime \prime}\right)$
$=\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$
$+\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$
$\bigcup$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.V^{\prime \prime}\right)$ )
$\cap \bigcup\left(\right.$ image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$=\{ \}$
proof -
have $s 1 \in$ nodes M2
using assms $(14,16)$ unfolding Prereq.simps by blast
have s2 $\in$ nodes M2
using assms $(14,17)$ unfolding Prereq.simps by blast
have card ( $R P$ M2 s1 vs xs $V^{\prime \prime}$ )
$=\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\right)$
using $R P$-count-alt-def $[$ OF assms $(1-5)\langle s 1 \in \operatorname{nodes} M 2\rangle \operatorname{assms}(6-13)]$
by linarith
moreover have card (RP M2 s2 vs xs $V^{\prime \prime}$ )
$=\operatorname{card}\left(\bigcup\right.$ (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)$ )
using RP-count-alt-def[OF assms(1-5) <s2 $\in$ nodes M2〉 assms(6-13)]
by linarith
moreover show $\bigcup$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.\left.V^{\prime \prime}\right)\right)$
$\cap \bigcup\left(\right.$ image (io-targets M1 (initial M1)) $\left(\right.$ RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right)=\{ \}$
proof (rule ccontr)
assume $\bigcup$ (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs $\left.V^{\prime \prime}\right)$ )
$\cap \bigcup\left(\right.$ image (io-targets M1 (initial M1)) (RP M2 s2 vs xs $\left.\left.V^{\prime \prime}\right)\right) \neq\{ \}$
then obtain io1 io2 $t$ where shared-elem-def :
io1 $\in\left(R P\right.$ M2 s1 vs xs $\left.V^{\prime \prime}\right)$
$i o 2 \in\left(R P M 2\right.$ s2 vs xs $\left.V^{\prime \prime}\right)$
$t \in$ io-targets M1 (initial M1) io1
$t \in$ io-targets M1 (initial M1) io2
by blast

```
    have dist-prop: (\forall s1 \inS.\forall s2 \inS.s1 # s2
        \longrightarrow ( \forall ~ i o 1 ~ \in R P ~ M 2 ~ s 1 ~ v s ~ x s ~ V ' ' .
            \forall io2 \in RP M2 s2 vs xs V'".
                        B M1 io1 \Omega\not= 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 \not={}
    then have io-targets M1 (initial M1) io1 }\not={}\mathrm{ io-targets M1 (initial M1) io2 }\not={
        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 <io-targets M1 (initial M1) io1 \cap io-targets M1 (initial M1) io2 f= {}>
        by auto
    then have B M1 io1 \Omega= B M1 io2 \Omega
        using <io-targets M1 (initial M1) io1 = {s1}>\langleio-targets M1 (initial M1) io2 = {s2}>
        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 }\mp@subsup{V}{}{\prime\prime})+\operatorname{card (RP M2 s2 vs xs V V')
    = card (U (image (io-targets M1 (initial M1)) (RP M2 s1 vs xs V'\)))
        +card (U (image (io-targets M1 (initial M1)) (RP M2 s2 vs xs V'}\mp@subsup{V}{}{\prime\prime}))
    by linarith
qed
```

lemma LB-count-helper-RP-card-union :
assumes observable M2
and $\quad s 1 \neq s 2$
shows RP M2 s1 vs xs $V^{\prime \prime} \cap R P$ M2 s2 vs xs $V^{\prime \prime}=\{ \}$
proof (rule ccontr)
assume $R P$ M2 s1 vs xs $V^{\prime \prime} \cap R P$ M2 s2 vs $x s V^{\prime \prime} \neq\{ \}$
then obtain io where $i o \in R P$ M2 s1 vs xs $V^{\prime \prime} \wedge i o \in R P M 2$ s2 vs xs $V^{\prime \prime}$
by blast
then have s1 $\in$ io-targets M2 (initial M2) io s2 $\in$ io-targets M2 (initial M2) io
by auto
then have $s 1=s 2$
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 \(\forall q \in\left(\bigcup\right.\) (image \(\left(\lambda s . \bigcup\right.\) (image (io-targets M1 (initial M1)) (RP M2 s vs xs \(\left.\left.V^{\prime \prime}\right)\right)\) ) S)).
            \(f q \in\) nodes M1
        inj-on \(f\left(\bigcup\right.\) (image \(\left(\lambda s . \bigcup\left(\right.\right.\) image (io-targets M1 (initial M1)) (RPM2 s vs xs \(\left.\left.\left.\left.\left.V^{\prime \prime}\right)\right)\right) S\right)\right)\)
proof -
    let \(? f=\)
        \(\lambda q\). if \(\left(q \in\left(\bigcup\right.\right.\) (image \(\left(\lambda s . \bigcup\right.\) (image (io-targets M1 (initial M1)) (RPM2 s vs xs \(\left.\left.\left.\left.\left.V^{\prime \prime}\right)\right)\right) S\right)\right)\)
            then \(q\)
            else (initial M1)
    have \(\left(\bigcup\left(\right.\right.\) image \(\left(\lambda s . \bigcup(\right.\) image (io-targets M1 (initial M1) \()\left(R P M 2 s\right.\) vs \(\left.\left.\left.\left.x s V^{\prime \prime}\right)\right)\right) S\right) \subseteq\) nodes M1
        by blast
    then have \(\forall q \in\left(\bigcup\right.\) (image \(\left(\lambda s . \bigcup\right.\) (image (io-targets M1 (initial M1)) (RP M2 s vs xs \(\left.\left.\left.\left.V^{\prime \prime}\right)\right)\right) S\right)\) ).
            ?f \(q \in\) nodes M1
        by (metis Un-iff sup.order-iff)
    moreover have inj-on ?f \((\bigcup\) (image ( \(\lambda s . \bigcup\) (image (io-targets M1 (initial M1))
                                    ( \(R P\) M2 \(s\) vs \(\left.\left.x s V^{\prime \prime}\right)\right)\) ) \(S\) )
    proof
        fix \(x\) assume \(x \in\left(\bigcup\right.\) (image \(\left(\lambda s . \bigcup\right.\) (image (io-targets M1 (initial M1)) (RP M2 s vs xs \(\left.\left.V^{\prime \prime}\right)\right)\) ) S))
        then have ?f \(x=x\)
            by presburger
        fix \(y\) assume \(y \in\left(\bigcup\right.\) (image ( \(\lambda s . \bigcup\) (image (io-targets M1 (initial M1)) (RP M2 s vs xs \(\left.\left.V^{\prime \prime}\right)\right)\) ) S) )
        then have ?f \(y=y\)
        by presburger
        assume ?f \(x=\) ?f \(y\)
        then show \(x=y\) using «? \(x=x\) 〉〈?f \(y=y\) 〉
        by presburger
    qed
    ultimately show ?thesis
        using that by presburger
qed
abbreviation (input) UNION :: 'a set \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right.\) set \() \Rightarrow{ }^{\prime} b\) set
    where UNION \(A f \equiv \bigcup\left(f{ }^{\prime} A\right)\)
lemma \(L B\)-count-helper-RP-card-union-sum :
    assumes \((v s @ x s) \in L M 2 \cap L M 1\)
    and OFSM M1
    and OFSM M2
    and asc-fault-domain M2 M1 m
    and test-tools M2 M1 FAIL PM V \(\Omega\)
    and \(\quad V^{\prime \prime} \in \operatorname{Perm} V M 1\)
    and Prereq M2 M1 vs xs TS \(\Omega V^{\prime \prime}\)
    and \(\quad \neg\) Rep-Pre M2 M1 vs xs
    and \(\neg\) Rep-Cov M2 M1 \(V^{\prime \prime}\) vs xs
shows sum \(\left(\lambda s\right.\). card ( \(R P\) M2 \(s\) vs xs \(\left.\left.V^{\prime \prime}\right)\right) S\)
        \(=\operatorname{sum}\left(\lambda s . \operatorname{card}\left(\bigcup\left(\right.\right.\right.\) image \(\left(\right.\) io-targets M1 (initial M1)) \(\left(R P M 2 s\right.\) vs xs \(\left.\left.\left.\left.V^{\prime \prime}\right)\right)\right)\right) S\)
using assms proof -
    have finite (nodes M2)
        using assms(3) by auto
    moreover have \(S \subseteq\) nodes M2
        using \(\operatorname{assms}(7)\) by simp
    ultimately have finite \(S\)
        using infinite-super by blast
    then have \(\operatorname{sum}\left(\lambda s . \operatorname{card}\left(R P\right.\right.\) M2 \(s\) vs \(\left.\left.x s V^{\prime \prime}\right)\right) S\)
                \(=\operatorname{sum}\left(\lambda s\right.\).card \(\left(\bigcup\right.\) (image (io-targets M1 (initial M1)) \(\left(R P\right.\) M2 s vs xs \(\left.\left.\left.\left.V^{\prime \prime}\right)\right)\right)\right) S\)
    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.prems(7) by simp
    then have s\in nodes M2
        by simp
    have Prereq M2 M1 vs xs TS\Omega V'
    using <Prereq M2 M1 vs xs T (insert s S) \Omega V'>}>\mathrm{ by simp
    then have ( \sums\inS.card (RP M2 s vs xs V}\mp@subsup{V}{}{\prime\prime})
            =(\sums\inS.card (\bigcupa\inRP M2 s vs xs V''. io-targets M1 (initial M1) a))
        using insert.IH[OF insert.prems (1-6)-assms(8,9)] by metis
    moreover have (\sum s'\in(insert s S). card (RP M2 s' vs xs V V'))
                =(\sum\mp@subsup{s}{}{\prime}\inS.card (RP M2 s' vs xs }\mp@subsup{V}{}{\prime\prime}))+\operatorname{card}(RP M2 s vs xs V V'), 
        by (simp add: add.commute insert.hyps(1) insert.hyps(2))
    ultimately have S-prop:( \sum s'\in(insert s S).card (RP M2 s' vs xs V''})\mathrm{ )
                    = (\sums\inS. card (\bigcup \ a\inRP M2 s vs xs V V'. io-targets M1 (initial M1) a))
        by presburger
    have vs@xs\inL M1 \capL M2
    using insert.prems(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.prems(1)
            test-tools-props(1)[OF insert.prems(5,4)] OFSM-props(2,1)[OF insert.prems(3)]
                                OFSM-props(2,1)[OF insert.prems(2)]]
    by blast
    have card (RP M2 s vs xs V')
        = card (U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'')))
        using OFSM-props(2,1)[OF insert.prems(3)] OFSM-props(2,1)[OF insert.prems(2)]
            RP-count-alt-def[OF <vs@xs \inL M1 \capL M2`----
                            <s\innodes M2` test-tools-props(1)[OF insert.prems(5,4)]
                                suffix-path insert.prems(8)
                            test-tools-props(2)[OF insert.prems(5,4)] assms(6) insert.prems(9)]
        by linarith
    show (\sums\ininsert s S.card (RP M2 s vs xs V}\mp@subsup{V}{}{\prime\prime}))
                (\sums\ininsert s S.card (UNION (RP M2 s vs xs V'') (io-targets M1 (initial M1))))
    proof -
    have (\sumc\ininsert s S.card (UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1))))
                = card (UNION (RP M2 s vs xs V')}\mp@subsup{V}{}{\prime\prime})(\mathrm{ io-targets M1 (initial M1)))
                +(\sumc\inS.card (UNION (RP M2 c vs xs V'') (io-targets M1 (initial M1))))
            by (meson insert.hyps(1) insert.hyps(2) sum.insert)
        then show ?thesis
            using «(\sum s'\ininsert s S.card (RP M2 s' vs xs }\mp@subsup{V}{}{\prime\prime})
                    =(\sums\inS.card (\bigcupa\inRP M2 s vs xs V V'. io-targets M1 (initial M1) a))
                        + card (RP M2 s vs xs V}\mp@subsup{V}{}{\prime\prime})\mathrm{ )
                    <card (RP M2 s vs xs V V')
                        = card (UNION (RP M2 s vs xs V') (io-targets M1 (initial M1)))>
        by presburger
    qed
qed
then show ?thesis
    using assms by blast
qed
```

```
lemma finite-insert-card :
    assumes finite (\SS)
    and finite S
    and}\quadS\cap(\bigcupSS)={
shows card (\bigcup (insert SSS)) = card (\bigcupSS) + 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) \inL 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}\quad\mp@subsup{V}{}{\prime\prime}\in\operatorname{Perm V M1
    and Prereq M2 M1 vs xs TS \Omega V'
    and }\neg\mathrm{ Rep-Pre M2 M1 vs xs
    and \neg Rep-Cov M2 M1 V' vs xs
shows sum ( }\lambdas.\operatorname{card}(RPM2 M vs xs V''))
        = card (U (image (\lambda s.\bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V''))) S))
using assms proof -
    have finite (nodes M2)
        using assms(3) by auto
    moreover have S\subseteq nodes M2
        using assms(7) by simp
    ultimately have finite S
        using infinite-super by blast
    then show sum ( }\lambdas.\operatorname{card}(RPM2 s vs xs V'')) 
                = card (U (image (\lambda s.U (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'')))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.prems(7) by simp
        then have s\in nodes M2
        by simp
        have Prereq M2 M1 vs xs TS\Omega V'
        using <Prereq M2 M1 vs xs T (insert s S) \Omega V ''> by simp
        then have applied-IH : (\sums\inS.card (RP M2 s vs xs V'\}
                                    = card (\bigcups\inS. \bigcupa\inRP M2 s vs xs V''. io-targets M1 (initial M1) a)
        using insert.IH[OF insert.prems(1-6) - insert.prems(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.prems(1) test-tools-props(1)[OF insert.prems(5,4)]
                        OFSM-props(2,1)[OF insert.prems(3)] OFSM-props(2,1)[OF insert.prems(2)]]
        by blast
        have s\in insert s S
        by simp
        have vs@xs \inL M1 \capL M2
        using insert.prems(1) by simp
        have }\forall\mp@subsup{s}{}{\prime}\inS.(\bigcupa\inRP M2 s vs xs V V''. io-targets M1 (initial M1) a)
                        \cap(\bigcupa\inRP M2 s' vs xs V''. io-targets M1 (initial M1) a)={}
        proof
            fix }\mp@subsup{s}{}{\prime}\mathrm{ assume }\mp@subsup{s}{}{\prime}\in
```

```
have s\not= s'
    using insert.hyps(2)<\mp@subsup{s}{}{\prime}\inS\rangle by blast
    have }\mp@subsup{s}{}{\prime}\in\mathrm{ insert }s\mathrm{ S
    using < s'\inS` by simp
```

    show ( \(\bigcup a \in R P\) M2 s vs xs \(V^{\prime \prime}\). io-targets M1 (initial M1) a)
        \(\cap\left(\bigcup a \in R P\right.\) M2 s' vs xs \(V^{\prime \prime}\). io-targets M1 (initial M1) a) \(=\{ \}\)
    using \(\operatorname{OFSM}\)-props(2,1)[OF assms(3)] OFSM-props(2,1,3)[OF assms(2)]
        LB-count-helper-RP-disjoint-M1-pair (2)
            \([\) OF \(\langle v s @ x s \in L M 1 \cap L\) M2 - - - test-tools-props(1)[OF insert.prems \((5,4)]\)
            suffix-path insert.prems(8) test-tools-props(2)[OF insert.prems (5,4)]
            insert.prems \((6,9,7)\left\langle s \neq s^{\prime}\right\rangle\langle s \in\) insert \(s S\rangle\left\langle s^{\prime} \in\right.\) insert \(\left.s S\right\rangle\)
            test-tools-props(4)[OF insert.prems(5,4)]]
    by linarith
    qed
then have disj-insert : ( $\bigcup s \in S . \bigcup a \in R P$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
$\cap\left(\bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a) $=\{ \}$
by blast
have finite-S : finite $\left(\bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
using $R P$-io-targets-finite-M1[OF insert.prems(1)]
by (meson RP-io-targets-finite-M1 «vs @ xs $\in L M 1 \cap L M 2\rangle \operatorname{assms}(2) \operatorname{assms}(5)$ insert.prems(6))
have finite-s : finite $\left(\bigcup s \in S . \bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
by (meson RP-io-targets-finite-M1〈vs @ xs $\in L M 1 \cap L M 2\rangle \operatorname{assms}(2) \operatorname{assms}(5)$
finite-UN-I insert.hyps(1) insert.prems(6))
have card $\left(\bigcup s \in\right.$ insert $s S . \bigcup a \in R P$ M2 s vs $x s V^{\prime \prime}$. io-targets M1 (initial M1) a)
$=$ card $\left(\bigcup s \in S . \bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
+ card $\left(\bigcup a \in R P\right.$ M2 $s$ vs $x s V^{\prime \prime}$. io-targets M1 (initial M1) a)
proof -
have f1: insert (UNION (RP M2 s vs xs $\left.V^{\prime \prime}\right)$ (io-targets M1 (initial M1)))
(( $\lambda c$. UNION (RP M2 c vs xs $\left.V^{\prime \prime}\right)($ io-targets M1 (initial M1) )) ' $S$ )
$=\left(\lambda c\right.$. UNION $\left(R P M 2 c\right.$ vs xs $\left.V^{\prime \prime}\right)($ io-targets M1 (initial M1) $\left.)\right)$ ' insert $s ~ S$
by blast
have $\forall c . c \in S \longrightarrow$ UNION (RP M2 s vs xs $\left.V^{\prime \prime}\right)$ (io-targets M1 (initial M1))
$\cap \operatorname{UNION}\left(R P M 2 c\right.$ vs xs $\left.V^{\prime \prime}\right)($ io-targets M1 (initial M1 $\left.)\right)=\{ \}$
by (meson $\left\langle\forall s^{\prime} \in S .\left(\bigcup a \in R P\right.\right.$ M2 $s$ vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
$\cap\left(\bigcup a \in R P\right.$ M2 $s^{\prime}$ vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) $\left.\left.\left.a\right)=\{ \}\right\rangle\right)$
then have UNION (RP M2 s vs xs $\left.V^{\prime \prime}\right)$ (io-targets M1 (initial M1))
$\cap\left(\bigcup c \in S\right.$. UNION $\left(R P\right.$ M2 $c$ vs $\left.x s V^{\prime \prime}\right)($ io-targets M1 (initial M1 $\left.\left.)\right)\right)=\{ \}$
by blast
then show ?thesis
using $f 1$ by (metis finite-S finite-insert-card finite-s)
qed
have card ( $R P$ M2 $s$ vs $x s V^{\prime \prime}$ )
$=$ card $\left(\bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
using assms(2) assms(3)
RP-count-alt-def[OF $\langle v s @ x s \in L M 1 \cap L M 2\rangle---\langle s \in$ nodes M2〉
test-tools-props(1)[OF insert.prems $(5,4)]$ suffix-path
insert.prems(8) test-tools-props(2)[OF insert.prems (5,4)]
insert.prems $(6,9)$ ]
by metis
show ?case
proof -
have ( $\sum c \in$ insert $s$ S. card ( $R P$ M2 c vs xs $\left.V^{\prime \prime}\right)$ )
$=\operatorname{card}\left(R P\right.$ M2 s vs xs $\left.V^{\prime \prime}\right)+\left(\sum c \in S . \operatorname{card}\left(R P\right.\right.$ M2 c vs xs $\left.\left.V^{\prime \prime}\right)\right)$
by (meson insert.hyps(1) insert.hyps(2) sum.insert)
then show?thesis
using <card (RP M2 s vs xs $V^{\prime \prime}$ )
$=$ card $\left(\bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a) >
<card $(\bigcup\} \in$ insert $s S . \bigcup a \in R P$ M2 $s$ vs $x s V^{\prime \prime}$. io-targets M1 (initial M1) a)
$=$ card $\left(\bigcup s \in S . \bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a)
+ card $\left(\bigcup a \in R P\right.$ M2 s vs xs $V^{\prime \prime}$. io-targets M1 (initial M1) a) > applied-IH
by presburger
qed
qed
qed

```
lemma LB-count-helper-LB1 :
    assumes (vs@ xs) \(\in L\) M2 \(\cap L M 1\)
    and OFSM M1
    and OFSM M2
    and asc-fault-domain M2 M1 m
    and test-tools M2 M1 FAIL PMV \(\Omega\)
    and \(\quad V^{\prime \prime} \in \operatorname{Perm} V M 1\)
    and Prereq M2 M1 vs xs TS \(\Omega V^{\prime \prime}\)
    and \(\neg\) Rep-Pre M2 M1 vs xs
    and \(\neg\) Rep-Cov M2 M1 \(V^{\prime \prime}\) vs xs
shows \(\left(\operatorname{sum}\left(\lambda s . \operatorname{card}\left(R P\right.\right.\right.\) M2 s vs xs \(\left.\left.\left.V^{\prime \prime}\right)\right) S\right) \leq \operatorname{card}(\) nodes M1)
proof -
    have \(\left(\bigcup s \in S\right.\). UNION \(\left(R P\right.\) M2 s vs xs \(\left.V^{\prime \prime}\right)(\) io-targets M1 (initial M1 \(\left.\left.)\right)\right) \subseteq\) 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 \(\left.V^{\prime \prime}\right)\) (io-targets M1 (initial M1)))
                \(\leq \operatorname{card}(\) nodes M1)
        by (meson card-mono)
    moreover have ( \(\sum s \in S\). card ( \(R P\) M2 \(s\) vs \(\left.x s V^{\prime \prime}\right)\) )
                \(=\) card \(\left(\bigcup s \in S\right.\). UNION (RP M2 s vs xs \(\left.V^{\prime \prime}\right)\) (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 $\quad R S \in(D M T \Omega)$
obtains $q$
where $q \in$ nodes $M \wedge R S=I O$-set $M q \Omega$
proof -
have $R S \in$ image ( $\lambda$ io. $B M$ io $\Omega)\left(L S_{i n} M(\right.$ initial $\left.M) T\right)$ using assms by simp
then obtain io where $R S=B M$ io $\Omega$ io $\in L S_{i n} M($ 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 $\Omega=I O$-set $M q \Omega$ by simp
then have $R S=I O$-set $M q \Omega$ using $\langle R S=B M$ io $\Omega\rangle$ by $\operatorname{simp}$
moreover have $q \in$ nodes $M$ using $\langle\{q\}=$ io-targets $M($ initial $M)$ io>
by (metis FSM.nodes.initial insertI1 io-targets-nodes)
ultimately show ?thesis
using that by simp
qed

```
lemma LB-count-helper-LB2 :
    assumes observable M1
```



```
shows q\not\in(\bigcup (image (\lambdas.\bigcup (image (io-targets M1 (initial M1)) (RP M2 s vs xs V'\))) S))
proof
    assume q\in(\bigcups\inS.UNION (RP M2 s vs xs V'))(io-targets M1 (initial M1)))
    then obtain s' where s'\inS q\in(\bigcup (image (io-targets M1 (initial M1)) (RP M2 s' vs xs V'\}\mp@subsup{|}{}{\prime\prime})
        by blast
```



```
        by blast
    then have {q} = io-targets M1 (initial M1) xs'
        by (metis assms(1) observable-io-target-is-singleton)
    then have BM1 xs'\Omega=\bigcup(image ( }\lambda\mathrm{ s. IO-set M1 s }\Omega){q}
        by simp
    then have B M1 xs' \Omega=IO-set M1 q \Omega
        by simp
    moreover have B M1 xs' \Omega\in{B M1 xs' \Omega|x\mp@subsup{s}{}{\prime}\mp@subsup{s}{}{\prime}.\mp@subsup{s}{}{\prime}\inS\wedgex\mp@subsup{s}{}{\prime}\inRP M2 s' vs xs \mp@subsup{V}{}{\prime\prime}}
        using \langles' \inS\rangle\langlex\mp@subsup{s}{}{\prime}\inRP M2 s' vs xs }\mp@subsup{V}{}{\prime\prime}\rangle\mathrm{ by blast
    ultimately have IO-set M1 q \Omega \in{B M1 xs'\Omega|x\mp@subsup{s}{}{\prime}}\mp@subsup{s}{}{\prime}.\mp@subsup{s}{}{\prime}\inS\wedgex\mp@subsup{s}{}{\prime}\inRP M2 s' vs xs V V'} 
        by blast
    moreover have IO-set M1 q \Omega\not\in{B M1 xs'\Omega|x\mp@subsup{s}{}{\prime}\mp@subsup{s}{}{\prime}.\mp@subsup{s}{}{\prime}\inS\wedgex\mp@subsup{s}{}{\prime}\inRP M2 s' vs xs V't}
        using assms(2) by blast
    ultimately show False
        by simp
qed
```


### 4.7 Validity of the result of LB constituting a lower bound

lemma $L B$-count :
assumes (vs@xs) $\in L M 1$
and OFSM M1
and OFSM M2
and asc-fault-domain M2 M1 m
and test-tools M2 M1 FAIL PMV $\Omega$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
and Prereq M2 M1 vs xs TS $\Omega V^{\prime \prime}$
and $\quad \neg$ Rep-Pre M2 M1 vs xs
and $\neg$ Rep-Cov M2 M1 $V^{\prime \prime}$ vs xs
shows $L B$ M2 M1 vs xs $T S \Omega V^{\prime \prime} \leq|M 1|$
proof -
let $? D=D \mathrm{M} 1 \mathrm{~T} \Omega$
let ? $B=\left\{B M 1 x s^{\prime} \Omega \mid x s^{\prime} s^{\prime} . s^{\prime} \in S \wedge x s^{\prime} \in R P M 2 s^{\prime}\right.$ vs $\left.x s V^{\prime \prime}\right\}$
let $? D B=? D-$ ? $B$
let $? R P=\bigcup s \in S . \bigcup a \in R P$ M2 s vs xs $V^{\prime \prime}$. io-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 $\Omega$ ] unfolding Prereq.simps by linarith
then have finite ? $D B$
by $\operatorname{simp}$

- 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.

```
proof -
    fix }D\mp@subsup{B}{}{\prime}\mathrm{ assume }D\mp@subsup{B}{}{\prime}\subseteq?D
    have finite D\mp@subsup{B}{}{\prime}
    proof (rule ccontr)
        assume infinite DB'
```

have states- $: ~ \bigwedge D B^{\prime} . D B^{\prime} \subseteq ? D B \Longrightarrow \exists f$. inj-on $f D B^{\prime}$
$\wedge$ image f $D B^{\prime} \subseteq($ nodes $M 1)-? R P$
$\wedge\left(\forall R S \in D B^{\prime} . I O\right.$-set M1 $\left.(f R S) \Omega=R S\right)$

```
    have infinite ?DB
    using infinite-super [OF \langleD\mp@subsup{B}{}{\prime}\subseteq?DB\rangle\langleinfinite D\mp@subsup{B}{}{\prime}>}]\mathrm{ by simp
    then show False
    using <finite ?DB> by simp
qed
then show \exists f.inj-on fD\mp@subsup{B}{}{\prime}\wedge image f D\mp@subsup{B}{}{\prime}\subseteq(\mathrm{ nodes M1) - ?RP}
                        \wedge(\forallRS\inDB'.IO-set M1 (fRS) \Omega=RS)
using assms }\langleD\mp@subsup{B}{}{\prime}\subseteq?,DB\rangle\mathrm{ proof (induction DB')
    case empty
    show ?case by simp
next
    case (insert RS DB')
    have }D\mp@subsup{B}{}{\prime}\subseteq??D
        using insert.prems(10) by blast
    obtain f}\mp@subsup{f}{}{\prime}\mathrm{ where inj-on f}\mp@subsup{f}{}{\prime}D\mp@subsup{B}{}{\prime
                    image f}\mp@subsup{f}{}{\prime}D\mp@subsup{B}{}{\prime}\subseteq(\mathrm{ nodes M1 ) - ?RP
                            \forallRS\inD\mp@subsup{B}{}{\prime}.IO-set M1 ( f' RS) \Omega=RS
        using insert.IH[OF insert.prems(1-9)\langleD\mp@subsup{B}{}{\prime}\subseteq?DB\rangle]
        by blast
    have RS\inD M1 T \Omega
        using insert.prems(10) by blast
    obtain q}\mathrm{ where qG nodes M1 RS=IO-set M1 q ת
        using insert.prems(2) LB-count-helper-D-states[OF - <RS \inD M1 T \Omega>]
        by blast
    then have IO-set M1 q \Omega\in? DB
    using insert.prems(10) by blast
have q#?RP
    using insert.prems(2) LB-count-helper-LB2[OF - <IO-set M1 q \Omega \in?DB`]
    by blast
let ?f = f
have inj-on ?f (insert RS DB')
proof
    have ?f RS & ?f ' (DB' - {RS})
    proof
        assume ?f RS \in ?f ' (DB' - {RS})
        then have q\in?f '(DB' - {RS}) by auto
        have RS\inD\mp@subsup{B}{}{\prime}
        proof -
            have }\forallPcf.\existsPa.((c::'c)\not\in\mp@subsup{f}{}{\prime}P\vee(Pa::('a\times'b) list set)\inP
                                    \wedge(c\not\inf'P\veefPa=c)
                by auto
            moreover
            { assume q}\not\in\mp@subsup{f}{}{\prime}`D\mp@subsup{B}{}{\prime
                moreover
                { assume q\not\in\mp@subsup{f}{}{\prime}(RS:=q)`D\mp@subsup{B}{}{\prime}
                    then have ?thesis
                    using <q \in 生(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 \Omega\rangle\langle\forall RS\inDB'.IO-set M1 (f' RS) \Omega=RS`)
        qed
        then show False using insert.hyps(2) by simp
    qed
    then show inj-on ?f D\mp@subsup{B}{}{\prime}}\wedge\mathrm{ ?f RS & ?f '(DB' - {RS})
        using <inj-on f' DB'` inj-on-fun-updI by fastforce
    qed
    moreover have image ?f (insert RS DB')\subseteq(nodes M1) - ?RP
    proof -
    have image ?f {RS}={q} by simp
    then have image ?f {RS}\subseteq(nodes M1) - ?RP
        using <q\in nodes M1 <q & ?RP` by auto
```

```
            moreover have image ?f (insert RS DB')= image ?f {RS}\cup image ?f D\mp@subsup{B}{}{\prime}
                by auto
            ultimately show ?thesis
                by (metis (no-types,lifting)<image f'DB'\subseteq(nodes M1) - ?RP> fun-upd-other image-cong
                    image-insert insert.hyps(2) insert-subset)
    qed
    moreover have }\forallRS\in(\mathrm{ insert RS DB'). IO-set M1 (?f RS) }\Omega=R
        using \langleRS = IO-set M1 q \Omega\rangle\langle\forallRS\inD\mp@subsup{B}{}{\prime}.IO-set M1 ( f' RS) \Omega=RS` by auto
    ultimately show ?case
        by blast
    qed
qed
have ?}DB\subseteq?D
    by simp
obtain f}\mathrm{ where inj-on f?DB image f ? DB }\subseteq(\mathrm{ nodes M1) - ?RP
    using states-f[OF <?DB \subseteq?DB`] by blast
have finite (nodes M1 - ?RP)
    using <finite (nodes M1)> by simp
have card ?DB \leqcard (nodes M1 - ?RP)
    using card-inj-on-le[OF <inj-on f ?DB`<image f ?DB \subseteq(nodes M1) - ?RP〉
                            <finite (nodes M1 - ?RP)〉]
    by assumption
have ?RP\subseteq nodes M1
    by blast
then have card (nodes M1 - ?RP) = card (nodes M1) - card ?RP
    by (meson<finite (nodes M1)〉 card-Diff-subset infinite-subset)
then have card? DB\leqcard (nodes M1) - card ?RP
    using <card ?DB \leq card (nodes M1 - ?RP)` by linarith
    have vs @ xs \inL M2 \capL M1
    using assms(7) by simp
    have (sum ( }\lambda\mathrm{ s.card (RP M2 s vs xs V'')) S)= card ?RP
    using LB-count-helper-RP-disjoint-M1-union[OF<vs @ xs \inL M2 \capL M1`assms(2-9)] by simp
    moreover have card ?RP \leq card (nodes M1)
    using card-mono[OF<finite (nodes M1)〉<?RP \subseteq nodes M1>] by assumption
ultimately show ?thesis
    unfolding LB.simps using <card ?DB \leq card (nodes M1) - card ?RP>
    by linarith
qed
```

lemma contradiction－via－LB：
assumes（vs＠$x s$ ）$\in L$ M1
and OFSM M1
and OFSM M2
and asc－fault－domain M2 M1 m
and test－tools M2 M1 FAIL PMV $\Omega$
and $\quad V^{\prime \prime} \in \operatorname{Perm} V M 1$
and Prereq M2 M1 vs xs TS $\Omega V^{\prime \prime}$
and $\quad \neg$ Rep－Pre M2 M1 vs xs
and $\neg$ Rep－Cov M2 M1 $V^{\prime \prime}$ vs xs
and LBM2M1 vs xs $T S \Omega V^{\prime \prime}>m$
shows False
proof－
have $L B$ M2 M1 vs xs $T S \Omega V^{\prime \prime} \leq$ card（nodes M1）
using $L B$－count $[O F \operatorname{assms}(1-9)]$ 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 $A S C-L B$
begin

## 5 Test suite generated by the Adaptive State Counting Algorithm

## 5．1 Maximum length contained prefix

fun $m c p::$＇$a$ list $\Rightarrow{ }^{\prime} a$ list set $\Rightarrow{ }^{\prime}$ a list $\Rightarrow$ bool where
mcp $z W p=($ prefix $p z \wedge p \in W \wedge$
$\left(\forall p^{\prime} .\left(\right.\right.$ prefix $\left.p^{\prime} z \wedge p^{\prime} \in W\right) \longrightarrow$ length $p^{\prime} \leq$ length $\left.\left.p\right)\right)$
lemma mcp－ex
assumes []$\in W$
and finite $W$
obtains $p$
where mcp $z W p$
proof－
let $? P=\{p \cdot$ prefix $p z \wedge p \in W\}$
let ？ $\max P=\arg -m a x$ length $(\lambda p . p \in ? P)$
have finite $\{p$ ．prefix $p z\}$
proof－
have $\{p$. prefix $p z\} \subseteq$ image $(\lambda i$ ．take $i z)(\operatorname{set}[0 . .<$ Suc（length $z)])$
proof
fix $p$ assume $p \in\{p$ ．prefix $p z\}$
then obtain $i$ where $i \leq$ length $z \wedge p=$ take $i z$
by（metis append－eq－conv－conj mem－Collect－eq prefix－def prefix－length－le）
then have $i<S u c$（length $z$ ）$\wedge p=$ take $i z$
by $\operatorname{simp}$
then show $p \in \operatorname{image}(\lambda i$ ．take $i z)(\operatorname{set}[0 . .<$ Suc（length $z)])$
using atLeast－upt by blast
qed
then show ？thesis
using finite－surj by blast
qed
then have finite？P
by $\operatorname{simp}$
have $? P \neq\{ \}$
using Nil－prefix assms（1）by blast
have $\exists \max P \in ? P . \forall p \in ? P$. length $p \leq$ length $\max P$
proof（rule ccontr）
assume $\neg(\exists \max P \in ? P . \forall p \in ? P$ ．length $p \leq$ length $\max P)$
then have $\forall p \in ? P . \exists p^{\prime} \in ? P$. length $p<$ length $p^{\prime}$ by（meson not－less）
then have $\forall l \in($ image length ？$P) . \exists l^{\prime} \in($ image length ？$P) . l<l^{\prime}$ by auto
then have infinite（image length ？P）
by（metis（no－types，lifting）〈？P $\neq\{ \}>$ image－is－empty infinite－growing）
then have infinite？P
by blast
then show False
using〈finite？？〉 by simp
qed
then obtain $\max P$ where $\max P \in ? P \forall p \in ? P$ ．length $p \leq$ length max $P$ by blast
then have $m c p z W \max P$
unfolding mcp．simps by blast

```
    then show ?thesis
    using that by auto
qed
```

```
lemma mcp-unique :
    assumes mcp \(z W p\)
    and \(\quad m c p z W p^{\prime}\)
shows \(p=p^{\prime}\)
proof -
    have length \(p^{\prime} \leq\) length \(p\)
        using \(\operatorname{assms}(1) \operatorname{assms}(2)\) by auto
    moreover have length \(p \leq\) length \(p^{\prime}\)
        using assms(1) assms(2) by auto
    ultimately have length \(p^{\prime}=\) length \(p\)
        by \(\operatorname{simp}\)
```

    moreover have prefix \(p z\)
        using assms(1) by auto
    moreover have prefix \(p^{\prime} z\)
        using assms(2) by auto
    ultimately show ?thesis
        by (metis append-eq-conv-conj prefixE)
    qed
fun $m c p^{\prime}$ :: ' $a$ list $\Rightarrow$ 'a list set $\Rightarrow{ }^{\prime}$ 'a list where
$m c p^{\prime} z W=($ THE $p . m c p z W p)$
lemma $m c p^{\prime}$-intro :
assumes $m c p z W p$
shows $m c p^{\prime} z W=p$
using assms mcp-unique by (metis mcp'.elims theI-unique)
lemma mcp-prefix-of-suffix :
assumes mcp (vs@xs) Vvs
and prefix xs' ${ }^{\prime}$ ss
shows mcp (vs@xs') V vs
proof (rule ccontr)
assume $\neg m c p\left(v s @ x s^{\prime}\right) V v s$
then have $\neg\left(\right.$ prefix vs $\left(v s @ x s^{\prime}\right) \wedge v s \in V \wedge$
$\left(\forall p^{\prime} .\left(\right.\right.$ prefix $\left.p^{\prime}\left(v s @ x s^{\prime}\right) \wedge p^{\prime} \in V\right) \longrightarrow$ length $p^{\prime} \leq$ length vs $\left.)\right)$
by auto
then have $\neg\left(\forall p^{\prime} .\left(\right.\right.$ prefix $\left.p^{\prime}\left(v s @ x s^{\prime}\right) \wedge p^{\prime} \in V\right) \longrightarrow$ length $p^{\prime} \leq$ length $\left.v s\right)$
using assms(1) by auto
then obtain $v s^{\prime}$ where $v s^{\prime} \in V \wedge$ prefix $v s^{\prime}(v s @ x s) \wedge$ length $v s<$ length $v s^{\prime}$
by (meson assms(2) leI prefix-append prefix-order.dual-order.trans)
then have $\neg(m c p(v s @ x s) V v s)$
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 $\neg m c p($ map fst $(v s @ x s)) V($ map fst vs)
moreover have prefix (map fst vs) (map fst (vs @ xs))
by auto
moreover have (map fst vs) $\in V$
using mstfe-prefix-input-in-V assms(4) by auto
ultimately obtain $v^{\prime}$ where prefix $v^{\prime}($ map fst (vs @ xs))
$v^{\prime} \in V$

```
    length v'> length (map fst vs)
    using leI by auto
then obtain }\mp@subsup{x}{}{\prime}\mathrm{ where (map fst (vs@xs)) = v'@ ' 
    using prefixE by blast
have vs@xs \inL M1 - L M2
    using assms(4) unfolding minimal-sequence-to-failure-extending.simps sequence-to-failure.simps
    by blast
then have vs@xs\in Lin M1 {map fst (vs@xs)}
    by (meson DiffE insertI1 language-state-for-inputs-map-fst)
havevs@xs \in Lin M1 {v'@x'}
    using<map fst (vs @ xs)= v'@ x'><vs @ xs \in 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' \in L in M1 V
```



```
        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'〉\langleprefix v}\mp@subsup{v}{}{\prime}(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) \in 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 determinisitc 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 \(\times\) 'out) list \(\Rightarrow\) ('in, 'out, 'state) \(F S M \Rightarrow\) 'in list set \(\Rightarrow\) ('in \(\times\) 'out) list set set
    where
    \(N\) io \(M V=\left\{V^{\prime \prime} \in \operatorname{Perm} V M .\left(\right.\right.\) map fst \(\left(\right.\) mcp \(^{\prime}\) io \(\left.\left.V^{\prime \prime}\right)\right)=\left(\right.\) mcp \(^{\prime}(\) map fst io) \(\left.V)\right\}\)
```

lemma $N$-nonempty :
assumes is-det-state-cover M2 $V$
and OFSM M1
and OFSM M2
and asc-fault-domain M2 M1 m
and $\quad i o \in L M 1$
shows $N$ io M1 $V \neq\{ \}$
proof -
have [] $\in V$
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\[] G V〉\langlefinite V`] by blast
then have ioV\inV
    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) 〈ioV $\in V$ 〉] by blast
then obtain io $V^{\prime}$ io $P$ where io-path: length io $V=$ length io $V^{\prime}$
$\wedge$ length io $V=$ length ioP
$\wedge\left(\right.$ path M2 ((ioV \|io $\left.V^{\prime}\right) \|$ ioP) (initial M2))
$\wedge$ target $\left(\left(i o V \|\right.\right.$ io $\left.\left.V^{\prime}\right) \| i o P\right)($ initial M2 $)=q 2$
by auto

```
have well-formed M2
```

    using assms by auto
    have map fst $\left(\right.$ map fst $\left.\left(\left(i o V \| i o V^{\prime}\right) \| i o P\right)\right)=i o V$
proof -
have length ( ioV $\| i o V^{\prime}$ ) = length ioP
using io-path by simp
then show?thesis
using io-path by auto
qed
moreover have set (map fst (map fst $\left(\left(i o V \| i o V^{\prime}\right) \|\right.$ ioP $\left.)\right)$ ) $\subseteq$ inputs M2
using path-input-containment[OF «well-formed M2〉, of (ioV \|ioV') \|ioP initial M2 ]
io-path
by linarith
ultimately have set io $V \subseteq$ inputs M2
by presburger
then have set io $V \subseteq$ inputs M1
using assms by auto
then have $L_{i n} M 1\{i o V\} \neq\{ \}$
using assms(2) language-state-for-inputs-nonempty by (metis FSM.nodes.initial)
have prefix ioV (map fst io)
using «mcp (map fst io) VioV〉 mcp.simps by blast
then have length io $V \leq$ length (map fst io)
using prefix-length-le by blast
then have length io $V \leq$ 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)

```
                    \inlanguage-state-for-input M1 (initial M1) (map fst io)
    unfolding language-state-def
    by (metis (mono-tags, lifting)<map fst io | map snd io \inL M1>
    language-state-for-input.simps mem-Collect-eq)
have ioV = take (length ioV) (map fst io)
    by (metis (no-types)<prefix ioV (map fst io)> append-eq-conv-conj prefixE)
    then have take (length ioV) io \in language-state-for-input M1 (initial M1) ioV
    using language-state-for-input-take
    by (metis <map fst io | map snd io \inlanguage-state-for-input M1 (initial M1)(map fst io)`
        zip-map-fst-snd)
    then obtain }\mp@subsup{V}{}{\prime\prime}\mathrm{ where }\mp@subsup{V}{}{\prime\prime}\inP\mathrm{ Perm V M1 take (length ioV) io }\in\mp@subsup{V}{}{\prime\prime
    using perm-elem[OF assms(1-3)<inputs M1 = inputs M2`〈ioV \inV`] by blast
have ioV =mcp' (map fst io) V
    using «mcp (map fst io) V ioV` mcp'-intro by blast
have map fst (take (length ioV) io) = ioV
    by (metis «ioV = take (length ioV) (map fst io)` take-map)
obtain mcpV") where mcp io }\mp@subsup{V}{}{\prime\prime}mcp\mp@subsup{V}{}{\prime\prime
    by (meson 〈V'" \in Perm V M1`〈well-formed M2`assms(1) mcp-ex perm-elem-finite perm-empty)
have map fst mcp V }\mp@subsup{V}{}{\prime\prime}\inV\mathrm{ using perm-inputs
    using <\mp@subsup{V}{}{\prime\prime}\inPerm V M1\rangle\langlemcp io V' mcp V'\ mcp.simps by blast
    have map fst mcp V'\prime}=io
    by (metis (no-types)<map fst (take (length ioV) io)=ioV〉\langlemap fst mcpV'" \inV`
        <mcp (map fst io) V ioV\rangle\langlemcp io V V'mcpV'\}\langletake (length ioV) io \in V'\>
        map-mono-prefix mcp.elims(2) prefix-length-prefix prefix-order.dual-order.antisym
        take-is-prefix)
    have map fst (mcp' io V}\mp@subsup{V}{}{\prime\prime})=mc\mp@subsup{p}{}{\prime}(\mathrm{ map fst io) V
```



```
    by blast
    then show ?thesis
    using < 'V' \in Perm V M1> by fastforce
qed
lemma N-mcp-prefix :
    assumes map fst vs=mcp' (map fst (vs@xs))V
    and }\mp@subsup{V}{}{\prime\prime}\inN(vs@xs)M1
    and is-det-state-cover M2 V
    and well-formed M2
    and finite V
shows vs \in V'vs=mcp' (vs@xs) V'
proof -
    have map fst (mcp' (vs@xs) V'')=mcp'(map fst (vs@xs))V
        using assms(2) by auto
    then have map fst (mcp' (vs@xs) V') = map fst vs
        using assms(1) by presburger
    then have length (mcp' (vs@xs) V'')= length vs
        by (metis length-map)
    have [] \in V '/
        using perm-empty[OF assms(3)] N.simps assms(2) by blast
    moreover have finite }\mp@subsup{V}{}{\prime\prime
        using perm-elem-finite[OF assms(3,4)] N.simps assms(2) by blast
    ultimately obtain p where mcp(vs@xs) V" p
        using mcp-ex by auto
    then have mcp'(vs@xs) V'I}=
```

using $m c p^{\prime}$－intro by simp

```
then have prefix (mcp' (vs@xs) \(\left.V^{\prime \prime}\right)(v s @ x s)\)
    unfolding \(m c p^{\prime}\).simps mcp.simps
    using \(\left\langle m c p(v s @ x s) V^{\prime \prime} p\right\rangle m c p . e l i m s(2)\) by blast
then show \(v s=m c p^{\prime}(v s @ x s) V^{\prime \prime}\)
    by (metis 〈length \(\left(m c p^{\prime}(v s @ x s) V^{\prime \prime}\right)=\) length vs〉append-eq-append-conv prefix-def)
    show \(v s \in V^{\prime \prime}\)
    using \(\left\langle m c p(v s @ x s) V^{\prime \prime} p\right\rangle\left\langle m c p^{\prime}(v s @ x s) V^{\prime \prime}=p\right\rangle\left\langle v s=m c p^{\prime}(v s @ x s) V^{\prime \prime}\right\rangle\)
    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 \(T X \equiv\{x s @[x] \mid x s x . x s \in T \wedge x \in X\}\)
abbreviation append-sets :: 'a list set \(\Rightarrow{ }^{\prime} a\) list set \(\Rightarrow{ }^{\prime} a\) list set where
    append-sets \(T X \equiv\left\{x s @ x s^{\prime} \mid x s x s^{\prime} . x s \in T \wedge x s^{\prime} \in X\right\}\)
fun \(T S::\) ('in, 'out, 'state1) \(F S M \Rightarrow\) ('in, 'out, 'state2) FSM
    \(\Rightarrow\left({ }^{\prime}\right.\) in, 'out) \(A T C\) set \(\Rightarrow{ }^{\prime}\) in list set \(\Rightarrow\) nat \(\Rightarrow\) nat
    \(\Rightarrow{ }^{\prime}\) in list set
and \(C::(\) 'in, 'out, 'state1) \(F S M \Rightarrow(\) 'in, 'out, 'state2) FSM
    \(\Rightarrow(\) 'in, 'out \()\) ATC set \(\Rightarrow\) 'in list set \(\Rightarrow\) nat \(\Rightarrow\) nat
    \(\Rightarrow{ }^{\prime}\) in list set
and \(R M::\left({ }^{\prime}\right.\) in, 'out, 'state1) \(F S M \Rightarrow\left({ }^{\prime}\right.\) in, ' out, 'state2) \(F S M\)
            \(\Rightarrow(\) 'in, 'out) ATC set \(\Rightarrow\) 'in list set \(\Rightarrow\) nat \(\Rightarrow\) nat
            \(\Rightarrow\) 'in list set
    where
    RM M2 M1 \(\Omega \operatorname{Vm} 0=\{ \} \mid\)
    TS M2 M1 \(\Omega V m 0=\{ \} \mid\)
    TS M2 M1 \(\Omega V m(\) Suc 0) \()=V \mid\)
    \(C\) M2 M1 \(\Omega \operatorname{Vm} 0=\{ \} \mid\)
    \(C\) M2 M1 \(\Omega V m(\) Suc 0\()=V \mid\)
    RM M2 M1 \(\Omega V m(\) Suc \(n)=\)
    \(\left\{x s^{\prime} \in C\right.\) M2 M1 \(\Omega \operatorname{Vm}(\) Suc \(n)\)
        \(\left(\neg\left(L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\}\right)\right)\)
        \(\vee\left(\forall\right.\) io \(\in L_{i n} M 1\left\{x s^{\prime}\right\}\).
            \(\exists V^{\prime \prime} \in N\) io M1 \(V\).
                \(\exists S 1\) 。
                    \(\exists\) vs \(x s\).
                    \(i o=(v s @ x s)\)
                        \(\wedge m c p(v s @ x s) V^{\prime \prime} v s\)
                        \(\wedge S 1 \subseteq\) nodes M2
                \(\wedge(\forall s 1 \in S 1 . \forall s \mathcal{Z} \in S 1\).
                    \(s 1 \neq s 2 \longrightarrow\)
                        \(\left(\forall i o 1 \in R P\right.\) M2 s1 vs xs \(V^{\prime \prime}\).
                        \(\forall\) io2 \(\in R P\) M2 s2 vs xs \(V^{\prime \prime}\).
                        B M1 io1 \(\Omega \neq B\) M1 io2 \(\Omega\) ))
                \(\wedge m<L B M 2 M 1\) vs xs (TS M2 M1 \(\left.\left.\Omega V m n \cup V) S 1 \Omega V^{\prime \prime}\right)\right\} \mid\)
    C M2 M1 \(\Omega V m(\) Suc \(n)=\)
        (append-set ((CM2 M1 \(\Omega V m n)-(R M M 2 M 1 \Omega V m n))(\) inputs M2) \()\)
        \(-(T S M 2 M 1 \Omega V m n) \mid\)
    TS M2 M1 \(\Omega V m(S u c n)=\)
    \((T S M 2 M 1 \Omega V m n) \cup(C\) M2 M1 \(\Omega V m(\) Suc \(n))\)
```

```
abbreviation lists-of-length :: 'a set }=>\mathrm{ nat }=>\mathrm{ ' 'a list set where
    lists-of-length X n}\equiv{xs.length xs =n^ set xs\subseteqX
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 \mp@code { a p p e n d - s e t ~ ( a p p e n d - s e t s ~ T ~ ( l i s t s - o f - l e n g t h ~ X ~ n ) ) ~ X }
    proof
        fix tx assume tx a append-sets T (lists-of-length X (Suc n))
        then obtain tx}\mathrm{ where t@x=tx t T T length x=Suc n set x}\subseteq
        by blast
    then have x 
        by auto
        moreover have set (butlast x)\subseteqX
            using <set x\subseteqX> by (meson dual-order.trans prefixeq-butlast set-mono-prefix)
            ultimately have butlast }x\in\mathrm{ lists-of-length X n
                by auto
    then have t@(butlast x) \in append-sets T (lists-of-length X n)
            using <t }\inT\rangle\mathrm{ by blast
        moreover have last x}\in
            using <set x\subseteqX\rangle\langlex\not= []> by auto
            ultimately have t@(butlast x)@[last x] E append-set (append-sets T (lists-of-length X n))X
                by auto
            then show tx f append-set (append-sets T (lists-of-length X n)) X
            using <t@x=tx> by (simp add: <x = []>)
    qed
    show append-set (append-sets T (lists-of-length X n)) X
                    \subseteq \mp@code { a p p e n d - s e t s ~ T ~ ( l i s t s - o f - l e n g t h ~ X ~ ( S u c ~ n ) ) }
    proof
        fix tx assume tx append-set (append-sets T (lists-of-length X n)) X
        then obtain tx' x where tx=t\mp@subsup{x}{}{\prime}@[x] tx' \in append-sets T (lists-of-length X n) x 
            by blast
    then obtain t\mp@subsup{x}{}{\prime\prime}\mp@subsup{x}{}{\prime}\mathrm{ where tx"@ }\mp@subsup{x}{}{\prime}=t\mp@subsup{x}{}{\prime}t\mp@subsup{x}{}{\prime\prime}\inT\mathrm{ length }\mp@subsup{x}{}{\prime}=n\mathrm{ set }\mp@subsup{x}{}{\prime}\subseteqX
            by blast
    then have tx''@ x'@ [x]=tx
                by (simp add: <tx = tx' @ [x]>)
    moreover have tx" }\in
        by (meson <tx" ' 
    moreover have length ( }\mp@subsup{x}{}{\prime}@[x])=\mathrm{ Suc n
        by (simp add: <length }\mp@subsup{x}{}{\prime}=n>
        moreover have set ( }\mp@subsup{x}{}{@}@[x])\subseteq
            by (simp add: <set \mp@subsup{x}{}{\prime}\subseteqX\rangle\langlex\inX>)
```



```
        by blast
    qed
qed
```

5.4 Basic properties of the test suite calculation functions

```
lemma C-step :
    assumes n>0
    shows C M2 M1 \OmegaVm(Suc n)\subseteq(append-set (CM2 M1 \Omega Vmn)(inputs M2)) - C M2 M1 \OmegaVmn
proof -
    let ?TS = \lambdan.TS M2 M1 \Omega Vmn
    let ?C = \lambdan.C M2 M1 \OmegaVmn
    let ?RM = \lambdan.RM M2 M1 \OmegaVmn
    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)\subseteqappend-set (?C n) (inputs M2) - ?C n
```

```
    by blast
qed
```

lemma $C$-extension :
C M2 M1 $\Omega V m($ Suc $n) \subseteq$ append-sets $V($ lists-of-length (inputs M2) $n)$
proof (induction $n$ )
case 0
then show? case by auto
next
case (Suc $k$ )
let $? T S=\lambda n . T S M 2 M 1 \Omega V m n$
let ${ }^{2} C=\lambda n . C$ M2 $M 1 \Omega V m n$
let $? R M=\lambda n \cdot R M M 2 M 1 \Omega V m n$
have $0<S u c k$ by simp
have ?C $($ Suc $($ Suc $k)) \subseteq($ append-set $($ ? $C(S u c k))($ inputs M2) $)-$ ? $C(S u c k)$
using C-step $[$ OF $\langle 0<$ Suc $k\rangle$ ] 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 Vkinputs M2] I-Step
by presburger
qed
lemma TS-union :
shows TS M2 M1 $\Omega V m i=(\bigcup j \in(\operatorname{set}[0 . .<S u c i])$. CM2 M1 $\Omega V m j)$
proof (induction $i$ )
case 0
then show ?case by auto
next
case (Suc i)
let ? $T S=\lambda n$. TS M2 M1 $\Omega V m n$
let ? $C=\lambda n$. C M2 M1 $\Omega V m n$
let ? $R M=\lambda n . R M M 2 M 1 \Omega V m n$
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 $i)=(\bigcup j \in(\operatorname{set}[0 . .<S u c i])$. ? $C j) \cup$ ? $C(S u c i)$
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 ? $\mathrm{TS}=\lambda n . T S M 2 M 1 \Omega V m n$
let $? C=\lambda n . C$ M2 $M 1 \Omega V m n$
let $? R M=\lambda n \cdot R M M 2 M 1 \Omega V m n$

```
    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 Vm j] by fastforce
    ultimately show ?thesis
        by blast
qed
lemma C-disj-lt :
    assumes i<j
shows C M2 M1 \Omega Vmi \capC M2 M1 \Omega Vmj={}
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\not=j
shows C M2 M1 \Omega Vmi \capC M2 M1 \Omega Vmj ={}
    by (metis C-disj-lt Int-commute antisym-conv3 assms)
```

lemma RM-subset: RM M2 M1 $\Omega V m i \subseteq C M 2 M 1 \Omega V m i$
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 RMM2 M1 $\Omega V m i \cap R M M 2 M 1 \Omega V m(S u c j)=\{ \}$
proof -
let $? T S=\lambda n . T S M 2 M 1 \Omega V m n$
let ? $C=\lambda n . C M 2 M 1 \Omega V m n$
let $? R M=\lambda n . R M$ M2 M1 $\Omega V m n$
have ?RM $i \subseteq$ ? $C i$ ? RM $(S u c j) \subseteq$ ? $C($ 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 \OmegaVm(Suc n) - TS M2 M1 \OmegaVmn
                \subseteq(append-set (TS M2 M1 \Omega Vm n) (inputs M2)) - TS M2 M1 \Omega Vm n
proof -
    let ?TS = \lambda n.TS M2 M1 \Omega Vm n
    let ?C = \lambda n.C M2 M1 \Omega Vm n
    let ?RM = \lambda n. RMM2 M1 \Omega Vmn
    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 using C.simps(3) by blast
    then have ?C (Suc n)\subseteq append-set (?C n) (inputs M2) - ?TS n
        by blast
    moreover have ?C n\subseteq ?TS n using TS-union[of M2 M1 \Omega Vm n]
        by fastforce
    ultimately have ?C (Suc n)\subseteq append-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 \Omega Vm k] using n-def by blast
    ultimately show ?thesis
        by blast
qed
```

lemma append-set-prefix :
assumes $x s \in$ append-set $T X$
shows butlast $x s \in T$
using assms by auto
lemma C-subset : C M2 M1 $\Omega V m i \subseteq T S$ M2 M1 $\Omega V m i$
by (simp add: TS-union)

```
lemma TS-subset :
    assumes i\leqj
    shows TS M2 M1 \Omega Vmi\subseteqTSM2 M1 \OmegaVmj
proof -
    have TS M2 M1 \Omega Vm i=(\bigcupk f (set [0..<Suc i]). C M2 M1 \Omega Vmk)
            TS M2 M1 \Omega Vm j=(\bigcupk\in(set [0..<Suc j]).CM2 M1 \Omega Vmk)
        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 $v s @ x s \in C M 2 M 1 \Omega V m(S u c(S u c i))$
and $\quad x s \neq[]$
shows vs@(butlast xs $) \in C$ M2 M1 $\Omega V m($ Suc $i)-R M M 2 M 1 \Omega V m($ Suc $i)$
proof (rule ccontr)
let ? $T S=\lambda n$. TS M2 M1 $\Omega V m n$
let ? $C=\lambda n$. $C$ M2 M1 $\Omega V m n$
let ? $R M=\lambda n$. RMM2M1 $\Omega V m n$
assume vs @ butlast xs $\notin C M 2 M 1 \Omega V m($ Suc $i)-R M M 2 M 1 \Omega V m($ Suc $i)$
have ?C $($ Suc $($ Suc $i)) \subseteq$ append-set $(? C$ (Suc $i)-$ ?RM (Suc i)) (inputs M2)
using C.simps(3) by blast
then have ?C $($ Suc $($ Suc i) $) \subseteq$ append-set $(? C$ (Suc i) - ?RM (Suc i)) UNIV

```
    by blast
    moreover have vs @ xs & append-set (?C (Suc i) - ?RM (Suc i)) UNIV
    proof -
    have \forallas a.vs @ xs \not= as @ [a]
                    \vee as\not\inCM2 M1 \OmegaVm(Suc i) - RMM2 M1 \OmegaVm(Suc i)
                    v a\not\inUNIV
        by (metis «vs @ butlast xs &C M2 M1 \Omega Vm (Suc i) - RM M2 M1 \Omega Vm (Suc i)>
            assms(2) butlast-append butlast-snoc)
    then show ?thesis
        by blast
    qed
    ultimately have vs @ xs &?C (Suc (Suc i))
    by blast
    then show False
    using assms(1) by blast
qed
```

lemma TS-immediate-prefix-containment :
assumes $v s @ x s \in T S M 2 M 1 \Omega V m i$
and $\quad m c p(v s @ x s) V v s$
and $0<i$
shows vs@(butlast xs) $\in$ TS M2 M1 $\Omega V m i$
proof -
let ? $T S=\lambda n . T S M 2 M 1 \Omega V m n$
let ? $C=\lambda n . C$ M2 M1 $\Omega V m n$
let $? R M=\lambda n . R M M 2 M 1 \Omega V m n$
obtain $j$ where $j$-def : $j \leq i \wedge v s @ x s \in ? C j$
using $\operatorname{assms}(1)$ TS-union[where $i=i]$
proof -
assume a1: $\bigwedge j . j \leq i \wedge v s @ x s \in C$ M2 M1 $\Omega V m j \Longrightarrow$ thesis
obtain $n n::$ nat set $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right.$ list set $) \Rightarrow$ 'a list $\Rightarrow$ nat where
f2: $\forall x 0 x 1 x 2 .(\exists v 3 . v 3 \in x 0 \wedge x 2 \in x 1 v 3)=(n n x 0 x 1 x 2 \in x 0 \wedge x 2 \in x 1(n n x 0 x 1 x 2))$
by moura
have vs @ xs $\in$ UNION (set $[0 . .<$ Suc i $]$ ) (C M2 M1 $\Omega$ V m)
by (metis $\wedge \bigwedge \Omega V T S M 2$ M1. TS M2 M1 $\Omega V m i=(\bigcup j \in$ set $[0 . .<$ Suc $i]$. C M2 M1 $\Omega V m j)$,
$\langle v s @ x s \in T S M 2 M 1 \Omega V m i\rangle)$
then have nn (set $[0 . .<$ Suc $i])(C$ M2 M1 $\Omega V m)(v s @ x s) \in \operatorname{set}[0 . .<$ Suc $i]$
$\wedge v s @ x s \in C M 2 M 1 \Omega V m(n n(\operatorname{set}[0 . .<$ Suc $i])(C M 2 M 1 \Omega V m)(v s @ x s))$
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 $m c p(v s @ x s) V(v s @ x s)$
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<vs @ xs = vs〉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<?C (Suc (Suc n))\subseteq append-set (?C (Suc n)) (inputs M2)>
                <v @ xs \in ?C (Suc (Suc n))> append-set-prefix subsetCE)
    qed
    moreover have xs }\not=[
    proof -
        have 1\leqk
            using j-assms by auto
        then have ?C }j\cap\mathrm{ ? C 1 = {}
            using C-disj-le-gz[of 1 k] j-assms(1) less-numeral-extra(1) by blast
            then have ? C j\capV={}
            by auto
        then have vs@xs \not\inV
            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 Vm j]〈Suc n < Suc j〉
        by (metis UN-upper atLeast-upt lessThan-iff)
            have vs@ butlast xs \inTS M2 M1 \Omega Vm j
            using <vs@(butlast xs) \in?C (Suc n)><?C (Suc n)\subseteq?TS j> 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 $v s @ x s \in T S M 2 M 1 \Omega V m i$
and $m c p(v s @ x s) V v s$
and prefix $x s^{\prime} x s$
shows vs@xs' $\in$ TS M2 M1 $\Omega$ Vmi

- 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 $x s=x s^{\prime}$
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) \(<0<i\rangle T S . s i m p s(2) T S\)-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 \(=x s^{\prime}\) )
            case True
            then show ?thesis
            using assms(1) by simp
        next
        case False
        then obtain \(x s^{\prime \prime}\) where \(x s=x s^{\prime} @ x s^{\prime \prime}\)
            using Suc.prems(3) prefixE by blast
        then have \(x s^{\prime \prime} \neq[]\)
            using False by auto
        then have \(k=\) length \(x s-\) length \(\left(x s^{\prime} @\left[h d x s^{\prime \prime}\right]\right)\)
            using 〈xs = xs'@xs">Suc.hyps(2) by auto
        moreover have prefix (xs' @ [hd xs \(\left.{ }^{\prime}\right]\) ) xs
            using \(\left\langle x s=x s^{\prime} @ x s^{\prime \prime}\right\rangle\left\langle x s^{\prime \prime} \neq[]\right\rangle\)
            by (metis Cons-prefix-Cons list.exhaust-sel prefix-code(1) same-prefix-prefix)
        ultimately have vs @ \(\left(x s^{\prime}\right.\) @ \(\left.\left[h d x s^{\prime}\right]\right) \in T S\) M2 M1 \(\Omega\) Vmi
            using Suc.hyps(1)[OF - Suc.prems(1,2)] by simp
        have \(m c p\) (vs @ \(x s^{\prime}\) @ \(\left.\left[h d x s^{\prime}\right]\right) V v s\)
            using \(\left\langle x s=x s^{\prime} @ x s^{\prime \prime}\right\rangle\left\langle x s^{\prime \prime} \neq[]\right\rangle \operatorname{assms}(2)\)
        proof -
            obtain aas :: 'a list \(\Rightarrow\) 'a list set \(\Rightarrow\) 'a list \(\Rightarrow\) ' \(a\) list where
                \(\forall x 0 x 1 x 2 .(\exists v 3\). (prefix v3 x2 \(\wedge v 3 \in x 1) \wedge \neg\) length \(v 3 \leq\) length \(x 0)\)
                    \(=((\) prefix \((\) aas x0 x1 x2) x2 \(\wedge\) aas x0 x1 x2 \(\in \overline{x 1})\)
                        \(\wedge \neg\) length (aas x0 x1 x2) \(\leq\) length x0)
            by moura
            then have \(f 1: \forall\) as \(A\) asa. ( \(\neg\) mcp as \(A\) asa
                                    \(\vee\) prefix asa as \(\wedge\) asa \(\in A \wedge(\forall\) asb. \((\neg\) prefix asb as \(\vee\) asb \(\notin A)\)
                                    \(\vee\) length asb \(\leq\) length asa))
                                    \(\wedge(m c p\) as \(A\) asa
                                \(\checkmark \neg\) prefix asa as
                                \(\vee\) asa \(\notin A\)
                                \(\vee(\) prefix (aas asa \(A\) as) as \(\wedge\) aas asa \(A\) as \(\in A)\)
                                \(\wedge \neg\) length (aas asa \(A\) as \() \leq\) length asa)
by auto
obtain aasa \(::\) 'a list \(\Rightarrow\) ' \(a\) list \(\Rightarrow\) 'a list where
f2: \(\forall x 0 x 1\). \((\exists v 2 . x 0=x 1\) @ v2 \()=(x 0=x 1\) @ aasa \(x 0 x 1)\)
by moura
then have \(f 3\) : ([] @ [hd xs \(\left.\left.{ }^{\prime}\right]\right)\) @ aasa \(\left(x s^{\prime}\right.\) @ \(\left.x s^{\prime \prime}\right)\left(x s^{\prime}\right.\) @ \(\left.\left[h d x s^{\prime}\right]\right)\)
\[
\begin{aligned}
= & \left([] @\left[h d x s^{\prime \prime}\right]\right) @ \text { aasa }\left(\left([] @\left[h d x s^{\prime \prime}\right]\right)\right. \\
& \left.@ \text { aasa }\left(x s^{\prime} @ x s^{\prime \prime}\right)\left(x s^{\prime} @\left[h d x s^{\prime \prime}\right]\right)\right)\left([] @\left[h d x s^{\prime}\right]\right)
\end{aligned}
\]
by (meson prefixE prefixI)
have \(x s^{\prime} @ x s^{\prime \prime}=\left(x s^{\prime} @\left[h d x s^{\prime}\right]\right)\) @ aasa \(\left(x s^{\prime} @ x s^{\prime \prime}\right)\left(x s^{\prime} @\left[h d x s^{\prime}\right]\right)\)
```

```
            using f2 by (metis (no-types)<prefix (xs' @ [hd x\mp@subsup{s}{}{\prime}]) xs><xs = xs' @ xs'>> prefixE)
            then have (vs @ (a # list) @ [hd xs']) @ aasa (([] @ [hd x\mp@subsup{s}{}{\prime}|)
                    @ aasa (x\mp@subsup{s}{}{\prime}@ x\mp@subsup{s}{}{\prime\prime})(x\mp@subsup{s}{}{\prime}@[hdx\mp@subsup{s}{}{\prime\prime}]))([] @ [hd x\mp@subsup{s}{}{\prime\prime}])
                    =vs@ xs
            using f3 by (simp add: <xs=xs' @ xs'>}> local.Cons
            then have ᄀ prefix (aasvs V (vs @ x\mp@subsup{s}{}{\prime}@ [hd x\mp@subsup{s}{}{\prime}]))(vs@ @s\mp@subsup{s}{}{\prime}@[hdx\mp@subsup{s}{}{\prime\prime}])
                V aas vs V (vs @ xs' @ [hd xs'']) &V
                    V length (aas vs V (vs @ xs' @ [hd xs''])) \leq 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']}\mp@subsup{}{}{\prime}])\inTS M2 M1 \Omega Vmi
            using TS-immediate-prefix-containment
            [OF<vs@ (xs'@ [hd x\mp@subsup{s}{}{\prime}])\inTS M2 M1 \OmegaVmi>-<0<i\rangle]
            by simp
            moreover have }x\mp@subsup{s}{}{\prime}=\mathrm{ butlast (xs'@ [hd xs''])
            using «x\mp@subsup{s}{}{\prime\prime}\not=[]〉 by simp
            ultimately show ?thesis
            by simp
        qed
    qed
qed
```

```
lemma C-index :
    assumes \(v s\) @ \(x s \in C\) M2 M1 \(\Omega V m i\)
    and \(m c p(v s @ x s) V v s\)
shows Suc (length xs) \(=i\)
using assms proof (induction xs arbitrary: i rule: rev-induct)
    case Nil
    then have vs @ [] \(\in C\) M2 M1 \(\Omega V m 1\)
        by auto
    then have vs @ [] \(\in C\) M2 M1 \(\Omega \operatorname{Vm}\) (Suc (length []))
        by \(\operatorname{simp}\)
    show? case
    proof (rule ccontr)
        assume Suc (length []\() \neq i\)
        moreover have vs @ [] CC M2 M1 \(\Omega V m i \cap C\) M2 M1 \(\Omega V m\) (Suc (length []))
            using Nil.prems(1) <vs @ [] C M2 M1 \(\Omega V m\) (Suc (length [])) >by auto
        ultimately show False
            using C-disj by blast
    qed
next
    case (snoc \(\left.x x s^{\prime}\right)\)
    let \(? T S=\lambda n . T S M 2 M 1 \Omega V m n\)
    let ? \(C=\lambda n . C\) M2 M1 \(\Omega V m n\)
    let \(? R M=\lambda n . R M M 2 M 1 \Omega V m n\)
    have vs @ \(x s^{\prime}\) @ \([x] \notin V\)
        using snoc.prems(2) by auto
    then have vs @ \(x s^{\prime} @[x] \notin ? C\) 1
        by auto
    moreover have vs @ \(x s^{\prime} @[x] \notin ? C 0\)
        by auto
```

```
    ultimately have 1<i
    using snoc.prems(1) by (metis less-one linorder-neqE-nat)
    then have vs @ butlast (xs' @ [x]) \inC M2 M1 \Omega Vm(i-1)
    proof -
        have Suc 0<i
        using <1<i> 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 <1 < i`)
    then show ?thesis
        using f1 by (metis C-immediate-prefix-containment DiffD1 One-nat-def Suc-pred' snoc.prems(1)
                snoc-eq-iff-butlast)
    qed
    moreover have mcp(vs @ butlast (xs' @ [x])) V vs
    by (meson mcp-prefix-of-suffix prefixeq-butlast snoc.prems(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 \inTS M2 M1 \Omega Vmi
    and mcp(vs@xs)Vvs
shows Suc (length xs)\leqivs@xs \inC M2 M1 \Omega Vm (Suc (length xs))
proof -
    let?TS = \lambdan.TS M2 M1 \OmegaVmn
    let?C = \lambdan.CM2 M1 \OmegaVmn
    let ?RM = \lambdan.RM M2 M1 \Omega Vmn
    obtain j where j<Suc i vs@xs \in?C j
        using TS-union[of M2 M1 \Omega Vmi]
        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)\leqi
        using <j<Suc i` by auto
    show vs@xs \inC M2 M1 \Omega Vm (Suc (length xs))
    using <vs@xs \in?C j\rangle\langleSuc (length xs) = j> by auto
qed
lemma C-extension-options:
    assumes vs @ xs \inC M2 M1 \OmegaVmi
    and mcp(vs@ xs @ [x])Vvs
    and}\quadx\in\mathrm{ inputs M2
    and 0<i
shows vs@xs@[x]\inC M2 M1 \OmegaVm(Suc i)\veevs@xs \inRM M2 M1 \Omega Vmi
proof (cases vs@xs \in RM M2 M1 \Omega Vmi)
    case True
    then show ?thesis by auto
next
    case False
    let ?TS = \lambdan.TS M2 M1 \Omega Vmn
    let ?C = \lambdan.C M2 M1 \OmegaVmn
    let ?RM=\lambdan.RM M2 M1 \OmegaVmn
    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 append-set (?C i - ?RM i) (inputs M2)
    by (simp add: assms(3))
    moreover have vs@xs@[x]\not\in?TS i
    proof (rule ccontr)
    assume ᄀvs @ xs @ [x]&?TS i
    then obtain j where j<Suc i vs@xs@[x]\in?C j
        using TS-union[of M2 M1 \Omega Vmi] by fastforce
    then have Suc(length (xs@[x]))=j
        using C-index assms(2) by blast
    then have Suc (length (xs@[x]))<Suc i
        using <j<Suc i` by auto
    moreover have Suc (length xs) =i
        using C-index
        by (metis assms(1) assms(2) mcp-prefix-of-suffix prefixI)
    ultimately have Suc (length (xs@[x])) < Suc (Suc (length xs))
        by auto
    then show False
        by auto
    qed
    ultimately show ?thesis
    by (simp add:<?C (Suc i) = append-set (?C i - ?RM i) (inputs M2) - ?TS i`)
qed
```

lemma TS-non-containment-causes :
assumes vs@xs $\neq T S$ M2 M1 $\Omega V m i$
and $m c p(v s @ x s) V v s$
and set $x s \subseteq$ inputs M2
and $0<i$
shows $(\exists x r j . x r \neq x s \wedge \operatorname{prefix} x r x s \wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j)$
$\vee(\exists x c . x c \neq x s \wedge$ prefix $x c x s \wedge v s @ x c \in(C$ M2 M1 $\Omega V m i)-(R M M 2 M 1 \Omega V m i))$
(is ?PrefPreviouslyRemoved $\vee$ ?PrefJustContained)
$\neg((\exists x r j . x r \neq x s \wedge$ prefix $x r x s \wedge j \leq i \wedge v s @ x r \in R M$ M2 M1 $\Omega V m j)$
$\wedge(\exists x c . x c \neq x s \wedge$ prefix $x c x s \wedge v s @ x c \in(C M 2 M 1 \Omega V m i)-(R M M 2 M 1 \Omega V m i)))$

- 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 ? $T S=\lambda n$. TS M2 M1 $\Omega V m n$
let ? $C=\lambda n$. C M2 M1 $\Omega V m n$
let ? $R M=\lambda n$. RMM2 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 $j$. prefix xr xs $\wedge j \leq i \wedge v s @ x r \in$ ? $R M j)$
proof
assume $\exists x r j$. prefix xr xs $\wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j$
then obtain $x r j$ where prefix xr xs $j \leq i v s @ x r \in ? R M j$
by blast
then show False
proof (cases $x r=x s$ )

```
    case True
    then have vs @ xs\in??RM jusing <vs @ xr \in?RM j> by auto
    then have vs @ xs \in?TS j
        using C-subset RM-subset <vs @ xr \in?RM j> by blast
    then have vs @ xs \in?TS i
        using TS-subset <j }\leqi\rangle\mathrm{ by blast
    then show ?thesis using assms(1) by blast
    next
    case False
    then show ?thesis
        using <ᄀ ?PrefPreviouslyRemoved〉\langleprefix xr xs\rangle\langlej\leqi\rangle\langlevs@ xr \in?RM j>
        by blast
    qed
qed
```

have $v s \in V$ using assms(2) by auto
then have vs $\in$ ? $C 1$ by auto
have $\bigwedge k .(1 \leq S u c k \wedge$ Suc $k \leq i) \longrightarrow v s @($ take $k x s) \in ? C(S u c k)-? R M($ Suc $k)$
proof
fix $k$ assume $1 \leq$ Suc $k \wedge$ Suc $k \leq i$
then show vs @ $($ take $k x s) \in ? C($ Suc $k)-? R M($ Suc $k)$
proof (induction $k$ )
case 0
show ?case using $\langle v s \in ?$ ? 1$\rangle$
by (metis 0.prems DiffI One-nat-def
$\prec(\exists$ xr $j$. prefix xr xs $\wedge j \leq i \wedge v s @ x r \in R M$ M2 M1 $\Omega V m j)$ )
append-Nil2 take-0 take-is-prefix)
next
case (Suc k)
have $1 \leq$ Suc $k \wedge$ Suc $k \leq i$
using Suc.prems by auto
then have vs @ take $k x s \in$ ? $C$ (Suc $k$ )
using Suc.IH by simp
moreover have vs @ take $k$ xs $\notin ? R M$ (Suc $k$ )
using $\langle 1 \leq$ Suc $k \wedge$ Suc $k \leq i\rangle \prec \neg$ ?PrefPreviouslyRemoved〉take-is-prefix Suc.IH
by blast
ultimately have vs @ take $k x s \in(? C(S u c k))-(? R M(S u c k))$
by blast
have $k<$ length $x s$
proof (rule ccontr)
assume $\neg k<$ length $x s$
then have vs @ xs $\in$ ? $C$ (Suc k) using «vs @ take kxs $\in$ ? $C$ (Suc k) >
by $\operatorname{simp}$
have vs @ $x s \in$ ?TS $i$
by (metis C-subset TS-subset $\langle 1 \leq S u c k \wedge S u c k \leq i\rangle\langle v s @ x s \in ? C(S u c k)\rangle$
contra-subsetD)
then show False
using assms(1) by simp
qed
moreover have set $x s \subseteq$ 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 append-set ((?C (Suc k)) - (?RM (Suc k))) (inputs M2)
proof -
have $f 1: x s!k \in$ inputs M2
by (meson 〈k<length xs〉〈set xs $\subseteq$ inputs M2〉 nth-mem subset-iff)
have vs @ take (Suc k) xs =(vs @ take $k x s) @[x s!k]$
by (simp add: 〈 $k<$ length xs〉 take-Suc-conv-app-nth)

```
        then show ?thesis
            using f1<vs @ take k xs C C M2 M1 \Omega Vm (Suc k) - RM M2 M1 \Omega Vm (Suc k)` by blast
        qed
        moreover have vs @ take (Suc k) xs \not\in?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 \vs @ take k xs \in?C (Suc k)`
            by (metis assms(2) mcp-prefix-of-suffix take-is-prefix)
            ultimately show False using <k< length xs>
            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 Vmk]
    by (metis (no-types, lifting) DiffI Suc.prems
            \imath(\exists xr j. prefix xr xs ^j\leqi^vs @ xr \inRM M2 M1 \Omega Vmj)>
            <vs @ take (Suck) xs \not\inTS M2 M1 \OmegaVm(Suc k)> calculation take-is-prefix)
    qed
    qed
    then have vs @ take (i-1) xs \inC M2 M1 \Omega Vmi - RM M2 M1 \Omega Vmi
    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 <ᄀ ?PrefJustContained` by simp
qed
```

```
show \(\neg(\) ?PrefPreviouslyRemoved \(\wedge\) ?PrefJustContained \()\)
proof
    assume ?PrefPreviouslyRemoved \(\wedge\) ?PrefJustContained
    then have ?PrefPreviouslyRemoved
        ?PrefJustContained
    by auto
    obtain \(x r j\) where prefix xr xs \(j \leq i v s @ x r \in ? R M j\)
    using 〈?PrefPreviouslyRemoved» by blast
obtain \(x c\) where prefix xc xs vs@xc \(\in\) ? \(C i-\) ? \(R M i\)
    using 〈?PrefJustContained» by blast
then have Suc (length xc) \(=i\)
    using C-index
    by (metis Diff-iff assms(2) mcp-prefix-of-suffix)
moreover have length \(x c \leq\) length \(x s\)
    using \(\langle p r e f i x\) xc \(x s\rangle\) by (simp add: prefix-length-le)
moreover have \(x c \neq x s\)
proof
    assume \(x c=x s\)
    then have vs@xs \(\in\) ? \(C i\)
        using 〈vs@xc \(\in\) ? \(C i-\) ? RM i〉 by auto
    then have vs@xs \(\in\) ? TS \(i\)
        using \(C\)-subset by blast
    then show False
        using assms(1) by blast
    qed
ultimately have \(i \leq\) length \(x s\)
        using «prefix xc xs〉 not-less-eq-eq prefix-length-prefix prefix-order.antisym
        by blast
```

```
have \(\bigwedge n .(n<i) \Longrightarrow v s @(\) take \(n x s) \in\) ? \(C\) (Suc \(n)\)
proof -
    fix \(n\) assume \(n<i\)
    show vs @ taken xs \(\in C\) M2 M1 \(\Omega V m\) (Suc n)
    proof -
        have \(n \leq\) length \(x c\)
        using \(\langle n<i\rangle\langle\) Suc (length \(x c\) ) \(=i\rangle\) less-Suc-eq-le
        by blast
    then have prefix (vs @ (take n xs)) (vs@xc)
    proof -
        have \(n \leq\) length \(x s\)
            using 〈length \(x c \leq\) length \(x s\rangle\langle n \leq\) length \(x c\rangle\) order-trans
            by blast
        then have prefix (take \(n\) xs) xc
            by (metis (no-types) 〈n \(\leq\) length \(x c\rangle\langle p r e f i x x c\) xs〉 length-take min.absorb2
                prefix-length-prefix take-is-prefix)
        then show?thesis
            by \(\operatorname{simp}\)
        qed
        then have vs @ take nxs \(\in\) ?TS i
        by (meson C-subset DiffD1 TS-prefix-containment 〈prefix xc xs〉
                \(\langle v s @ x c \in C M 2 M 1 \Omega V m i-R M M 2 M 1 \Omega V m i\rangle\) assms(2) contra-subsetD
                mcp-prefix-of-suffix same-prefix-prefix)
        then obtain \(j n\) where \(j n<\) Suc \(i\) vs \(@(\) take \(n x s) \in ? C\) jn
        using TS-union[of M2 M1 \(\Omega\) Vmi]
        by (metis UN-iff atLeast-upt lessThan-iff)
    moreover have mcp (vs @ take nxs) V vs
        by (meson assms(2) mcp-prefix-of-suffix take-is-prefix)
    ultimately have \(j n=\) Suc (length (take \(n x s)\) )
        using C-index[of vs take n xs M2 M1 \(\Omega V m j n]\) by auto
        then have \(j n=\) Suc \(n\)
        using 〈length \(x c \leq\) length \(x s\rangle\langle n \leq\) length \(x c\rangle\) by auto
        then show \(v s @(\) take \(n x s) \in ? C(\) Suc \(n)\)
        using \(\langle v s @(t a k e n x s) \in ? C\) jn> by auto
    qed
qed
have \(\bigwedge n .(n<i) \Longrightarrow v s @(\) take \(n x s) \notin ? R M(\) Suc \(n)\)
proof -
    fix \(n\) assume \(n<i\)
    show vs @ take nxs \(\notin R M\) M2 M1 \(\Omega V m(\) Suc n)
    proof (cases \(n=\) length \(x c\) )
        case True
        then show ?thesis
            using \(\langle v s @ x c \in ? C i-? R M i\rangle\)
            by (metis DiffD2 〈Suc (length \(x c\) ) \(=i\rangle\langle p r e f i x x c\) xs〉 append-eq-conv-conj prefixE)
    next
        case False
        then have \(n<\) length \(x c\)
            using \(\langle n<i\rangle\langle\) Suc (length \(x c\) ) \(=i\rangle\) by linarith
        show ?thesis
        proof (cases Suc \(n<\) length xc)
        case True
        then have Suc \(n<i\)
            using \(\langle\) Suc (length \(x c\) ) \(=i\rangle\langle n<\) length \(x c\rangle\) by blast
        then have vs @ (take (Suc n) xs) \(\in\) ? C (Suc (Suc n))
            using \(\langle\bigwedge n .(n<i) \Longrightarrow v s @(\) take \(n x s) \in ? C(\) 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 \(\Omega\) Vmn]
            by (metis Suc-neq-Zero 〈prefix xc xs〉〈xc \(\neq\) xs〉 prefix-Nil take-eq-Nil)
        then show?thesis
            by (metis DiffD2 Suc-lessD True «length xc \(\leq\) length xs» butlast-snoc less-le-trans
```

```
                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) \in?C (Suc (Suc n))
                using<Suc (length xc) = i\rangle<\bigwedgen.n<i\Longrightarrowvs@ take n xs \inCM2 M1 \OmegaVm (Suc n)>
                    by auto
            then have vs @ butlast (take (Suc n) xs) \in?C (Suc n) - ?RM (Suc n)
                using False C-immediate-prefix-containment[of vs take (Suc n) xs M2 M1 \Omega Vm n]
                    by (metis Suc-neq-Zero <prefix xc xs\rangle\langlexc \not= xs〉 prefix-Nil take-eq-Nil)
            then show ?thesis
                by (metis Diff-iff 〈Suc n = length xc><length xc \leq length xs`butlast-take diff-Suc-1)
            qed
        qed
    qed
    have xr= take jxs
    proof -
        have vs@xr \in?C j
            using <vs@xr \in ?RM j> RM-subset by blast
            then show ?thesis
            using C-index
            by (metis Suc-le-lessD<\n.n<i\Longrightarrowvs@ take n xs \not=RMM2 M1 \Omega Vm (Suc n)><j\leqi>
                <prefix xr xs〉\langlevs@ xr \in RM M2 M1 \Omega Vm j〉append-eq-conv-conj assms(2)
                mcp-prefix-of-suffix prefix-def)
    qed
    have vs@xr &?RMj
    by (metis (no-types) C-index RM-subset <i\leqlength xs〉<j \leqi\rangle<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 \in?RM j> by simp
    qed
qed
```

lemma TS－non－containment－causes－rev ：
assumes mcp（vs＠xs）Vvs
and $(\exists x r j . x r \neq x s \wedge \operatorname{prefix} x r x s \wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j)$
$\vee(\exists x c . x c \neq x s \wedge$ prefix xc $x s \wedge v s @ x c \in(C M 2 M 1 \Omega V m i)-(R M M 2 M 1 \Omega V m i))$
(is ?PrefPreviouslyRemoved $\vee$ ?PrefJustContained)
shows $v s @ x s \notin T S M 2 M 1 \Omega V m i$
proof
let ? $T S=\lambda n$. TS M2 M1 $\Omega V m n$
let ? $C=\lambda n$. C M2 M1 $\Omega V m n$
let ? $R M=\lambda n$. RMM2 M1 $\Omega V m n$
assume vs @ xs $\in T S M 2 M 1 \Omega V m i$
have ?PrefPreviouslyRemoved $\Longrightarrow$ False
proof -
assume ?PrefPreviouslyRemoved
then obtain $x r j$ where $x r \neq x s$ prefix xr xs $j \leq i v s @ x r \in ? R M j$
by blast
then have $v s @ x r \notin ? C j-? R M j$
by blast
have $v s @($ take $($ Suc $($ length $x r))$ xs $) \notin ? C$ (Suc j)

```
    proof -
    have vs@(take (length xr)xs)\not\in?C j - ?RM j
        by (metis <prefix xr xs><vs @ xr #C M2 M1 \Omega Vmj - RM M2 M1 \OmegaVm j>
            append-eq-conv-conj prefix-def)
    show ?thesis
    proof (cases j)
        case 0
        then show ?thesis
            using RM.simps(1)<vs @ xr \inRM M2 M1 \OmegaVm j> 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 «prefix xr xs\rangle\langlexr # xs` append-eq-conv-conj not-le prefix-def
                    take-Suc-conv-app-nth take-all)
        have vs@(take (length xr) xs)@ [x] & append-set (?C j - ?RM j) (inputs M2)
            using <vs@(take (length xr) xs) &?C j - ?RM j> by simp
        then have vs@(take (length xr) xs) @ [x] &?C (Suc j)
            using <?C (Suc j)\subseteq append-set (?C j - ?RM j) (inputs M2)` by blast
        then show?thesis
            using <vs@(take (Suc (length xr)) xs)=vs@(take (length xr) xs)@ [x]> 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 <vs @ xs \inTS M2 M1 \Omega V m i` assms(1)] by simp
    then obtain j' where j'<Suc i}\wedgevs@(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) <prefix xr xs`<xr \not= xs` append-Nil2 append-eq-conv-conj leD
            nat-less-le prefix-def prefix-length-le)
    then show ?thesis
        by (metis (no-types)<\bigwedgei\Omega V TS M2 M1. \llbracketvs @ take (Suc (length xr)) xs \inC M2 M1 \Omega Vmi;
                        mcp (vs @ take (Suc (length xr)) xs) V vs\rrbracket
                    Suc (length (take (Suc (length xr)) xs)) = i>
            <j'<Suc i^vs @ take (Suc (length xr)) xs \inC M2 M1 \OmegaV m j'>
            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 <vs@xr \in?RM j> RM-subset C-index
    by (metis <prefix xr xs` 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 <j'< Suc i}^vs@(take (Suc (length xr)) xs) \in?C j'> by aut
    then show False
    using <vs@(take (Suc (length xr)) xs) & ?C (Suc j)> by blast
qed
moreover have ?PrefJustContained \Longrightarrow False
proof -
    assume ?PrefJustContained
    then obtain xc where xc\not= xs
                        prefix xc xs
                        vs@ xc\in?C}i-?RM
    by blast
```

```
    - only possible if xc = xs
    then show False
        by (metis C-index DiffD1 Suc-less-eq TS-index(1) <vs @ xs \in?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-ref)
    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 \(\Omega V m n\) )
using assms proof (induction \(n\) )
    case 0
    then show? case by auto
next
    case (Suc n)
    let \(? T S=\lambda n . T S M 2 M 1 \Omega V m n\)
    let \(? C=\lambda n . C\) M2 \(M 1 \Omega V m n\)
    let \(? R M=\lambda n . R M M 2 M 1 \Omega V m n\)
    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 \(n)=? T S n \cup ? C(\) Suc \(n)\)
            by (metis TS.simps(3) grO-implies-Suc neq0-conv)
        moreover have finite (?TS n)
            using Suc.IH[OF Suc.prems] by assumption
        moreover have finite (?C (Suc n))
        proof -
            have ?C (Suc n) \(\subseteq\) append-set (?C n) (inputs M2)
                using \(C\)-step False by blast
            moreover have ? \(C n \subseteq\) ? TS \(n\)
                by (simp add: C-subset)
            ultimately have ?C (Suc n) \(\subseteq\) append-set (?TS n) (inputs M2)
                by blast
            moreover have finite (append-set (?TS n) (inputs M2))
                by (simp add: 〈finite (TS M2 M1 \(\Omega V m n)\rangle\) 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 \(\Omega V m n\) )
proof -
    have \(C\) M2 \(M 1 \Omega V m n \subseteq T S M 2 M 1 \Omega V m n\)
        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 Vmi\equivTS M2 M1 \OmegaVmi=TS M2 M1 \OmegaVm(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 Vm(Suc (|M2| *m))
proof -
    let ?i = Suc ( }|M2|*m
    let ?TS = \lambdan.TSM2 M1 \OmegaVmn
    let ?C = \lambdan.C M2 M1 \Omega Vmn
    let ?RM = \lambdan. RM M2 M1 \Omega Vmn
    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 }\forallseq\in?C ?i. seq \in?RM ?
    proof
        fix seq assume seq \in?C ?i
    show seq \in?RM ?i
    proof -
        have [] ] 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 \inC M2 M1 \OmegaVm(Suc (|M2|*m))` by blast
        then have length xs =( |M2| * m) by auto
        have RM-def: ?RM ?i = {x\mp@subsup{s}{}{\prime}\inC M2 M1 \Omega Vm ?i .
                        (\neg(\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}\subseteq\mp@subsup{L}{in}{}M2{x\mp@subsup{s}{}{\prime}}))
                        \vee (\forall io \in Lin M1 {xs'}.
                            (\exists V'|}\inN\mathrm{ io M1 V.
                            (\exists S1.
                                    (\exists vs xs .
```

$$
i o=(v s @ x s)
$$

$\wedge m c p(v s @ x s) V^{\prime \prime} v s$
$\wedge S 1 \subseteq$ nodes M2
$\wedge(\forall s 1 \in S 1 . \forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
$\left(\forall\right.$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall$ io2 $\in R P$ M2 s2 vs xs $V^{\prime \prime}$.
B M1 io1 $\Omega \neq B$ M1 io2 $\Omega$ ))
$\wedge m<L B M 2 M 1$ vs xs $\left.\left.\left.\left.\left.(? T S((|M 2| * m)) \cup V) S 1 \Omega V^{\prime \prime}\right)\right)\right)\right)\right\}$
using RM.simps(2)[of M2 M1 $\Omega \operatorname{Vm}((\operatorname{card}($ nodes M2) $) * m)]$ by assumption

```
have ( }\neg(\mp@subsup{L}{in}{}M1{seq}\subseteq\mp@subsup{L}{in}{}M2{seq})
    \vee (\forall io \in Lin M1 {seq}.
        (\exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V .
            (\exists S1.
            (\exists vs xs.
                io =(vs@xs)
                ^mcp(vs@xs) V' vs
                ^S1\subseteq nodes M2
                \wedge (\forall s1 \inS1.}\forall\mp@code{s2 \inS1.
                s1 }=s2
                    (\forall io1 \in RP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}
                    io2 \inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime
                            B M1 io1 \Omega\not=B M1 io2 \Omega ))
                \wedge m<LB M2 M1 vs xs (?TS (( |M2|*m))\cupV)S1 \Omega V''))))
proof (cases (\neg)(\mp@subsup{L}{in}{}M1 {seq}\subseteq Lin M2 {seq})))
    case True
    then show ?thesis
        using RM-def by blast
next
    case False
    have ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{}M1{seq}
            (\exists V V' }\inN\mathrm{ io M1 V .
                (\exists S1.
                    (\exists vs xs.
                    io =(vs@xs)
                    ^mcp(vs@xs) V'vs
                    \wedge S1\subseteq nodes M2
                    \wedge (\forall s1 \inS1.}\forall \2 \inS1
                        s1 = s2\longrightarrow
                    ( }\forall\mathrm{ io1 }\inRP M2 s1 vs xs V'".
                        \forallo2 \in RP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}
                        B M1 io1 \Omega\not=B M1 io2 \Omega ))
                    \wedgem<LB M2 M1 vs xs (?TS ((|M2 * m)) \cupV)S1 \Omega V''))))
    proof
        fix io assume io\inLin M1 {seq}
        then have io \inL M1
        by auto
        moreover have is-det-state-cover M2 V
            using assms(4) by auto
            ultimately obtain }\mp@subsup{V}{}{\prime\prime}\mathrm{ where }\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V
                using N-nonempty[OF - assms(1-3), of V io] by blast
            have io \inL M2
                using <io\inLin M1 {seq}> False by auto
```

            have \(V^{\prime \prime} \in \operatorname{Perm} V M 1\)
            using \(\left\langle V^{\prime \prime} \in N\right.\) io \(\left.M 1 V\right\rangle\) by auto
            have []\(\in V^{\prime \prime}\)
                using \(\left\langle V^{\prime \prime} \in \operatorname{Perm} V\right.\) M1 \(\rangle\) assms(4) perm-empty by blast
            have finite \(V^{\prime \prime}\)
                using \(\left\langle V^{\prime \prime} \in \operatorname{Perm} V\right.\) M1 \(\operatorname{assms}^{(2)} \operatorname{assms}(4)\) perm-elem-finite by blast
            obtain vs where mcp io \(V^{\prime \prime}\) vs
    using mcp－ex［OF $\left\langle[] \in V^{\prime \prime}\right\rangle\left\langle\right.$ finite $\left.\left.V^{\prime \prime}\right\rangle\right]$ by blast
obtain $x s$ where $i o=(v s @ x s)$
using «mcp io $V^{\prime \prime}$ vs〉 prefixE by auto
then have $v s @ x s \in L M 1 v s @ x s \in L M 2$
using 〈io $\in L$ M1〉〈io $\in L$ M2 $\rangle$ by auto
have io $\in L$ M1 map fst io $\in\{s e q\}$
using 〈io $\in L_{\text {in }}$ M1 \｛seq\}〉 by auto
then have map fst io $=$ seq by auto
then have map fst io $\in$ ？$C$ ？i using $\langle s e q \in$ ？$C$ ？$i\rangle$ by blast
then have（map fst vs）＠（map fst xs）$\in$ ？$C$ ？i using $\langle i o=(v s @ x s)\rangle$ by（metis map－append）
have $m c p^{\prime}$ io $V^{\prime \prime}=v s$
using «mcp io $V^{\prime \prime}$ vs〉 $m c p^{\prime}$－intro by blast
have mcp＇（map fst io）$V=($ map fst vs） using $\left\langle V^{\prime \prime} \in N\right.$ io $\left.M 1 V\right\rangle\left\langle m c p^{\prime}\right.$ io $\left.V^{\prime \prime}=v s\right\rangle$ by auto
then have map（map fst io）$V$（map fst vs）
by（metis « $\bigwedge$ thesis．（ $\bigwedge v s$. mcp seq $V v s \Longrightarrow$ thesis）$\Longrightarrow$ thesis
$\langle$ map fst io $=s e q\rangle m c p^{\prime}$－intro）
then have mcp（map fst vs＠map fst xs）$V$（map fst vs） by（simp add：＜io＝vs＠xs〉）
then have Suc（length xs）＝？i using C－index［OF $\langle($ map fst vs）＠（map fst xs）$\in$ ？$C$ ？$i\rangle]$ by $\operatorname{simp}$
then have $(|M 2| * m) \leq$ length $x s$ by $\operatorname{simp}$

```
have \(|M 1| \leq m\)
    using assms(3) by auto
have vs @ \(x s \in L M 2 \cap L M 1\)
    using <vs @ xs \(\in L\) M1〉〈vs @ \(x s \in L\) M2 \(\rangle\) by blast
obtain \(q\) where \(q \in\) nodes M2 \(m<\operatorname{card}\) ( \(R P\) M2 \(q\) vs xs \(V^{\prime \prime}\) )
    using \(R P\)-state-repetition-distribution-productF
        \([\) OF assms \((2,1)\langle(|M 2| * m) \leq\) length \(x s\rangle\langle | M 1|\leq m\rangle\langle v s @ x s \in L M 2 \cap L M 1\rangle\)
            \(\langle i s\)-det-state-cover M2 \(V\rangle\left\langle V^{\prime \prime} \in \operatorname{Perm} V\right.\) M1 \(\left.\rangle\right]\)
    by blast
```

have $m<L B$ M2 M1 vs xs $(? T S((|M 2| * m)) \cup V)\{q\} \Omega V^{\prime \prime}$
proof -
have $m<\left(\operatorname{sum}\left(\lambda s . \operatorname{card}\left(R P M 2 s\right.\right.\right.$ vs xs $\left.\left.\left.V^{\prime \prime}\right)\right)\{q\}\right)$
using $\left\langle m<\operatorname{card}\left(R P\right.\right.$ M2 $q$ vs xs $\left.V^{\prime \prime}\right)$ )
by auto
moreover have (sum ( $\lambda s$. card ( $R P$ M2 s vs xs $\left.V^{\prime \prime}\right)$ ) $\{q\}$ )
$\leq L B$ M2 M1 vs xs $(? T S((|M 2| * m)) \cup V)\{q\} \Omega V^{\prime \prime}$
by auto
ultimately show ?thesis
by linarith
qed
show $\exists V^{\prime \prime} \in N$ io M1 $V$.
$\exists$ S1 vs xs.
$i o=v s @ x s \wedge$
$m c p(v s @ x s) V^{\prime \prime} v s \wedge$

```
            S1\subseteq nodes M2 ^
            ( }\foralls1\inS1
                    \foralls2\inS1.
                    s1 = s2 \longrightarrow
                    (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}.\forallio2\inRP M2 s2 vs xs V V'. 
                                    B M1 io1 \Omega\not=B M1 io2 \Omega))^
                m<LB M2 M1 vs xs (?TS (( |M2|*m))\cupV)S1 \Omega V'
    proof -
        have io = vs@xs
            using <io = vs@xs` by assumption
    moreover have mcp(vs@xs) V'vs
            using «io = vs @ xs`<mcp io V V'vs` by presburger
    moreover have {q}\subseteq nodes M2
            using <q\in nodes M2` by auto
    moreover have ( }\foralls1\in{q}.\foralls2\in{q} 
                s1 = s2 \longrightarrow
                    (}\forall\mathrm{ io1 }\inRP M2 s1 vs xs V''.
                    \forall io2 \in RP M2 s2 vs xs V"'.
                            B M1 io1 \Omega}=\mathrm{ B M1 io2 }\Omega\mathrm{ ))
    proof -
        have }\foralls1\in{q}.\foralls2\in{q}.s1=s
            by blast
        then show ?thesis
            by blast
    qed
    ultimately have RM-body:io = (vs@xs)
                    ^mcp(vs@xs) V'lvs
                    \wedge {q}\subseteq nodes M2
                    \wedge }\forall\textrm{s}1\in{q}.\foralls2\in{q}
                    s1\not=s2\longrightarrow
                    (}\forall\mathrm{ io1 }\inRP M2 s1 vs xs V'".
                    \forallo2 \inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime
                            B M1 io1 \Omega\not=B M1 io2 \Omega ))
                \wedgem<LB M2 M1 vs xs (?TS ((|M2|*m))\cupV){q} \Omega V''
        using <m<LB M2 M1 vs xs (?TS (( |M2| * m))\cupV) {q} \Omega V'>
        by linarith
        show ?thesis
        using 〈\mp@subsup{V}{}{\prime\prime}\inN io M1 V> RM-body
        by metis
    qed
    qed
    then show ?thesis
    by metis
qed
then have seq }\in{x\mp@subsup{s}{}{\prime}\inCM2 M1 \OmegaVm((Suc (|M2|*m)))
    \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
    ( }\forallio\in\mp@subsup{L}{in}{\primen}M1{xs'}
        \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
            \existsS1 vs xs.
                    io = vs @ xs^
                    mcp (vs@ @s) V'v}vs
                S1 \subseteq nodes M2 ^
                    ( }\foralls1\inS1\mathrm{ .
                        \foralls }\inS\1
                            s1 = s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs V}\mp@subsup{V}{}{\prime\prime}.\forallio2\inRP M2 s2 vs xs V V'.
                                    B M1 io1 \Omega\not= B M1 io2 \Omega))^
                                    m<LB M2 M1 vs xs (?TS ((|M2|*m))\cupV)S1\Omega V'|)}
    using <seq \in?C ?i` by blast
```

```
        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 <?C ? i - ?RM ?i = {}>
```

    then have ?C (Suc ?i)={} using <?C ? i - ?RM ?i = {}>
    by blast
    by blast
    then have ?TS (Suc ?i) = ?TS ?i
    then have ?TS (Suc ?i) = ?TS ?i
    using TS.simps(3) by blast
    using TS.simps(3) by blast
    then show final-iteration M2 M1 \Omega Vm?i
    then show final-iteration M2 M1 \Omega Vm?i
    by blast
    by blast
    qed

```
qed
```

lemma TS-non-containment-causes-final :
assumes $v s @ x s \notin T S M 2 M 1 \Omega V m i$
and $m c p$ (vs@xs) V vs
and set $x s \subseteq$ inputs M2
and final-iteration M2 M1 $\Omega V m i$
and OFSM M2
shows $(\exists x r j . x r \neq x s$
$\wedge$ prefix xr xs
$\wedge j \leq i$
$\wedge v s @ x r \in R M M 2 M 1 \Omega V m j)$
proof -
let ? $T S=\lambda n . T S M 2 M 1 \Omega V m n$
let ? $C=\lambda n . C M 2 M 1 \Omega V m n$
let $? R M=\lambda n . R M M 2 M 1 \Omega V m n$
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: $(\exists x r j$. xr $\neq x s \wedge$ prefix $x r$ xs $\wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j) \vee$
$(\exists x c . x c \neq x s \wedge$ prefix $x c x s \wedge v s @ x c \in C$ M2 M1 $\Omega V m i-R M M 2 M 1 \Omega V m i)$ using $T S$-non-containment-causes $(1)[O F \operatorname{assms}(1-3)<0<i\rangle]$ by assumption
have ncc2 : $\neg((\exists x r j$. xr $\neq x s \wedge$ prefix xr xs $\wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j) \wedge$ $(\exists x c . x c \neq x s \wedge$ prefix xc xs $\wedge v s @ x c \in C$ M2 M1 $\Omega V m i-R M M 2 M 1 \Omega V m i))$ using TS-non-containment-causes(2)[OF assms(1-3) <0<i〉] by assumption
from ncc1 show ?thesis
proof
show $\exists x r j$. xr $\neq x s \wedge$ prefix xr $x s \wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j \Longrightarrow$
$\exists x r j$. xr $\neq x s \wedge$ prefix xr xs $\wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j$
by $\operatorname{simp}$
show $\exists x c . x c \neq x s \wedge$ prefix $x c x s \wedge v s @ x c \in C$ M2 M1 $\Omega V m i-R M M 2 M 1 \Omega V m i \Longrightarrow$
$\exists x r j$. xr $\neq x s \wedge$ prefix xr xs $\wedge j \leq i \wedge v s @ x r \in R M M 2 M 1 \Omega V m j$
proof -
assume $\exists x c . x c \neq x s \wedge$ prefix $x c x s \wedge v s @ x c \in C$ M2 M1 $\Omega V m i-R M M 2 M 1 \Omega V m i$
then obtain $x c$ where $x c \neq x s$ prefix xc xs vs @ $x c \in$ ? $C i-$ ? $R M i$
by blast

```
then have vs @ \(x c \in ? C i\)
    by blast
have mcp (vs @ xc) Vvs
    using <prefix xc xs〉 assms(2) mcp-prefix-of-suffix by blast
then have Suc (length \(x c\) ) \(=i\) using \(C\)-index \([O F\langle v s @ x c \in ? C i\rangle]\)
    by \(\operatorname{simp}\)
have length \(x c<\) length \(x s\)
    by (metis \(\langle p r e f i x\) xc \(x s\rangle\langle x c \neq x s\rangle\) 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 «prefix xc xs〉 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 $v s @ x c @[x] \notin ? C($ Suc $i)$
proof
assume vs @ xc @ $[x] \in ? C$ (Suc $i)$
then have vs @ $x c$ @ $[x] \notin ? T S i$
by (metis (no-types, lifting) C.simps(3) Diffe $\langle$ Suc (length $x c$ ) $=i\rangle$ )
then have ?TS $i \neq$ ?TS (Suc i)
using C-subset $\langle v s$ @ $x c$ @ $[x] \in C$ 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) 〈Suc (length $x c$ ) $=i\rangle$ by blast
ultimately have vs @ xc@ $[x] \notin$ append-set (?C $i-? R M$ i) (inputs M2) - ?TS $i$
by blast
have vs @ xc @ $x \mathrm{x}] \notin$ ?TS (Suc i)
by (metis Suc-n-not-le-n TS-index(1) 〈Suc (length xc) $=i\rangle$
$\langle p r e f i x(v s @ x c @[x])(v s @ x s)\rangle \operatorname{assms}(2)$ assms(4) length-append-singleton
mcp-prefix-of-suffix same-prefix-prefix)
then have vs @ $x c$ @ $[x] \notin ? T S i$
by (simp add: assms(4))
have vs @ xc @ $[x] \notin$ append-set (?C $i-$ ?RM i) (inputs M2)
using $\langle v s$ @ $x c @[x] \notin T S$ M2 M1 $\Omega V m i\rangle$
$\langle v s @ x c @[x] \notin$ append-set (C M2 M1 $\Omega V m i-R M M 2 M 1 \Omega V m i)($ inputs M2)
- TS M2 M1 $\Omega$ Vmi>
by blast
then have $v s @ x c \notin(? C i-? R M i)$
proof -
have $11: \forall a A$ Aa. $\left(a::^{\prime} a\right) \notin A \wedge a \notin A a \vee a \in A a \cup A$
by (meson UnCI)
obtain aas :: 'a list $\Rightarrow$ 'a list $\Rightarrow$ 'a list where
$\forall x 0 x 1$. $(\exists v 2 . x 0=x 1$ @ v2 $)=(x 0=x 1$ @ aas $x 0 x 1)$
by moura
then have vs @ $x s=(v s @ x c @[x])$ @ aas (vs @ xs) (vs @ $x c$ @ $[x]$ )
by (meson 〈prefix (vs @ xc @ [x]) (vs @ xs)〉prefixE)
then have $x s=(x c @[x])$ @ aas (vs@ xs) (vs @ xc@ $[x]$ )
by $\operatorname{simp}$
then have $x \in$ inputs M2
using $f 1$ by (metis (no-types) assms(3) contra-subsetD insert-iff list.set(2) set-append)
then show?thesis
using <vs @ xc @ $[x] \notin$ append-set (C M2 M1 $\Omega V m i-R M$ M2 M1 $\Omega V m i)($ inputs M2)
by force
qed

```
        then have False
            using «vs @ xc\in ?C i - ?RM i> by blast
            then show ?thesis by simp
        qed
    qed
qed
```

```
lemma TS-non-containment-causes-final-suc :
    assumes vs@xs &TS M2 M1 \OmegaVmi
    and mcp(vs@xs)V vs
    and set xs\subseteq inputs M2
    and final-iteration M2 M1 \Omega Vmi
    and OFSMM2
obtains xr j
where xr \not= xs prefix xr xs Suc j\leqivs@xr \inRM M2 M1 \Omega Vm (Suc j)
proof -
    obtain xr j where xr f=xs^ prefix xr xs ^j\leqi^vs@xr \inRM M2 M1 \OmegaVmj
        using TS-non-containment-causes-final[OF assms] by blast
    moreover have RM M2 M1 \Omega Vm0={}
        by auto
    ultimately have j\not=0
        by (metis empty-iff)
    then obtain jp where j=Suc jp
        using not0-implies-Suc by blast
    then have xr f xs ^ prefix xr xs ^Suc jp \leqi^vs@xr \inRM M2 M1 \Omega Vm (Suc jp)
        using <xr \not= xs ^ prefix xr xs ^j\leqi^vs@xr \inRM M2 M1 \Omega V m j`
        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}\quad\mp@subsup{V}{}{\prime\prime}\inN(vs@xs')M1
    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 = xs2
```

```
io-targets M2 (initial M2) (vs@ xs1) ={s2}
io-targets M2 (initial M2) (vs@ xs2) ={s\mathcal{Z}
io-targets M1 (initial M1) (vs@ xs1) ={s1}
io-targets M1 (initial M1) (vs@ xs2) ={s1}
```

by auto
then have $s 2 \in$ io-targets M2 (initial M2) (vs @ xs1)
$s 2 \in$ io-targets M2 (initial M2) (vs @ xs2)
$s 1 \in$ io-targets M1 (initial M1) (vs @ xs1)
$s 1 \in$ io-targets M1 (initial M1) (vs @ xs2)
by auto
have $v s @ x s 1 \in L$ M1
using io－target－implies－$L[O F<s 1 \in$ io－targets M1（initial M1）（vs＠xs1）$\rangle]$ by assumption have $v s @ x s 2 \in L M 1$
using io－target－implies－L［OF $\langle s 1 \in$ io－targets M1（initial M1）（vs＠xs2）$\rangle$ ］by assumption
have $v s @ x s 1 \in L M 2$
using io－target－implies－L［OF＜s2 $\in$ io－targets M2（initial M2）（vs＠xs1）$\rangle]$ by assumption
have $v s @ x s 2 \in L M 2$
using io－target－implies－$L[O F\langle s \mathcal{Z} \in$ io－targets M2（initial M2）（vs＠xs2）$)]$ by assumption
obtain tr1－1 where path M1（vs＠xs1｜｜tr1－1）（initial M1）
length tr1－1＝length（vs＠xs1）
target $(v s @ x s 1|\mid t r 1-1)($ initial M1）＝s1
using $\langle s 1 \in$ io－targets M1（initial M1）（vs＠xs1）＞by auto
obtain tr1－2 where path M1（vs＠xs2｜｜tr1－2）（initial M1）
length tr1－2＝length（vs＠xs2）
target（vs＠xs2｜｜tr1－2）（initial M1）＝s1
using $\langle s 1 \in$ io－targets M1（initial M1）（vs＠xs2）〉by auto
obtain tr2－1 where path M2（vs＠xs1｜｜tr2－1）（initial M2）
length tr2－1＝length（vs＠xs1）
target（vs＠xs1 \｜tr2－1）（initial M2）＝s2
using $\langle s 2 \in$ io－targets M2（initial M2）（vs＠xs1）＞by auto
obtain tr2－2 where path M2（vs＠xs2｜｜tr2－2）（initial M2）
length tr2－2＝length（vs＠xs2）
target $(v s @ x s 2 \| t r 2-2)($ initial M2）$=s 2$
using $\langle s 2 \in$ io－targets M2（initial M2）（vs＠xs2）＞by auto
have productF M2 M1 FAIL PM
using assms（4）by auto
have well－formed M1
using assms（2）by auto
have well－formed M2
using assms（3）by auto
have observable PM
by（meson assms（2）assms（3）assms（4）observable－productF）
have length $(v s @ x s 1)=$ length tr2－1
using＜length tr2－1＝length（vs＠xs1）〉by presburger
then have length tr2－1＝length tr1－1
using＜length tr1－1＝length（vs＠xs1）〉by presburger
have $v s @ x s 1 \in L P M$
using productF－path－inclusion $[O F 〈 l e n g t h ~(v s @ x s 1)=$ length tr2－1〉〈length tr2－1 $=$ length tr1－1〉
〈productF M2 M1 FAIL PM〉〈well－formed M2〉〈well－formed M1〉］
by（meson Int－iff 〈productF M2 M1 FAIL PM〉〈vs＠xs1 $\in L M 1\rangle\langle v s @ x s 1 \in L M 2\rangle\langle w e l l-f o r m e d ~ M 1\rangle$〈well－formed M2〉 productF－language）
have length（vs＠xs2）＝length tr2－2
using＜length tr2－2＝length（vs＠xs2）＞by presburger
then have length tr2－2 $=$ length tr1－2
using＜length tr1－2＝length（vs＠xs2）〉by presburger
have $v s @ x s 2 \in L P M$
using productF－path－inclusion $[O F 〈 l e n g t h ~(v s @ x s 2)=$ length tr2－2〉〈length tr2－2 $=$ length tr1－2〉

〈productF M2 M1 FAIL PM〉〈well－formed M2〉〈well－formed M1〉］
by（meson Int－iff 〈productF M2 M1 FAIL PM〉〈vs＠xs2 $\in L$ M1〉〈vs＠xs2 $\in L$ M2〉〈well－formed M1〉〈well－formed M2〉 productF－language）

```
have io-targets PM (initial M2, initial M1) (vs @ xs1) = {(s2, s1)}
    using productF-path-io-targets-reverse
        [OF <productF M2 M1 FAIL PM><s2 \in io-targets M2 (initial M2) (vs @ xs1)>
            <s1\in io-targets M1 (initial M1) (vs @ xs1)\rangle\langlevs@ xs1\inL M2\rangle\langlevs@ xs1\inL M1>]
proof -
    have }\forallcf.c\not=\operatorname{initial (f::('a,'b, 'c) FSM)\veec\in nodes f
        by blast
    then show ?thesis
        by (metis (no-types) «\llbracketobservable M2; observable M1; well-formed M2; well-formed M1;
                        initial M2 \in nodes M2; initial M1 \in nodes M1]
                                    Co-targets PM (initial M2, initial M1) (vs @ xs1) ={(s2, s1)}>
            assms(2) assms(3))
qed
have io-targets PM(initial M2, initial M1) (vs @ xs2) ={(s2, s1)}
    using productF-path-io-targets-reverse
            [OF<productF M2 M1 FAIL PM><s2 \in io-targets M2 (initial M2) (vs @ xs2)>
            <s1 \in io-targets M1 (initial M1) (vs @ xs2)>\langlevs @ xs2 \inL M2\rangle\langlevs @ xs2 \inL M1>]
proof -
    have }\forallcf.c\not=\mathrm{ initial ( }f::('a,'b,'c) FSM)\veec\in nodes 
        by blast
    then show ?thesis
        by (metis (no-types) «\llbracketobservable M2; observable M1; well-formed M2; well-formed M1;
                        initial M2 \in nodes M2; initial M1 \in nodes M1】
                        \Longrightarrowio-targets PM(initial M2, initial M1) (vs @ xs2) ={(s2, s1)}>
            assms(2) assms(3))
qed
have prefix (vs @ xs1) (vs @ xs2)
    using <prefix xs1 xs2` by auto
```

have sequence-to-failure M1 M2 (vs@xs)
using assms(1) by auto
have prefix (vs@xs1) (vs@xs')
using 〈prefix xs1 xs2〉〈prefix xs2 xs'〉 prefix-order.dual-order.trans same-prefix-prefix
by blast
have prefix (vs@xs2) (vs@xs')
using 〈prefix xs2 xs'〉 prefix-order.dual-order.trans same-prefix-prefix by blast
have io-targets $P M($ initial $P M)(v s @ x s 1)=\{(s 2, s 1)\}$
using 〈io-targets $P M$ (initial M2, initial M1) (vs @ xs1) $=\{(s 2, s 1)\}$ 〉assms(4) by auto
have io-targets $P M($ initial $P M)(v s @ x s 2)=\{(s 2, s 1)\}$
using «io-targets PM (initial M2, initial M1) (vs @ xs2) $=\{(s 2, s 1)\}$ 〉assms(4) by auto
have (vs @ xs2) @ (drop (length xs2) xs) = vs@xs
by (metis «prefix xs2 xs'〉 append-eq-appendI append-eq-conv-conj assms(6) prefixE)
moreover have io-targets PM (initial PM) (vs@xs) =\{FAIL\}
using sequence-to-failure-reaches-FAIL-ob $[O F$ ssequence-to-failure M1 M2 (vs@xs)〉assms(2,3)
〈productF M2 M1 FAIL PM〉]
by assumption
ultimately have io-targets PM (initial PM) ((vs @ xs2) @ (drop (length xs2) xs)) =\{FAIL\}
by auto

```
have io-targets PM (s\mathcal{Q},s1) (drop (length xs\mathcal{O})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(2,3)]
        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 \in 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 \neg 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 $O F S M M 1$
and OFSM M2
and test-tools M2 M1 FAIL PM V $\Omega$
and $\quad V^{\prime \prime} \in N(v s @ x s R) M 1 V$
and prefix xsR xs
shows $\neg R e p-C o v ~ M 2 ~ M 1 ~ V^{\prime \prime}$ vs $x s R$
proof
assume Rep-Cov M2 M1 $V^{\prime \prime}$ vs $x s R$
then obtain $x s^{\prime} v s^{\prime}$ s2 s1 where $x s^{\prime} \neq[]$
prefix $x s^{\prime} x s R$

```
vs'\in V'
io-targets M2 (initial M2) (vs @ xs')={s2}
io-targets M2 (initial M2) (vs')={s2}
io-targets M1 (initial M1) (vs @ xs')={s1}
io-targets M1 (initial M1) (vs')={s1}
```

by auto
then have $s 2 \in$ io－targets M2（initial M2）（vs＠$x s^{\prime}$ ）
s2 $\in$ io－targets M2（initial M2）（vs＇）
$s 1 \in$ io－targets M1（initial M1）（vs＠xs＇）
$s 1 \in$ io－targets M1（initial M1）（vs＇）
by auto
have $v s @ x s^{\prime} \in L M 1$
using io－target－implies－$L\left[O F\left\langle s 1 \in\right.\right.$ io－targets M1（initial M1）（vs＠xs $\left.\left.\left.{ }^{\prime}\right)\right\rangle\right]$ by assumption
have $v s^{\prime} \in L M 1$
using io－target－implies－L［OF $\langle s 1 \in$ io－targets $M 1$（initial M1）（vs＇）〉］by assumption
have $v s @ x s^{\prime} \in L$ M2
using io－target－implies－$L\left[O F<s \mathcal{Z} \in\right.$ io－targets M2（initial M2）（vs＠xs＇）$\left.{ }^{\prime}\right]$ by assumption
have $v s^{\prime} \in L$ M2
using io－target－implies－L［OF $\langle$ s2 $\in$ io－targets M2（initial M2）（vs＇）$\rangle$ ］by assumption
obtain tr1－1 where path M1（vs＠xs＇\｜tr1－1）（initial M1） length tr1－1＝length（vs＠xs＇） target $\left(v s @ x s^{\prime}| | \operatorname{tr} 1-1\right)($ initial M1）$=s 1$
using $\langle s 1 \in$ io－targets M1（initial M1）（vs＠xs＇）$\rangle$ by auto
obtain tr1－2 where path M1（vs＇｜｜tr1－2）（initial M1） length tr1－2 $=$ length $\left(v s^{\prime}\right)$ target $\left(v s^{\prime} \| \operatorname{tr1-2}\right)($ initial M1）$=s 1$
using $\left\langle s 1 \in\right.$ io－targets M1（initial M1）$\left.\left(v s^{\prime}\right)\right\rangle$ by auto
obtain tr2－1 where path M2（vs＠xs＇\｜tr2－1）（initial M2）
length tr2－1 $=$ length $\left(v s @ x s^{\prime}\right)$
target（vs＠xs＇\｜tr2－1）（initial M2）＝s2
using $\langle s 2 \in$ io－targets M2（initial M2）（vs＠xs＇）＞by auto
obtain tr2－2 where path M2（vs＇\｜tr2－2）（initial M2） length tr2－2 $=$ length $\left(v s^{\prime}\right)$ target $\left(v s^{\prime} \| t r 2-2\right)($ initial M2）$=s 2$
using $\left\langle s 2 \in\right.$ io－targets M2（initial M2）$\left.\left(v s^{\prime}\right)\right\rangle$ by auto
have productF M2 M1 FAIL PM
using assms（4）by auto
have well－formed M1
using $\operatorname{assms}$（2）by auto
have well－formed M2
using assms（3）by auto
have observable PM
by（meson assms（2）assms（3）assms（4）observable－productF）
have length $\left(v s @ x s^{\prime}\right)=$ length tr2－1
using＜length tr2－1＝length（vs＠xs＇）＞by presburger
then have length tr2－1 $=$ length tr1－1
using＜length tr1－1＝length（vs＠xs＇）〉 by presburger
have $v s @ x s^{\prime} \in L P M$
using productF－path－inclusion $\left[O F 〈 l e n g t h ~\left(v s @ x s^{\prime}\right)=\right.$ length tr2－1〉〈length tr2－1 $=$ length tr1－1〉
〈productF M2 M1 FAIL PM〉〈well－formed M2〉〈well－formed M1〉］
by（meson Int－iff〈productF M2 M1 FAIL PM〉〈vs＠$\left.x s^{\prime} \in L M 1\right\rangle\left\langle v s @ x s^{\prime} \in L M 2\right\rangle\langle w e l l-f o r m e d$ M1〉〈well－formed M2〉 productF－language）
have length $\left(v s^{\prime}\right)=$ length tr2－2
using＜length tr2－2＝length（vs＇）〉 by presburger
then have length tr2－2 $=$ length tr1－2
using＜length tr1－2＝length $\left(v s^{\prime}\right)$ 〉 by presburger

```
have vs'\inLPM
    using productF-path-inclusion[OF <length (vs') = length tr2-2`<length tr2-2 = length tr1-2>
                                    <productF M2 M1 FAIL PM〉 <well-formed M2` <well-formed M1>]
    by (meson Int-iff <productF M2 M1 FAIL PM`\langlevs' \inL M1>\langlevs' \inL M2\rangle〈well-formed M1>
        <well-formed M2` productF-language)
```

```
have io-targets PM (initial M2, initial M1) (vs @ xs \(\left.{ }^{\prime}\right)=\{(s 2, s 1)\}\)
    using productF-path-io-targets-reverse
        [OF \(\langle\) productF M2 M1 FAIL PM〉〈s2 \(\in\) io-targets M2 (initial M2) (vs @ xs') \(\rangle\)
            \(\left\langle s 1 \in\right.\) io-targets M1 (initial M1) (vs @ \(\left.\left.\left.x s^{\prime}\right)\right\rangle\left\langle v s @ x s^{\prime} \in L M 2\right\rangle\left\langle v s @ x s^{\prime} \in L M 1\right\rangle\right]\)
proof -
    have \(\forall c f . c \neq\) initial \(\left(f::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) F S M\right) \vee c \in\) nodes \(f\)
        by blast
    then show ?thesis
        by (metis (no-types) «【observable M2; observable M1; well-formed M2; well-formed M1;
                                    initial M2 \(\in\) nodes M2; initial M1 \(\in\) nodes M1】
                                    \(\Longrightarrow\) io-targets \(P M\left(\right.\) initial M2, initial M1) (vs @ \(\left.x s^{\prime}\right)=\{(s 2, s 1)\}>\)
            \(\operatorname{assms}(2) \operatorname{assms}(3))\)
qed
have io-targets \(P M\) (initial M2, initial M1) \(\left(v s^{\prime}\right)=\{(s 2, s 1)\}\)
    using productF-path-io-targets-reverse
            [OF 〈productF M2 M1 FAIL PM〉〈s2 \(\in\) io-targets M2 (initial M2) (vs') \(\rangle\)
                \(\left\langle s 1 \in\right.\) io-targets \(M 1\) (initial M1) \(\left.\left(v s^{\prime}\right)\right\rangle\left\langle v s^{\prime} \in L M 2\right\rangle\left\langle v s^{\prime} \in L\right.\) M1 \(\left.\rangle\right]\)
proof -
    have \(\forall c f . c \neq\) initial \(\left(f::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) F S M\right) \vee c \in \operatorname{nodes} f\)
        by blast
    then show ?thesis
        by (metis (no-types) «【observable M2; observable M1; well-formed M2; well-formed M1;
                        initial M2 \(\in\) nodes M2; initial M1 \(\in\) nodes M1]
                            \(\Longrightarrow\) io-targets \(P M\) (initial M2, initial M1) \(\left(v s^{\prime}\right)=\{(s 2, s 1)\}>\)
            \(\operatorname{assms}(2) \operatorname{assms}(3))\)
qed
have io-targets \(P M(\) initial \(P M)\left(v s^{\prime}\right)=\{(s 2, s 1)\}\)
    by (metis (no-types) 〈io-targets PM (initial M2, initial M1) vs \({ }^{\prime}=\{(s 2, s 1)\}>\)
        〈productF M2 M1 FAIL PM〉 productF-simps(4))
have sequence-to-failure M1 M2 (vs@xs)
    using assms(1) by auto
have \(x s=x s^{\prime}\) @ (drop (length \(\left.x s^{\prime}\right) x s\) )
    by (metis \(\left\langle p r e f i x ~ x s^{\prime}\right.\) xsR〉 append-assoc append-eq-conv-conj assms(6) prefixE)
then have io-targets PM (initial M2, initial M1) (vs @ \(x s^{\prime} @\left(\right.\) drop \(\left(\right.\) length \(\left.\left.\left.x s^{\prime}\right) x s\right)\right)=\{\) FAIL \(\}\)
    by (metis \(\langle\) productF M2 M1 FAIL PM〉 〈sequence-to-failure M1 M2 (vs @ xs)〉assms(2) assms(3)
        productF-simps(4) sequence-to-failure-reaches-FAIL-ob)
then have io-targets PM (initial M2, initial M1) ((vs @ xs') @ (drop (length xs') xs \()=\{\) FAIL \(\}\)
    by auto
have io-targets \(P M\left(s 2\right.\), s1) (drop (length \(\left.\left.x s^{\prime}\right) x s\right)=\{F A I L\}\)
    using observable-io-targets-split
        [OF «observable PM〉
                〈io-targets PM (initial M2,initial M1) ((vs @ xs') @ (drop (length xs') xs)) =\{FAIL\}>
                \(\left\langle\right.\) io-targets PM (initial M2, initial M1) \(\left.\left.\left(v s @ x s^{\prime}\right)=\{(s 2, s 1)\}\right\rangle\right]\)
    by assumption
have io-targets \(P M(\) initial \(P M)\left(v s^{\prime} @\left(\right.\right.\) drop \(\left(\right.\) length \(\left.\left.\left.x s^{\prime}\right) x s\right)\right)=\{F A I L\}\)
    using observable-io-targets-append
        [OF «observable PM〉〈io-targets PM (initial PM) \(\left.\left(v s^{\prime}\right)=\{(s 2, s 1)\}\right\rangle\)
            \(\left\langle\right.\) io-targets \(P M(s 2, s 1)\left(\right.\) drop (length \(\left.\left.x s^{\prime}\right) x s\right)=\{\) FAIL \(\left.\left.\}\right\rangle\right]\)
    by assumption
```

have sequence-to-failure M1 M2 (vs' @ (drop (length $\left.\left.x s^{\prime}\right) x s\right)$ )
using sequence-to-failure-alt-def
$\left[O F\left\langle i o-t a r g e t s ~ P M(\right.\right.$ initial $P M)\left(v s^{\prime} @\left(\operatorname{drop}\left(\right.\right.\right.$ length $\left.\left.\left.x s^{\prime}\right) x s\right)\right)=\{F A I L\}$ 〉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\rangle\langlex\mp@subsup{s}{}{\prime}\not=[]> length-append
        length-greater-0-conv less-add-same-cancel2)
    have vs'\in Lin M1 V
    proof -
    have }\mp@subsup{V}{}{\prime\prime}\inP\mathrm{ Perm V M1
        using assms(5) unfolding N.simps by blast
```

    then obtain \(f\) where \(f\)-def : \(V^{\prime \prime}=\) image \(f V\)
                                    \(\wedge(\forall v \in V . f v \in\) language-state-for-input M1 (initial M1) v)
        unfolding Perm.simps by blast
    then obtain \(v\) where \(v \in V v s^{\prime}=f v\)
        using \(\left\langle v s^{\prime} \in V^{\prime \prime}\right\rangle\) by auto
    then have \(v s^{\prime} \in\) language-state-for-input M1 (initial M1) v
        using \(f\)-def by auto
    have language-state-for-input M1 (initial M1) \(v=L_{i n} M 1\{v\}\)
        by auto
    moreover have \(\{v\} \subseteq V\)
        using \(\langle v \in V\rangle\) by blast
    ultimately have language-state-for-input M1 (initial M1) \(v \subseteq L_{i n} M 1 V\)
        unfolding language-state-for-inputs.simps language-state-for-input.simps by blast
    then show ?thesis
        using \(\left\langle v s^{\prime} \in\right.\) language-state-for-input M1 (initial M1) \(\left.v\right\rangle\) by blast
    qed
have $\neg$ minimal-sequence-to-failure-extending $V$ M1 M2 vs xs
using $\left\langle v s^{\prime} \in L_{i n} M 1 V\right\rangle$
«sequence-to-failure M1 M2 (vs' @ (drop (length $\left.\left.x s^{\prime}\right) x s\right)$ )〉
〈length (drop (length $x s^{\prime}$ ) xs) < length xs〉
using minimal-sequence-to-failure-extending.elims(2) by blast
then show False
using $\operatorname{assms}(1)$ 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 PMV \Omega
    and }\quad\mp@subsup{V}{}{\prime\prime}\inN(vs@x\mp@subsup{s}{}{\prime})M1
    and prefix xs' xs
shows \neg Rep-Pre M2 M1 vs xs'
    and }\neg\mathrm{ 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 Vmi
shows M1 \preceq\llbracket(TS M2 M1 \Omega Vmi). \Omega\rrbracketM2 \longrightarrowM1 \preceqM2
proof
    assume atc-io-reduction-on-sets M1 (TS M2 M1 \Omega Vmi) \Omega M2
    show M1 \preceq M2
    proof (rule ccontr)
    let ?TS = \lambdan. TS M2 M1 \Omega Vmn
let?C = \lambdan.C M2 M1 \Omega Vmn
let ?RM = \lambdan.RM M2 M1 \OmegaVmn
assume \negM1 \preceqM2
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 - -- \checkmarkᄀM1 \preceqM2`, of V]
    by blast
then have vs \in Lin M1 V
            sequence-to-failure M1 M2 (vs @ xs)
            \neg(\existsio'}.\exists\mp@subsup{w}{}{\prime}\in\mp@subsup{L}{in}{}M1V.sequence-to-failure M1 M2 ( w' @ io'
                        ^length io' < length xs)
    by auto
then have vs@xs\inL M1 - L M2
    by auto
have vs@xs\in Lin M1 {map fst (vs@xs)}
    by (metis (full-types) Diff-iff <vs @ xs \inL M1 - L M2` insertI1
        language-state-for-inputs-map-fst)
have vs@xs & Lin M2 {map fst (vs@xs)}
    by (meson Diff-iff <vs @ xs \inL M1 - L M2`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<atc-io-reduction-on-sets M1 (TS M2 M1 \Omega Vmi) \Omega M2`]
    by auto
have map fst (vs@xs)\not\in?TS i
proof -
    have f1: \forallps P Pa. (ps::(' }\mp@subsup{}{}{\prime}\times\mp@subsup{}{}{\prime}b) list) #P - Pa\vee ps\inP\wedge ps\not\inP
        by blast
    have }\forallPPa ps.\negP\subseteqPa\vee(ps::('a\times'b) list)\inPa\veeps\not\in
        by blast
    then show ?thesis
        using f1 by (metis (no-types) <vs @ xs \inL M1 - L M2`\io-reduction-on M1 (?TS i) M2>
                language-state-for-inputs-in-language-state language-state-for-inputs-map-fst)
qed
have map fst vs \inV
    using <vs \in Lin M1 V> by auto
let ?stf = map fst (vs@xs)
let ?stfV = map fst vs
let ?stfX = map fst xs
```

```
have ?stf = ?stfV @ ?stfX
    by simp
then have ?stfV @ ?stfX &?TS i
    using <?stf & ?TS i` by auto
have mcp (?stfV @ ?stfX) V ?stfV
    by (metis <map fst (vs @ xs)=map fst vs @ map fst xs>
        <minimal-sequence-to-failure-extending V M1 M2 vs xs` assms(1) assms(2) assms(4)
        minimal-sequence-to-failure-extending-mcp)
have set ?stf \subseteq inputs M1
    by (meson DiffD1 «vs @ xs \inL M1 - L M2`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 f ?stfX
                                    prefix xr ?stfX
            Suc j \leqi
            ?stfV@xr \in RM M2 M1 \Omega Vm (Suc j)
    using TS-non-containment-causes-final-suc[OF <?stfV @ ?stfX & ?TS i>
        <mcp (?stfV @ ?stfX) V ?stfV〉<set ?stfX \subseteq inputs M2`assms(5,2)]
    by blast
let ?yr = take (length xr) (map snd xs)
have length ?yr = length xr
    using <prefix xr (map fst xs)> prefix-length-le by fastforce
have (xr||yr) = take (length xr) xs
    by (metis (no-types, opaque-lifting) <prefix xr (map fst xs)〉 append-eq-conv-conj prefixE take-zip
        zip-map-fst-snd)
have prefix (vs@(xr || ?yr)) (vs@xs)
    by (simp add: <xr | take (length xr) (map snd xs) = take (length xr) xs` take-is-prefix)
have xr = take (length xr (map fst xs)
    by (metis «length (take (length xr) (map snd xs)) = length xr>
        <xr || take (length xr) (map snd xs) = take (length xr) xs` map-fst-zip take-map)
have vs@(xr ||yr)\inL M1
    by (metis DiffD1<prefix (vs @ (xr || take (length xr) (map snd xs))) (vs @ xs)>
        <vs @ xs \inL M1 - L M2` language-state-prefix prefixE)
then have vs@(xr || ?yr) \in Lin M1 {?stfV @ xr}
    by (metis «length (take (length xr) (map snd xs)) = length xr> insertI1
        language-state-for-inputs-map-fst map-append map-fst-zip)
have length xr < length xs
    by (metis «xr = take (length xr) (map fst xs)\rangle\langlexr \not= map fst xs\rangle not-le-imp-less take-all
        take-map)
from〈?stfV@xr <RM M2 M1 \OmegaVm(Suc j)> have ?stfV@xr \in{xs' \inC M2 M1 \Omega Vm(Suc j).
        (\neg(\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}\subseteq\mp@subsup{L}{in}{}M2{x\mp@subsup{s}{}{\prime}}))
        V (\forall io \in Lin M1 {xs'}.
            (\exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
            (\exists S1.
                    (\exists vs xs .
                io =(vs@xs)
                ^mcp(vs@xs) V' vs
```

```
\S1\subseteq nodes M2
\wedge (\forall s1\inS1.}\forall s2 \inS1
    s1\not=s2\longrightarrow
        (\forall io1 \in RP M2 s1 vs xs V''.
            \forallo2 \in RP M2 s2 vs xs V"'.
                B M1 io1 \Omega\not=B M1 io2 \Omega ))
\wedge m<LB M2 M1 vs xs (TS M2 M1 \Omega Vmj\cupV)S1\Omega V'))))}
    unfolding RM.simps by blast
```

```
moreover have }\forallx\mp@subsup{s}{}{\prime}\in?C(Suc j). . Lin M1 {xs'}\subseteq Lin M2 {x\mp@subsup{s}{}{\prime}
proof
    fix x\mp@subsup{s}{}{\prime}}\mathrm{ assume xs' }\in\mathrm{ ? C (Suc j)
    from 〈Suc j\leqi\rangle have ?C (Suc j)\subseteq?TS i
        using C-subset TS-subset by blast
    then have {xs'}\subseteq??TS i
        using <xs' \in? C (Suc j)> by blast
    show Lin M1 {xs'}\subseteq Lin M2 {xs'}
        using io-reduction-on-subset[OF <io-reduction-on M1 (?TS i) M2`<{xs'}\subseteq ?TS i`]
        by assumption
qed
ultimately have (\forall io \in Lin M1 {?stfV@xr}.
            (\exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V .
                (\exists S1 .
                    (\exists vs xs.
                io = (vs@xs)
                ^mcp(vs@xs) V'l vs
                ^S1\subseteq nodes M2
                \wedge (\forall s1 \inS1.}\forall \2 \inS1
                    s1\not=s2\longrightarrow
                        (\forall io1 \in RP M2 s1 vs xs V'".
                            io2 \inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}
                                    B M1 io1 \Omega\not= B M1 io2 \Omega ))
                \wedge m<LB M2 M1 vs xs (TS M2 M1 \Omega Vmj\cupV)S1\Omega V'))))
    by blast
```

then have
$\left(\exists V^{\prime \prime} \in N(v s @(x r \|\right.$ ?yr $)) M 1 V$.
( $\exists$ S1.
( $\exists v s^{\prime} x s^{\prime}$.
$v s @\left(x r|\mid\right.$ ? $y r)=\left(v s^{\prime} @ x s^{\prime}\right)$
$\wedge m c p\left(v s^{\prime} @ x s^{\prime}\right) V^{\prime \prime} v s^{\prime}$
$\wedge S 1 \subseteq$ nodes M2
$\wedge(\forall s 1 \in S 1 . \forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
$\left(\forall\right.$ io1 $\in R P M 2$ s1 $v s^{\prime} x s^{\prime} V^{\prime \prime}$.
$\forall i o 2 \in R P$ M2 s2 $v s^{\prime} x s^{\prime} V^{\prime \prime}$.
B M1 io1 $\Omega \neq B$ M1 io2 $\Omega$ ))
$\left.\left.\left.\wedge m<L B M 2 M 1 v s^{\prime} x s^{\prime}(T S M 2 M 1 \Omega V m j \cup V) S 1 \Omega V^{\prime \prime}\right)\right)\right)$
using $\left\langle v s @(x r \|\right.$ ? $y r) \in L_{i n} M 1$ ? ?stfV @ $\left.x r\right\}$ 〉
by blast
then obtain $V^{\prime \prime} S 1 v s^{\prime} x s^{\prime}$ where $R M$-impl:
$V^{\prime \prime} \in N(v s @(x r \|$ ? $y r)) M 1 V$
$v s @(x r \|$ ? $y r)=\left(v s^{\prime} @ x s^{\prime}\right)$
$m c p\left(v s^{\prime} @ x s^{\prime}\right) V^{\prime \prime} v s^{\prime}$
S1 $\subseteq$ nodes M2
$(\forall s 1 \in S 1 . \forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
$\left(\forall\right.$ io1 $\in R P M 2$ s1 $v s^{\prime} x s^{\prime} V^{\prime \prime}$.
$\forall i o 2 \in R P M 2$ s2 $v^{\prime} x s^{\prime} V^{\prime \prime}$.
B M1 io1 $\Omega \neq B$ M1 io2 $\Omega)$ )
$m<L B M 2 M 1 v s^{\prime} x s^{\prime}(T S M 2 M 1 \Omega V m j \cup V) S 1 \Omega V^{\prime \prime}$
by blast

```
have ?stfV = mcp' (map fst (vs @ (xr | take (length xr) (map snd xs)))) V
    by (metis (full-types)<length (take (length xr) (map snd xs)) = length xr>
        <mcp (map fst vs @ map fst xs) V (map fst vs)〉<prefix xr (map fst xs)> 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\mp@subsup{V}{}{\prime\prime
            vs=mcp'}(vs@(xr|take (length xr) (map snd xs))) V ''
    using N-mcp-prefix[OF〈?stfV = mcp' (map fst (vs @ (xr | take (length xr) (map snd xs)))) V`
        \langleV'|}\inN(vs@(xr||yr))M1 V\rangle,of M2]
    by simp+
have vs'=vs
    by (metis (no-types) <mcp (vs'@ @s') V '| vs'>
        <vs =mcp' (vs @ (xr| take (length xr) (map snd xs))) V'>
        <vs@ (xr | take (length xr) (map snd xs))=v\mp@subsup{s}{}{\prime}@ x\mp@subsup{s}{}{\prime}> mcp'-intro)
```

then have $x s^{\prime}=(x r \|$ ? $y r)$
using $<v s$ @ (xr \|take (length xr) (map snd $x s)$ ) $\left.=v s^{\prime} @ x s^{\prime}\right\rangle$ by blast
have $V \subseteq$ ?TS $i$
proof -
have $1 \leq i$
using $\langle S u c j \leq i\rangle$ 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 <?stfV@xr $\in$ RM M2 M1 $\Omega V m$ (Suc j)〉unfolding RM.simps by blast
－show that the prerequisites（Prereq）for LB are met by construction

```
have (\forallv\mp@subsup{s}{}{\prime}a\in\mp@subsup{V}{}{\prime\prime}.prefix v\mp@subsup{s}{}{\prime}a(v\mp@subsup{s}{}{\prime}@ @s\mp@subsup{s}{}{\prime})\longrightarrowlength v\mp@subsup{s}{}{\prime}a\leqlength vs')
    using <mcp (vs'@ @s') V' vs'> by auto
moreover have atc-io-reduction-on-sets M1 (?TS j\cupV) \Omega M2
proof -
    have j<i
        using <Suc j\leqi` by auto
    then have ?TS j\subseteq?TS i
        by (simp add: TS-subset)
    then show ?thesis
        using atc-io-reduction-on-subset
                [OF<atc-io-reduction-on-sets M1 (TS M2 M1 \Omega Vm i) \Omega M2`, of ?TS j]
        by (meson Un-subset-iff <V\subseteq?TS i〉 <atc-io-reduction-on-sets M1 (TS M2 M1 \Omega V mi) \Omega M2>
            atc-io-reduction-on-subset)
qed
moreover have finite (?TS j\cupV)
proof -
    have finite (?TS j)
        using TS-finite[OF〈{inite V〉, of M2 M1 \Omega mj] assms(2) by auto
    then show ?thesis
        using <finite V> by blast
qed
```

```
moreover have V\subseteq??TS j\cupV
```

    by blast
    ```
moreover have \(\left(\forall p .\left(\right.\right.\) prefix \(\left.\left.p x s^{\prime} \wedge p \neq x s^{\prime}\right) \longrightarrow \operatorname{map} f s t\left(v s^{\prime} @ p\right) \in ? T S j \cup V\right)\)
proof
    fix \(p\)
    show prefix \(p x s^{\prime} \wedge p \neq x s^{\prime} \longrightarrow\) map fst \(\left(v s^{\prime} @ p\right) \in T S M 2 M 1 \Omega V m j \cup V\)
    proof
        assume prefix \(p x s^{\prime} \wedge p \neq x s^{\prime}\)
        have prefix (map fst (vs' @ p)) (map fst (vs' @ xs'))
        by (simp add: «prefix \(p\) xs \(\left.s^{\prime} \wedge p \neq x s^{\prime}\right\rangle\) map-mono-prefix)
        have prefix (map fst (vs' @ p)) (?stfV @ xr)
            using 〈length (take (length xr) (map snd xs)) = length xr>
                    〈prefix (map fst (vs' @ p)) (map fst (vs' @ xs')) 〉
                    \(\left\langle v s^{\prime}=v s\right\rangle\left\langle x s^{\prime}=x r \|\right.\) take (length xr) (map snd \(\left.\left.x s\right)\right\rangle\)
        by auto
    then have prefix (map fst vs' @ map fst p) (?stfV @ xr)
        by \(\operatorname{simp}\)
        then have prefix (map fst p) xr
        by (simp add: \(\left.\left\langle v s^{\prime}=v s\right\rangle\right)\)
        have ?stfV @ \(x r \in\) ?TS (Suc j)
        proof (cases j)
            case 0
            then show ?thesis
            using <map fst vs @ xr \(\in C\) M2 M1 \(\Omega V m(S u c j)\) 〉by auto
        next
        case (Suc nat)
        then show?thesis
            using TS.simps(3)〈map fst vs @ xr \(\in\) CM2 M1 \(\Omega V m(S u c j)\rangle\) by blast
        qed
        have mcp (map fst vs @ xr) V (map fst vs)
        using «mcp (map fst vs @ map fst xs) V (map fst vs)〉〈prefix xr (map fst xs)〉
            mср-prefix-of-suffix
        by blast
        have map fst vs @ map fst \(p \in T S\) M2 M1 \(\Omega V m(S u c j)\)
        using TS-prefix-containment[OF 〈?stfV @ xr \(\in\) ?TS (Suc j) 〉
                        〈mcp (map fst vs @ xr) V (map fst vs) 〉
                        \(\langle p r e f i x(\) map fst \(p\) ) \(x r\rangle\) ]
        by assumption
```

        have Suc (length xr) \(=(\) Suc \(j)\)
        using C-index[OF «?stfV@xr \(\in\) ? \(C\) (Suc j)〉〈mcp (map fst vs @ xr) \(V(m a p f s t v s)\rangle]\)
        by assumption
    haveSuc (length \(p\) ) \(<(\) Suc \(j\) )
    proof -
        have map fst \(x s^{\prime}=x r\)
            by (metis \(\langle x r=\) take (length \(x r)(\) map fst \(x s)\rangle\)
            \(\langle x r| \mid\) take (length \(x r\) ) (map snd \(x s\) ) \(=\) take (length \(x r\) ) xs \(\rangle\)
            \(\left\langle x^{\prime}=x r \|\right.\) take (length \(x r\) ) (map snd xs) > take-map)
        then show ?thesis
            by (metis (no-types) Suc-less-eq \(\langle\) Suc (length xr) \(=\) Suc \(j\rangle\left\langle p r e f i x ~ p x s^{\prime} \wedge p \neq x s^{\prime}\right\rangle\)
            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 «mcp (map fst vs @ xr) V (map fst vs)〉〈prefix (map fst p) xr〉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 $p \in T S M 2 M 1 \Omega V m(S u c j)\rangle]$ by auto

```
    have map fst vs @ map fst p\in?TS j
    using TS-union[of M2 M1 \Omega Vm j]
    proof -
        have Suc (length p)\in{0..<Suc j}
        using <Suc (length p) < Suc j> by force
    then show ?thesis
        by (metis UN-I \TS M2 M1 \Omega Vmj = (\bigcupj\inset [0..<Suc j].C M2 M1 \Omega Vm j)`
            <map fst vs @ map fst p\inC M2 M1 \OmegaVm(Suc (length (map fst p)))>
            length-map set-upt)
    qed
    then show map fst (vs' @ p) \inTS M2 M1 \Omega Vmj\cupV
    by (simp add: <vs' = vs`)
    qed
qed
```

moreover have $v s^{\prime} @ x s^{\prime} \in L M 2 \cap L M 1$
by (metis (no-types, lifting) IntI RM-impl(2)
$\left\langle\forall x s^{\prime} \in C\right.$ M2 M1 $\left.\Omega V m(S u c j) . L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\}\right\rangle$
〈map fst vs @ xr $\in C$ M2 M1 $\Omega V m(S u c j)\rangle$
〈vs @ (xr \| take (length xr) (map snd xs)) $\in L_{\text {in }} M 1$ \{map fst vs @ xr\} $\rangle$
language-state-for-inputs-in-language-state subsetCE)
ultimately have Prereq M2 M1 vs $s^{\prime} s^{\prime}(? T S j \cup V) S 1 \Omega V^{\prime \prime}$
using RM-impl(4,5) unfolding Prereq.simps by blast
have $V^{\prime \prime} \in \operatorname{Perm} V$ M1
using $\left\langle V^{\prime \prime} \in N(v s @(x r \|\right.$ ? $\left.y r)) M 1 V\right\rangle$ unfolding $N . s i m p s$ by blast
have «prefix (xr \| ? $y$ r) $x s$ 〉
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 \neg 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 \Omega> RM-impl(1)
            <prefix (xr || take (length xr) (map snd xs)) xs`]
```

    by assumption
    then have $\neg$ Rep-Pre M2 M1 vs' $x s^{\prime}$
using $\left\langle v s^{\prime}=v s\right\rangle\left\langle x s^{\prime}=x r \|\right.$ ? $\left.y r\right\rangle$ by blast
have $\neg$ Rep-Cov M2 M1 $V^{\prime \prime}$ vs (xr \| ? yr )
using minimal-sequence-to-failure-extending-implies-Rep-Cov
[OF «minimal-sequence-to-failure-extending V M1 M2 vs xs〉 $\operatorname{assms}(1,2)$
〈test-tools M2 M1 FAIL PM V $\Omega$ 〉 RM-impl(1)
$\langle p r e f i x$ ( $x$ \| \| take (length $x r$ ) (map snd $x s$ )) $x s\rangle$
by assumption
then have $\neg \operatorname{Rep}-\operatorname{Cov}$ M2 M1 $V^{\prime \prime} v s^{\prime} x s^{\prime}$
using $\left\langle v s^{\prime}=v s\right\rangle\left\langle x s^{\prime}=x r \|\right.$ ? $\left.y r\right\rangle$ by blast
have $v s^{\prime} @ x s^{\prime} \in L M 1$
using $\langle v s$ @ (xr \|take (length xr) (map snd xs)) $\in L$ M1 $\rangle$
$\left\langle v s^{\prime}=v s\right\rangle\left\langle x s^{\prime}=x r \|\right.$ take (length xr) (map snd xs) $\rangle$
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\cupV) S1 \Omega V''
    using LB-count[OF <vs'@xs' \inL M1〉 assms(1,2,3)<test-tools M2 M1 FAIL PM V \Omega>
```



```
                    {\neg Rep-Pre M2 M1 vs' xs'\rangle<\neg Rep-Cov M2 M1 V''vs' xs'\rangle]
    by assumption
    then have LB M2 M1 vs' xs' (?TS j\cupV)S1 \Omega V' }\leq
    using assms(3) by linarith
    then show False
    using <m<LB M2 M1 vs' xs'(?TS j\cupV)S1 \Omega V'>}>\mathrm{ 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 \longrightarrowatc-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 PMV \Omega
    and final-iteration M2 M1 \Omega Vmi
shows M1 \preceq M2 \longleftrightarrow atc-io-reduction-on-sets M1 (TS M2 M1 \Omega Vmi) \Omega M2
by (metis asc-sufficiency assms(1-5) atc-io-reduction-on-sets-reduction)
```


## end

theory $A S C$-Hoare
imports ASC-Sufficiency HOL-Hoare.Hoare-Logic
begin

## 7 Correctness of the Adaptive State Counting Algorithm in HoareLogic

In this section we give an example implementation of the adaptive state counting algorithm in a simple WHILElanguage 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 \(L_{i n} M 1 T \subseteq L_{i n} M 2 T\)
    and \(\left(\bigcup i o \in L_{i n} M 1 T .\{i o\} \times B M 1\right.\) io \(\left.\Omega\right) \subseteq\left(\bigcup i o \in L_{i n} M 2 T .\{i o\} \times B\right.\) M2 io \(\left.\Omega\right)\)
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 \(i s e q \in T\)
    have \(L_{i n} M 1\{i s e q\} \subseteq L_{i n} M 2\{i s e q\}\)
        by (metis \(\langle i s e q \in T 〉 a s s m s(1)\) bot.extremum insert-mono io-reduction-on-subset
            mk-disjoint-insert)
    moreover have \(\forall i o \in L_{i n} M 1\{i s e q\} . B M 1\) io \(\Omega \subseteq B\) M2 io \(\Omega\)
    proof
        fix io assume io \(\in L_{i n} M 1\{i s e q\}\)
        then have \(i o \in L_{i n}\) M2 \(\{i s e q\}\)
```

```
    using calculation by blast
    show }BM1\mathrm{ io }\Omega\subseteqBM2 io 
    proof
    fix }x\mathrm{ assume }x\inBM1\mathrm{ io }
    have io \in Lin M1 T
        using <io \in Lin M1 {iseq}><iseq }\inT> by aut
    moreover have (io,x)\in{io}\timesBM1 io \Omega
        using <x \in B M1 io \Omega> by blast
    ultimately have (io,x)\in(\bigcupio\inLin M1 T. {io} }\times\mathrm{ ( B M1 io ת)
    by blast
    then have (io,x)\in(\bigcupio\inLin M2 T. {io} > B M2 io \Omega)
            using assms(2) by blast
    then have (io,x)\in{io}\timesB M2 io \Omega
        by blast
    then show }x\inB\mathrm{ M2 io }
        by blast
    qed
    qed
    ultimately show }\mp@subsup{L}{in}{}M1{iseq}\subseteq\mp@subsup{L}{in}{}M2{iseq
                        \wedge(\forallio\inLin M1 {iseq}. BM1 io \Omega\subseteqB 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 }\mp@subsup{L}{in}{}M1T\subseteq\mp@subsup{L}{in}{}M2
    and (\bigcupio\inL Lin M1 T. {io} }\times\mathrm{ (BM1 io ת) }\subseteq(\bigcupio\inLin M2 T. {io} 人 B M2 io \Omega
proof
    fix }x\mathrm{ assume }x\in\mp@subsup{L}{in}{}M1
    show }x\in\mp@subsup{L}{in}{}\mathrm{ M2 T
        using assms unfolding atc-io-reduction-on-sets.simps atc-io-reduction-on.simps
    proof -
        assume a1: \foralliseq\inT. Lin}M1{iseq}\subseteq Lin M2 {iseq
                        \wedge(\forallio\inL Lin M1 {iseq}. B M1 io \Omega\subseteqB M2 io \Omega)
        have f2: x U UNION T (language-state-for-input M1 (initial M1))
            by (metis (no-types) <x\in Lin M1 T> language-state-for-inputs-alt-def)
    obtain aas :: 'a list set }=>(\mp@subsup{}{}{\prime}a\mathrm{ list }=>('a\times'b) list set ) => ('a\times' 'b) list = 'a list
            where
            \forallx0 x1 x2. (\existsv3.v3 \inx0^x2 \inx1 v3) = (aas x0 x1 x2 \inx0 ^ x2 \in x1 (aas x0 x1 x2))
            by moura
            then have }\forallpsfA.(ps\not\inUNION Af\vee aas A f ps\inA\wedge ps\inf(aas Afps)
                        \wedge(ps\inUNION A f\vee (\forallas. as \not\inA\vee ps\not\infas))
            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 (\bigcupio\inLin M1 T. {io} }\times\mathrm{ (B M1 io }\Omega)\subseteq(\bigcupio\inLin M2 T. {io} > B M2 io \Omega
    proof
        fix iox assume iox }\in(\bigcup\mathrm{ \ofLLin M1 T. {io} }\times\mathrm{ B M1 io ת)
        then obtain io x where iox = (io,x)
            by blast
        have io }\in\mp@subsup{L}{in}{}M1
            using}\langleiox=(io,x)\rangle\langleiox\in(\bigcupio\inLin M1 T.{io}\timesB M1 io \Omega)> by blas
            have (io,x)\in{io} }\times\mathrm{ B M1 io }
            using <iox = (io, x)\rangle\langleiox \in(\bigcupio\inLin M1 T.{io} ×B M1 io \Omega)> by blast
            then have }x\inBM1\mathrm{ io }
                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<io \in Lin M1 T> language-state-for-input-alt-def
                language-state-for-inputs-alt-def subsetCE)
    then have (io,x)\in{io}\timesB M2 io \Omega
        by blast
    then have (io,x) \in(\bigcupio\inLin M2 T. {io} > B M2 io \Omega)
    using <io }\in\mp@subsup{L}{in}{}M1T> by aut
    then show iox }\in(\bigcup\mathrm{ iof Lin M2 T. {io} }\times\mathrm{ B M2 io ת)
    using}\langleiox=(io,x)〉 by aut
    qed
qed
lemma atc-io-reduction-on-sets-alt-def :
    shows atc-io-reduction-on-sets M1 T \Omega M2 =
        (L}\mp@subsup{L}{in}{}M1T\subseteq\mp@subsup{L}{in}{}M2
        \wedge(\bigcupio\inLin M1 T. {io} }\times\mathrm{ B M1 io }\Omega
            \subseteq ( \bigcup i o \in L _ { i n } M 2 ~ T . \{ i o \} \times B ~ M 2 ~ i o ~ \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 obs $I_{\Omega}$ iter isReduction
$\{$
OFSM M1 ^OFSM M2 $\wedge$ asc-fault-domain M2 M1 m $\wedge$ test-tools M2 M1 FAIL PM V $\Omega$
\}
$t s N:=\{ \} ;$
$c N:=V$;
$r m N:=\{ \} ;$
obs $:=L_{i n} M 2 c N$;
$o b s I:=L_{i n} M 1 c N ;$
$o b s \Omega:=\left(\bigcup i o \in L_{i n} M 2 c N .\{i o\} \times B\right.$ M2 io $\left.\Omega\right)$;
$o b s I_{\Omega}:=\left(\bigcup i o \in L_{i n} M 1 c N .\{i o\} \times B M 1\right.$ io $\left.\Omega\right) ;$
iter $:=1$;
WHILE $\left(c N \neq\{ \} \wedge o b s I \subseteq o b s \wedge o b s I_{\Omega} \subseteq o b s \Omega\right)$
INV \{
$0<$ iter
$\wedge t s N=T S M 2 M 1 \Omega V m($ iter -1$)$
$\wedge c N=C$ M2 M1 $\Omega V m$ iter
$\wedge r m N=R M M 2 M 1 \Omega V m($ iter -1$)$
$\wedge o b s=L_{i n} M 2(t s N \cup c N)$
$\wedge o b s I=L_{i n} M 1(t s N \cup c N)$
$\wedge o b s_{\Omega}=\left(\bigcup i o \in L_{i n} M 2(t s N \cup c N) .\{i o\} \times B\right.$ M2 io $\left.\Omega\right)$
$\wedge o b s I_{\Omega}=\left(\bigcup i o \in L_{i n} M 1(t s N \cup c N) .\{i o\} \times B M 1\right.$ io $\left.\Omega\right)$
$\wedge$ OFSM M1 ^OFSM M2 ^ asc-fault-domain M2 M1 m ^test-tools M2 M1 FAIL PMV $\Omega$
\}
DO
iter $:=$ iter +1 ;
$r m N:=\left\{x s^{\prime} \in c N\right.$.
$\left(\neg\left(L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\}\right)\right)$
$\vee\left(\forall\right.$ io $\in L_{i n} M 1\left\{x s^{\prime}\right\}$.
$\left(\exists V^{\prime \prime} \in N\right.$ io M1 $V$.
( $\exists$ S1.
( $\exists$ vs $x s$.
$i o=(v s @ x s)$
$\wedge m c p(v s @ x s) V^{\prime \prime} v s$
$\wedge S 1 \subseteq$ nodes M2
$\wedge(\forall s 1 \in S 1 . \forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
$\left(\forall\right.$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall i o 2 \in R P M 2$ s2 vs xs $V^{\prime \prime}$.
B M1 io1 $\Omega \neq B$ M1 io2 $\Omega$ ))
$\wedge m<L B M 2 M 1$ vs $\left.\left.\left.\left.\left.x s(t s N \cup V) S 1 \Omega V^{\prime \prime}\right)\right)\right)\right)\right\} ;$

```
    tsN := tsN\cupcN;
    cN := append-set (cN - rmN) (inputs M2) - tsN;
    obs:=obs \cup Lin M2 cN;
    obsI := obsI \cup Lin M1 cN;
    obs\Omega}:=ob\mp@subsup{s}{\Omega}{}\cup(\bigcupio\in\mp@subsup{L}{in}{}M2 cN.{io} > B M2 io \Omega)
    obs\mp@subsup{I}{\Omega}{}:=obs\mp@subsup{I}{\Omega}{}\cup(\bigcupio\in\mp@subsup{L}{in}{}M1cN.{io}\timesBM1 io \Omega)
    OD;
    isReduction := ((obsI\subseteqobs)}\wedge(obs\mp@subsup{I}{\Omega}{}\subseteqob\mp@subsup{s}{\Omega}{})
    {
    isReduction = M1 \preceqM2 — 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 precond: OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega
    have {} = TS M2 M1 \Omega Vm (1-1)
        V = CM2 M1 \Omega Vm1
        {} = RM M2 M1 \OmegaVm(1-1)
        Lin M2 V = Lin M2 ({}\cupV)
        Lin}M1V=\mp@subsup{L}{in}{}M1({}\cupV
        (\bigcupio\inLin M2 V. {io} > B M2 io \Omega)
                =(\bigcupio\in\mp@subsup{L}{in}{\primeM}M2}({}\cupV).{io}\timesB M2 io \Omega
            (\bigcupio\in\mp@subsup{L}{in}{}M1V.{io} > B M1 io \Omega)
                =(\bigcupio\inLin}M1({}\cupV).{io}\timesBM1 io \Omega
        using precond by auto
    moreover have OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega
        using precond by assumption
    ultimately show 0< (1::nat) ^
                {} = TS M2 M1 \OmegaVm (1-1)^
                    V = CM2 M1 \Omega Vm1^
                    {} = RM M2 M1 \OmegaVm (1-1)^
                Lin
                    Lin M1 V = Lin M1 ({}\cupV)^
                    (\bigcupio\inLin M2 V. {io} > B M2 io \Omega)
                    =(\bigcupio\in\mp@subsup{L}{in}{}}\mathrm{ M2 ({} UV). {io} > B M2 io }\Omega)
                (\bigcupio\inLin M1 V. {io} }\times\mathrm{ B M1 io ת)
                    =(\bigcupio\in\mp@subsup{L}{in}{}M1
                    OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega
        by linarith+
next
    fix tsN cN rmN obs obsI obs\Omega obs\mp@subsup{I}{\Omega}{}}\mathrm{ iter isReduction
    assume precond:(0< iter ^
                tsN = TS M2 M1 \OmegaVm (iter - 1)^
                cN = C M2 M1 \Omega V m iter ^
                    rmN = RM M2 M1 \Omega Vm (iter - 1) ^
                    obs = Lin M2 (tsN\cupcN)^
                obsI=L Lin M1 (tsN\cupcN)^
                obs\Omega}=(\bigcupio\in\mp@subsup{L}{in}{}M2(tsN\cupcN).{io}\timesB M2 io \Omega)
                obs\mp@subsup{I}{\Omega}{}=(\bigcupio\in\mp@subsup{L}{in}{}M1(tsN\cupcN).{io}\timesB M1 io \Omega)^
                    OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega)
                    \wedge N \not={} ^obsI\subseteqobs ^obsI\Omega\subseteqobs\Omega
    then have 0< iter
        OFSM M1
        OFSM M2
        asc-fault-domain M2 M1 m
        test-tools M2 M1 FAIL PM V \Omega
        cN\not={}
        obsI\subseteqobs
        tsN=TS M2 M1 \Omega Vm(iter-1)
        cN = C M2 M1 \Omega V m iter
        rmN = RM M2 M1 \Omega Vm(iter-1)
        obs = Lin M2 (tsN\cupcN)
        obsI= Lin M1 (tsN\cupcN)
        obs\Omega}=(\bigcupio\in\mp@subsup{L}{in}{\prime}M2(tsN\cupcN).{io}\timesB M2 io \Omega
        obs\mp@subsup{I}{\Omega}{}=(\bigcupio\in\mp@subsup{L}{in}{}M1(tsN\cupcN).{io}\timesB M1 io \Omega)
        by linarith+
```

```
obtain k where iter = Suc k
    using grO-implies-Suc[OF <0 < iter`] by blast
then have cN=C M2 M1 \Omega Vm(Suck)
        tsN = TS M2 M1 \Omega Vmk
    using <cN = C M2 M1 \Omega Vm iter> <tsN = TS M2 M1 \Omega Vm(iter-1)> by auto
have TS M2 M1 \Omega Vm iter = TS M2 M1 \Omega Vm (Suc k)
    C M2 M1 \Omega V m iter = C M2 M1 \Omega Vm (Suc k)
    RM M2 M1 \Omega Vm iter = RM M2 M1 \Omega Vm (Suc k)
    using}<iter = Suc k> by presburger +
have rmN-calc[simp]: {xs'\incN.
    \neg io-reduction-on M1 {xs'} M2 \vee
    ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\prime\prime}M1 {xs'}
        \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V
            \existsS1 vs xs.
                io = vs @ xs ^
                mcp (vs @ xs) V'vs vs
                S1\subseteq nodes M2 ^
                (}\foralls1\inS1
                    \forall s2\inS1.
                            s1\not= s2\longrightarrow
                            ( }\forall\textrm{io1}\inRP M2 s1 vs xs V''. \forallio2\inRP M2 s2 vs xs V''.
                            B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'')}=
    RM M2 M1 \Omega Vm iter
proof -
have {xs' \incN.
    \negio-reduction-on M1 {xs'} M2 \vee
    (}\forallio\in\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}
        \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
            \existsS1 vs xs.
                io =vs@ xs ^
                mcp (vs@ @s) V'vs \
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1\mathrm{ .
                    \forall }2\inS1\mathrm{ .
                            s1 # sc \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}.\forallio2\inRP M2 s2 vs xs V''.
                            B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'')}=
    {xs' \inC M2 M1 \Omega Vm (Suc k).
    \neg io-reduction-on M1 {xs'} M2 \vee
    ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\primen}M1 {xs'}
        \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V .
            \existsS1 vs xs.
                io =vs@ xs ^
                mcp (vs @ xs) V'vs ^
                S1\subseteq nodes M2 ^
            ( }\foralls1\inS1\mathrm{ .
                    \forall }2\inS1\mathrm{ .
                            s1 = s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}.\forallio2\inRP M2 s2 vs xs V''.
                            B M1 io1 \Omega\not= B M1 io2 \Omega))^
                m<LB M2 M1 vs xs ((TS M2 M1 \OmegaVmk)\cupV)S1 \Omega V'')}
    using <cN = CM2 M1 \OmegaVm (Suck)\rangle\langletsN=TS M2 M1 \Omega Vmk` by blast
moreover have {xs' \inC M2 M1 \Omega Vm (Suc k).
        \neg io-reduction-on M1 {xs'} M2 \vee
        ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\prime}M1 {xs'}
            \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                    \existsS1 vs xs.
                io =vs@ @s ^
```

```
mcp(vs@ xs) V'vs ^
S1\subseteq nodes M2 ^
( }\foralls1\inS1\mathrm{ .
    \forall2\mp@code{SS1.}
                    s1 f s2 }
                    (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}.\forallio2\inRP M2 s2 vs xs V''.
                        B M1 io1 \Omega\not=B M1 io2 \Omega)) ^
                    m<LB M2 M1 vs xs ((TS M2 M1 \Omega Vm k)\cupV)S1 \Omega V'')}=
        RM M2 M1 \Omega Vm(Suc k)
    using RM.simps(2)[of M2 M1 \Omega Vm k] by blast
    ultimately have {x\mp@subsup{s}{}{\prime}\incN
        \negio-reduction-on M1 {xs'} M2 \vee
        (\forallio\inL Lin M1 {xs'}.
            \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V
                \existsS1 vs xs.
                io =vs@ @s^
                mcp(vs@ @s) V'vs ^
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1\mathrm{ .
                    \forall2 < S1.
                        s1 = s2\longrightarrow
                        (\forallio1\inRP M2 s1 vs xs V''.}\forallio2\inRP M2 s2 vs xs V''.
                            B M1 io1 \Omega\not=B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'')}=
            RM M2 M1 \Omega Vm(Suc k)
    by presburger
    then show ?thesis
    using〈iter = Suc k> by presburger
qed
moreover have RM M2 M1 \Omega Vm iter = RM M2 M1 \OmegaVm (iter + 1-1) by simp
ultimately have rmN-calc'':{xs''\incN.
    \neg io-reduction-on M1 {xs'} M2 \vee
    (\forallio\inL Lin M1 {xs'}.
        \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V
            \existsS1 vs xs.
                io =vs@ xs ^
                mcp(vs@ xs) V'v}vs
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1\mathrm{ .
                    \forall2\mp@code{SN1.}
                    s1 # s2\longrightarrow
```



```
                        B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V V')}=
    RM M2 M1 \Omega Vm(iter +1-1) by presburger
have tsN\cupcN=TS M2 M1 \Omega Vm(Suck)
proof (cases k)
    case 0
    then show ?thesis
        using <tsN = TS M2 M1 \Omega Vm k\rangle\langlecN=CM2 M1 \OmegaVm (Suc k)\rangle by auto
next
    case (Suc nat)
    then have TS M2 M1 \OmegaVm (Suc k) = TS M2 M1 \Omega Vmk\cupCM2 M1 \Omega Vm (Suc k)
        using TS.simps(3) by blast
    moreover have tsN\cupcN=TS M2 M1 \OmegaVmk\cupCM2 M1 \Omega Vm (Suc k)
        using <tsN = TS M2 M1 \Omega Vm k\rangle\langlecN=CM2 M1 \OmegaVm (Suc k)\rangle by auto
    ultimately show ?thesis
        by auto
qed
then have tsN-calc:tsN\cupcN=TS M2 M1 \Omega V m iter
    using}\langleiter = Suc k> by presburger
```

have $c N$-calc : append-set
( $c N-$

```
    {xs'\incN.
    \neg io-reduction-on M1 {xs'} M2 \vee
    (\forallio\inLin}M1{x\mp@subsup{s}{}{\prime}}
            \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V .
                    \existsS1 vs xs.
                io =vs@ xs ^
                mcp (vs @ xs) V'vs vs
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1\mathrm{ .
                    \foralls2\inS1.
                    s1\not=s2\longrightarrow
                    (\forallio1\inRP M2 s1 vs xs V''.
                            \forallio2\inRP M2 s2 vs xs V'". B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'\prime)})
    (inputs M2) -
    (tsN\cupcN)=
    CM2 M1 \OmegaVm(iter + 1)
proof -
    have append-set
        (cN -
            {xs'\incN.
            \negio-reduction-on M1 {xs'} M2 \vee
            ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\primen}M1{xs'}
                \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                    \existsS1 vs xs.
                    io =vs@ xs ^
                mcp (vs @ xs) V'vs ^
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1\mathrm{ .
                    \forall }2\inS1\mathrm{ .
                            s1 # s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                        \forallio2\inRP M2 s2 vs xs V''. B M1 io1 \Omega\not=B M1 io2 \Omega))^
                m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'I)})
            (inputs M2) -
            (tsN\cupcN)=
            append-set
            ((C M2 M1 \Omega V m iter) -
            (RM M2 M1 \Omega Vm iter))
            (inputs M2) -
            (TS M2 M1 \Omega V m iter)
    using <cN = C M2 M1 \Omega Vm iter\rangle\langletsN \cupcN=TS M2 M1 \OmegaVm iter` rmN-calc by presburger
moreover have append-set
            ((C M2 M1 \Omega V m iter) -
            (RM M2 M1 \Omega V m iter))
            (inputs M2) -
            (TS M2 M1 \OmegaVm iter ) = C M2 M1 \OmegaVm(iter + 1)
proof -
    have C M2 M1 \OmegaVm(iter + 1) = C M2 M1 \OmegaVm ((Suck) + 1)
        using <iter =Suc k> by presburger+
    moreover have (Suc k)+1=Suc (Suc k)
        by simp
    ultimately have CM2 M1 \OmegaVm(iter + 1)=C M2 M1 \OmegaVm(Suc (Suc k))
        by presburger
    have C M2 M1 \Omega Vm(Suc (Suc k))
                = append-set (C M2 M1 \OmegaVm(Suc k) - RM M2 M1 \OmegaVm(Suc k)) (inputs M2)
                - TS M2 M1 \Omega Vm(Suc k)
        using C.simps(3)[of M2 M1 \Omega Vmk] by linarith
    show ?thesis
        using Suc-eq-plus1
            `C M2 M1 \Omega Vm(Suc (Suc k))
            = append-set (C M2 M1 \OmegaVm(Suc k) - RM M2 M1 \Omega Vm (Suc k)) (inputs M2)
            - TS M2 M1 \Omega Vm(Suc k)>
            <iter = Suc k
    by presburger
```

```
    qed
    ultimately show ?thesis
    by presburger
qed
have obs-calc:obs }
    Lin M2
        (append-set
            (cN -
                {x\mp@subsup{s}{}{\prime}\incN.
                \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
```



```
                \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                                io =vs@ xs ^
                                mcp (vs @ xs) V'vs ^
                                S1\subseteq nodes M2 ^
                                ( }\forall\mathrm{ s1 GS1.
                            \forall s2\inS1.
                            s1 # s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                                    \forallio2\inRP M2 s2 vs xs V V''. B M1 io1 \Omega\not=B M1 io2 \Omega)) ^
                                    m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'I)})
            (inputs M2) -
        (tsN\cupcN))=
    Lin M2
        (tsN\cupcN\cup
        (append-set
            (cN -
            {x\mp@subsup{s}{}{\prime}\incN.
                    \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
                    (\forallio\in\mp@subsup{L}{in}{\primen}M1{xs'}.
                        \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                                io=vs@ xs^
                                mcp(vs@ @s) V'vs ^
                                S1\subseteq nodes M2 ^
                                ( }\forall\textrm{s}1\inS1\mathrm{ .
                            \forall s2\inS1.
                                    s1\not=s2\longrightarrow
                                    (\forall io1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                                    \forallio2\inRP M2 s2 vs xs V'|. B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                                    m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'\)})
            (inputs M2) -
            (tsN\cupcN)))
proof -
    have \A. Lin M2 (tsN\cupcN\cupA)=obs\cupLin M2 A
        by (metis (no-types) language-state-for-inputs-union precond)
    then show ?thesis
        by blast
qed
have obsI-calc: obsI U
    Lin M1
        (append-set
            (cN -
                {x\mp@subsup{s}{}{\prime}\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\prime\prime}M1 {xs'}
                \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                            io = vs @ xs ^
                                mcp (vs@ @s) V'vs ^
```

```
                S1\subseteq nodes M2 ^
                    (}\foralls1\inS1
                    \foralls2\inS1.
                            s1 # s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                                    \forallio2\inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}.BM1 io1 \Omega\not=B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'|})}
        (inputs M2) -
        (tsN\cupcN))=
    Lin M1
    (tsN\cupcN\cup
    (append-set
        (cN -
            {xs'\incN.
                \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            ( }\forallio\in\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}
                \existsV'\prime}\inN\mathrm{ io M1 V.
                \existsS1 vs xs.
                    io = vs @ xs ^
                                    mcp(vs@ @s) V'vs ^
                                    S1\subseteq nodes M2 ^
                                    ( }\foralls1\inS1
                                    \forall s Q <S1.
                                    s1\not=s2\longrightarrow
                                    (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                                    \forallo2\inRP M2 s2 vs xs V }\mp@subsup{V}{}{\prime\prime}\mathrm{ . B M1 io1 }\Omega\not=B M1 io2 \Omega))
                                    m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'\)})
        (inputs M2) -
        (tsN\cupcN)))
proof -
    have }\bigwedgeA.\mp@subsup{L}{in}{}M1(tsN\cupcN\cupA)=obsI\cup\mp@subsup{L}{in}{}M1
        by (metis (no-types) language-state-for-inputs-union precond)
    then show ?thesis
        by blast
qed
have obs\Omega-calc :obs\Omega}
    \ \io\inLin M2
        (append-set
            (cN -
            {xs'\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            (\forallio\inL Lin M1 {xs'}.
                        \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V
                        \existsS1 vs xs.
                                io =vs@xs^
                                mcp (vs@ @s) V'vs ^
                                S1\subseteq nodes M2 ^
                                ( }\foralls1\inS1
                                \forall2\inS1.
                                s1 f s2 \longrightarrow
                                (\forallio1\inRP M2 s1 vs xs V ''.
                            \forallio2\inRP M2 s2 vs xs V''. B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                                    m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'')})
            inputs M2) -
            (tsN\cupcN)).
        {io} }\times\mathrm{ B M2 io }\Omega)
    (\bigcupio\inLin M2
        (tsN\cupcN\cup
            (append-set
                (cN -
                    {xs'\incN.
                    \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
                    (\forallio\inL Lin M1 {xs'}.
                    \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V
                        \existsS1 vs xs.
```

```
            io = vs @ xs ^
            mcp (vs@ @s) V' vs ^
            S1\subseteq nodes M2 ^
            (}\foralls1\inS1
            \forall }2\inS1
                        s1\not=s2\longrightarrow
                            (\forall io1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                            \forallio2\inRP M2 s2 vs xs V''. B M1 io1 \Omega\not=B M1 io2 \Omega)) ^
                    m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'\prime)})
            (inputs M2) -
            (tsN\cupcN))).
                            {io} }\times\mathrm{ B M2 io }\Omega
using <obs = Lin M2 (tsN\cupcN)>
    <obs\Omega = (\bigcupio\inLin M2 (tsN\cupcN).{io} × B M2 io \Omega)>
    obs-calc
by blast
have obs\mp@subsup{I}{\Omega}{}-calc : obs\mp@subsup{I}{\Omega}{}\cup
    (\bigcupio\inLin M1
        (append-set
            (cN -
            {x\mp@subsup{s}{}{\prime}\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            (\forallio\inL Lin M1 {xs'}.
                \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V .
                        \existsS1 vs xs.
                                io =vs@xs^
                                mcp(vs@ @s) V'\primevs ^
                                S1\subseteq nodes M2 ^
                                ( }\foralls1\inS1\mathrm{ .
                        \forall2\mp@code{S1.}
                            s1 # s2 \longrightarrow
                            (\forall io1\inRP M2 s1 vs xs V''.
                                    \forallo2\inRP M2 s2 vs xs V',
                m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'')})
            (inputs M2) -
            (tsN\cupcN)).
        {io} }\times\mathrm{ BM1 io }\Omega)
    (Uio\inLin M1
        (tsN\cupcN\cup
        (append-set
            (cN -
            {x\mp@subsup{s}{}{\prime}\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}
            ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}\mathrm{ .
                            \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V
                        \existsS1 vs xs.
                                io =vs@ @s ^
                                mcp(vs@xs) V'\primevs ^
                                S1\subseteq nodes M2 ^
                                ( }\foralls1\inS1\mathrm{ .
                            \forall2 < S1.
                            s1 f s2 }
                                    ( }\forall\mathrm{ io1 RP M2 s1 vs xs V''.
                                    \forallio2\inRP M2 s2 vs xs V''. B M1 io1 \Omega\not=B M1 io2 \Omega))^
                                    m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'')})
            (inputs M2) -
            (tsN\cupcN))).
    {io} }\times\mathrm{ B M1 io }\Omega\mathrm{ )
using <obsI = Lin M1 (tsN\cupcN)>
    obs\mp@subsup{I}{\Omega}{}=(\bigcupio\in\mp@subsup{L}{in}{}M1(tsN\cupcN).{io}\timesBM1 io \Omega)〉
    obsI-calc
by blast
```

```
have 0< iter + 1
    using <0 < iter` by simp
have tsN\cupcN=TS M2 M1 \Omega Vm(iter + 1-1)
    using tsN-calc by simp
from <0< iter + 1>
<tsN\cupcN = TS M2 M1 \Omega Vm(iter + 1-1)>
cN-calc
rmN-calc'
obs-calc
obsI-calc
obs\Omega-calc
obs\mp@subsup{I}{\Omega}{}-calc
〈OFSM M1`
<OFSM M2`
<asc-fault-domain M2 M1 m`
<test-tools M2 M1 FAIL PM V \Omega`
show 0< iter + 1^
    tsN\cupcN=TS M2 M1 \Omega Vm(iter + 1-1)^
    append-set
    (cN -
        {xs'\incN.
        \neg Lin M1 {xs'}\subseteq Lin M2 {xs'} \vee
        ( }\forall\mathrm{ io 竝 M1 {xs'}.
            \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V
                        \existsS1 vs xs.
                io = vs @ xs ^
                mcp(vs@ @s) V'vs ^
                S1\subseteq nodes M2 ^
                ( }\foralls1\inS1
                    \forall s2\inS1.
                            s1 # s2 \longrightarrow
                            (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}
                            \forallio2\inRP M2 s2 vs xs V''. B M1 io1 \Omega\not=B M1 io2 \Omega))^
                m<LB M2 M1 vs xs (tsN\cupV) S1 \Omega V'I)})
    (inputs M2) -
    (tsN\cupcN)=
    C M2 M1 \Omega Vm(iter + 1)^
    {xs' \in cN.
    \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
    (\forallio\inLin M1 {xs'}.
        \exists}\mp@subsup{V}{}{\prime\prime}\inN io M1 V .
            \existsS1 vs xs.
                io =vs@ @s^
                    mcp (vs@ @s) V'vs ^
                S1\subseteq nodes M2 ^
                    ( }\forall\mathrm{ s1 GS1.
                    \forall }2\inS1
                            s1 = s2\longrightarrow
                            (\forallio1\inRP M2 s1 vs xs V ''.}\forallio2\inRP M2 s2 vs xs V V'.
                            B M1 io1 \Omega\not= B M1 io2 \Omega)) ^
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'')}=
    RM M2 M1 \Omega Vm(iter + 1-1)^
    obs U
    Lin M2
    (append-set
        (cN -
            {x\mp@subsup{s}{}{\prime}\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\primen}M1 {xs'}
                \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                                    io = vs @ xs ^
                                    mcp (vs@ @s) V'vs ^
```

```
                S1\subseteq nodes M2 ^
            ( }\forall\mathrm{ s1 GS1.
            \foralls2\inS1.
                    s1 # s2 \longrightarrow
                    (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                        \forallo2\inRP M2 s2 vs xs V V''. B M1 io1 \Omega\not=B M1 io2 \Omega)) ^
            m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'|})}
    (inputs M2) -
    (tsN\cupcN))=
Lin M2
    (tsN\cupcN\cup
    (append-set
        (cN -
            {xs'\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            (\forallio\in\mp@subsup{L}{in}{\primen}M1{xs'}.
            \existsV'\prime}\inN\mathrm{ io M1 V.
                \existsS1 vs xs.
                    io =vs @ xs ^
                                    mcp(vs@ @s) V'vs ^
                                    S1\subseteq nodes M2 ^
                                    ( }\foralls1\inS1
                                    \forall < < <S1.
                                    s1\not=s2\longrightarrow
                                    (\forallio1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}\mathrm{ .
                                    \forallio2\inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}.BM1 io1 \Omega\not=B M1 io2 \Omega))
                    m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'\prime})}
            (inputs M2) -
            (tsN\cupcN)))^
obsI U
Lin M1
    (append-set
            (cN -
            {xs'\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            ( }\forall\mathrm{ io }\in\mp@subsup{L}{in}{\prime\prime}M1 {xs'}
                \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                    \existsS1 vs xs.
                io=vs@ \s ^
                        mcp (vs@ @xs) V'vs^
                                S1\subseteq nodes M2 ^
                                ( }\foralls1\inS1\mathrm{ .
                        \foralls2\inS1.
                            s1 = s2\longrightarrow
                                    (}\forall\mp@code{io1\inRP M2 s1 vs xs }\mp@subsup{V}{}{\prime\prime}
                                    \forallio2\inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}.\mathrm{ . M1 io1 }\Omega\not=B M1 io2 \Omega))
                m<LB M2 M1 vs xs (tsN\cupV)S1\Omega V'`)})
            (inputs M2) -
            (tsN\cupcN))=
Lin M1
(tsN\cupcNU
    (append-set
        (cN -
            {xs'\incN.
            \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
            (\forallio\inLin M1 {xs'}.
                \existsV'\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                            io=vs@ xs ^
                                    mcp (vs@ @s) V'vs}
                                    S1\subseteq nodes M2 ^
                                    ( }\foralls1\inS1\mathrm{ .
                                    \foralls2\inS1.
                                    s1 # s2 \longrightarrow
                                    (\forallio1\inRP M2 s1 vs xs V''.
                                    \forallo2\inRP M2 s2 vs xs }\mp@subsup{V}{}{\prime\prime}.\mathrm{ . B M1 io1 }\Omega\not=B M1 io2 \Omega)) ^
```

$$
\left.\left.\left.m<L B \text { M2 M1 vs xs }(t s N \cup V) S 1 \Omega V^{\prime \prime}\right)\right\}\right)
$$

(inputs M2) -
$(t s N \cup c N))) \wedge$
$o b s_{\Omega} \cup$
$\left(\bigcup i o \in L_{i n} M 2\right.$ (append-set ( $c N$ $\left\{x s^{\prime} \in c N\right.$.
$\neg L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\} \vee$
( $\forall i o \in L_{i n} M 1\left\{x s^{\prime}\right\}$.
$\exists V^{\prime \prime} \in N$ io $M 1 V$.
$\exists$ S1 vs xs.
$i o=v s @ x s \wedge$
$m c p(v s @ x s) V^{\prime \prime} v s \wedge$ S1 $\subseteq$ nodes M2 $\wedge$ ( $\forall s 1 \in S 1$.
$\forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
( $\forall$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall$ io2 $\in R P$ M2 s2 vs xs $V^{\prime \prime}$. B M1 io1 $\Omega \neq B$ M1 io2 $\left.\left.\Omega\right)\right) \wedge$ $m<L B M 2 M 1$ vs $\left.\left.\left.x s(t s N \cup V) S 1 \Omega V^{\prime \prime}\right)\right\}\right)$
(inputs M2) -
$(t s N \cup c N))$.
$\{i o\} \times B$ M2 io $\Omega)=$
$\left(\bigcup i o \in L_{i n} M 2\right.$ $(t s N \cup c N \cup$ (append-set

$$
(c N-
$$

$\left\{x s^{\prime} \in c N\right.$.
$\neg L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n} M 2\left\{x s^{\prime}\right\} \vee$
( $\forall i o \in L_{i n} M 1\left\{x s^{\prime}\right\}$.
$\exists V^{\prime \prime} \in N$ io $M 1 V$. $\exists$ S1 vs xs. $i o=v s @ x s \wedge$ $m c p(v s @ x s) V^{\prime \prime} v s \wedge$ $S 1 \subseteq$ nodes M2 $\wedge$ ( $\forall s 1 \in S 1$.
$\forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
( $\forall$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall$ io2 $\in R P$ M2 s2 vs xs $V^{\prime \prime}$. B M1 io1 $\Omega \neq B$ M1 io2 $\left.\left.\Omega\right)\right) \wedge$ $m<L B$ M2 M1 vs xs $\left.\left.\left.(t s N \cup V) S 1 \Omega V^{\prime \prime}\right)\right\}\right)$
(inputs M2) -
$(t s N \cup c N))$ ).
$\{i o\} \times B$ M2 io $\Omega) \wedge$
${ }^{o b s} I_{\Omega} \cup$
$\left(\bigcup i o \in L_{i n} M 1\right.$
(append-set
( $c \mathrm{~N}$ -
$\left\{x s^{\prime} \in c N\right.$.
$\neg L_{i n} M 1\left\{x s^{\prime}\right\} \subseteq L_{i n}$ M2 $\left\{x s^{\prime}\right\} \vee$
( $\forall i o \in L_{\text {in }} M 1\left\{x s^{\prime}\right\}$.
$\exists V^{\prime \prime} \in N$ io $M 1 V$.
$\exists$ S1 vs xs.
$i o=v s @ x s \wedge$
$m c p(v s @ x s) V^{\prime \prime} v s \wedge$
S1 $\subseteq$ nodes M2 $\wedge$
( $\forall s 1 \in S 1$.
$\forall s 2 \in S 1$.
$s 1 \neq s 2 \longrightarrow$
( $\forall$ io1 $\in R P$ M2 s1 vs xs $V^{\prime \prime}$.
$\forall i o 2 \in R P$ M2 s2 vs xs $V^{\prime \prime}$. B M1 io1 $\Omega \neq B$ M1 io2 $\left.\left.\Omega\right)\right) \wedge$ $m<L B$ M2 M1 vs xs $\left.\left.\left.(t s N \cup V) S 1 \Omega V^{\prime \prime}\right)\right\}\right)$
(inputs M2) -
$(t s N \cup c N))$.
$\{i o\} \times B M 1$ io $\Omega)=$

```
(\bigcupio\inLin M1
    tsN\cupcN\cup
        (append-set
            (cN -
            {x\mp@subsup{s}{}{\prime}\incN.
                \neg Lin M1 {xs'}\subseteq Lin M2 {xs'}\vee
                ( }\forallio\in\mp@subsup{L}{in}{}M1{x\mp@subsup{s}{}{\prime}}
                    \exists}\mp@subsup{V}{}{\prime\prime}\inN\mathrm{ io M1 V.
                        \existsS1 vs xs.
                    io =vs@ @s ^
                    mcp (vs@ @s) V'/}vs
                    S1\subseteq nodes M2 ^
                    ( }\foralls1\inS1\mathrm{ .
                        \forall2\inS1.
                        s1 f s2 \longrightarrow
                        (\forallio1\inRP M2 s1 vs xs V''.
                        \forallo2\inRP M2 s2 vs xs V', B M1 io1 \Omega = B M1 io2 \Omega)) ^
                    m<LB M2 M1 vs xs (tsN\cupV)S1 \Omega V'')})
            (inputs M2) -
                (tsN\cupcN))).
            {io} }\times\mathrm{ BM1 io }\Omega)
            OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega
    by linarith
next
    fix tsN cN rmN obs obsI obs\Omega obsI\Omega iter isReduction
    assume precond : ( 0< iter ^
                tsN = TS M2 M1 \Omega Vm (iter - 1) ^
                cN = C M2 M1 \Omega Vm iter ^
                rmN = RM M2 M1 \Omega Vm (iter - 1) ^
                obs=\mp@subsup{L}{in}{}M2(tsN\cupcN)^
                obsI= Lin}M1(tsN\cupcN)
                obs\Omega}=(\bigcupio\in\mp@subsup{L}{in}{}M2(tsN\cupcN).{io}\timesB M2 io \Omega)
                obs\mp@subsup{I}{\Omega}{}=(\bigcupio\in\mp@subsup{L}{in}{}M1(tsN\cupcN).{io}\timesBM1 io \Omega)^
                OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega) ^
                \neg ( c N \neq \{ \} \wedge o b s I \subseteq o b s \wedge o b s I _ { \Omega } \subseteq o b s \Omega )
    then have 0< iter
        OFSM M1
        OFSM M2
    asc-fault-domain M2 M1 m
    test-tools M2 M1 FAIL PM V \Omega
    cN={}\vee\negobsI\subseteqobs\vee\negobs\mp@subsup{I}{\Omega}{}\subseteqobs\Omega
    tsN=TS M2 M1 \Omega Vm (iter-1)
    cN = C M2 M1 \Omega V m iter
    rmN = RM M2 M1 \Omega Vm (iter-1)
    obs=\mp@subsup{L}{in}{}M2(tsN\cupcN)
    obsI= Lin M1 (tsN\cupcN)
    obs\Omega}=(\bigcupio\in\mp@subsup{L}{in}{}M2(tsN\cupcN).{io}\timesB M2 io \Omega
    obs\mp@subsup{I}{\Omega}{}=(\bigcupio\in\mp@subsup{L}{in}{}M1 (tsN\cupcN).{io} }\times\mathrm{ B M1 io }\Omega
by linarith +
```

```
show \(\left(o b s I \subseteq o b s \wedge o b s I_{\Omega} \subseteq o b s_{\Omega}\right)=M 1 \preceq M 2\)
proof (cases \(c N=\{ \}\) )
    case True
    then have \(C\) M2 M1 \(\Omega V\) m iter \(=\{ \}\)
        using \(\langle c N=C\) M2 M1 \(\Omega V m\) iter \(\rangle\) by auto
    have is-det-state-cover M2 \(V\)
        using <test-tools M2 M1 FAIL PM V \(\Omega\) 〉 by auto
    then have []\(\in V\)
        using det-state-cover-initial[of M2 V] by simp
    then have \(V \neq\{ \}\)
        by blast
    have Suc \(0<\) iter
    proof (rule ccontr)
```

```
    assume \(\neg\) Suc \(0<\) iter
    then have iter \(=\) Suc 0
        using 〈 \(0<\) iter \(\rangle\) by auto
    then have CM2 M1 \(\Omega V m(\) Suc 0\()=\{ \}\)
        using 〈C M2 M1 \(\Omega V m\) iter \(=\{ \}\) 〉 by auto
    moreover have \(C\) M2 M1 \(\Omega V m(\) Suc 0\()=V\)
        by auto
    ultimately showFalse
        using \(\langle V \neq\{ \}\) 〉 by blast
qed
obtain \(k\) where iter \(=\) Suc \(k\)
    using gr0-implies-Suc[OF \(\langle 0<\) iter \(\rangle\) ] by blast
then have Suc \(0<\) Suc \(k\)
    using 〈Suc \(0<\) iter〉 by auto
then have \(0<k\)
    by \(\operatorname{simp}\)
then obtain \(k^{\prime}\) where \(k=S u c k^{\prime}\)
    using gr0-implies-Suc by blast
have iter \(=\) Suc (Suc \(\left.k^{\prime}\right)\)
    using \(\langle\) iter \(=\) Suc \(k\rangle\left\langle k=\right.\) Suc \(\left.k^{\prime}\right\rangle\) by simp
have TS M2 M1 \(\Omega V m\left(S u c\left(S u c k^{\prime}\right)\right)=T S M 2 M 1 \Omega V m\left(S u c k^{\prime}\right) \cup C M 2 M 1 \Omega V m\left(S u c\left(S u c k^{\prime}\right)\right)\)
    using TS.simps(3)[of M2 M1 \(\Omega V m k]\) by blast
then have TS M2 M1 \(\Omega V m\) iter \(=T S M 2 M 1 \Omega V m(S u c k ')\)
    using True \(\langle c N=C\) M2 M1 \(\Omega V\) m iter \(\rangle\left\langle\right.\) iter \(\left.=S u c\left(S u c k^{\prime}\right)\right\rangle\) by blast
moreover have Suc \(k^{\prime}=\) iter -1
    using \(\left\langle\right.\) iter \(=\) Suc (Suc \(k^{\prime}\) ) 〉 by presburger
ultimately have TS M2 M1 \(\Omega V m\) iter \(=T S M 2 M 1 \Omega V m(\) iter -1\()\)
    by auto
then have \(t s N=T S\) M2 M1 \(\Omega V\) m iter
    using \(\langle t s N=T S M 2 M 1 \Omega V m(\) iter -1\()\rangle\) by simp
then have TS M2 M1 \(\Omega V m\) iter \(=T S\) M2 M1 \(\Omega V m(\) iter -1\()\)
    using \(\langle t s N=T S M 2 M 1 \Omega V m(\) iter -1\()\rangle\) by auto
then have final-iteration M2 M1 \(\Omega V m(\) iter -1\()\)
    using \(<0<\) iter \(\rangle\) by auto
have \(M 1 \preceq M 2=\) atc-io-reduction-on-sets \(M 1 t s N \Omega M 2\)
    using asc-main-theorem[OF 〈OFSM M1〉〈OFSM M2〉
                    〈asc-fault-domain M2 M1 m〉
                    〈test-tools M2 M1 FAIL PM V \(\Omega\) 〉
                    〈final-iteration M2 M1 \(\Omega V m(\) iter -1\()\rangle]\)
    using \(\langle t s N=T S M 2 M 1 \Omega V m(\) iter -1\()\rangle\)
    by blast
moreover have \(t s N \cup c N=t s N\)
    using \(\langle c N=\{ \}\rangle\) by blast
ultimately have \(M 1 \preceq M 2=\) atc-io-reduction-on-sets M1 \((t s N \cup c N) \Omega\) M2
    by presburger
have \(o b s I \subseteq o b s \equiv L_{i n} M 1(t s N \cup c N) \subseteq L_{i n} M 2(t s N \cup c N)\)
    by \(\left(s i m p ~ a d d:<o b s=L_{i n}\right.\) M2 \(\left.\left.(t s N \cup c N)\right\rangle\left\langle o b s I=L_{i n} M 1(t s N \cup c N)\right\rangle\right)\)
have \(o b s I_{\Omega} \subseteq o b s_{\Omega} \equiv\left(\bigcup i o \in L_{i n} M 1(t s N \cup c N) .\{i o\} \times B M 1\right.\) io \(\left.\Omega\right)\)
                    \(\subseteq\left(\bigcup i o \in L_{i n} M 2(t s N \cup c N) .\{i o\} \times B\right.\) M2 io \(\left.\Omega\right)\)
    by \(\left(\right.\) simp add: \(\left\langle o b s I_{\Omega}=\left(\bigcup i o \in L_{i n} M 1(t s N \cup c N) .\{i o\} \times B M 1\right.\right.\) io \(\left.\left.\Omega\right)\right\rangle\)
            \(\left\langle o b s_{\Omega}=\left(\bigcup i o \in L_{i n} M 2(t s N \cup c N) .\{i o\} \times B\right.\right.\) M2 io \(\left.\left.\left.\Omega\right)\right\rangle\right)\)
have \(\left(o b s I \subseteq o b s \wedge o b s I_{\Omega} \subseteq o b s_{\Omega}\right)=\) atc-io-reduction-on-sets \(M 1(t s N \cup c N) \Omega\) M2
proof
    assume \(o b s I \subseteq o b s \wedge o b s I_{\Omega} \subseteq o b s_{\Omega}\)
    show atc-io-reduction-on-sets M1 (tsN \(\cup c N) \Omega\) M2
        using atc-io-reduction-on-sets-from-obs[of M1 tsN \(\cup c N\) M2 \(\Omega\) ]
        using \(\left\langle o b s I \subseteq o b s \wedge o b s I_{\Omega} \subseteq o b s_{\Omega}\right\rangle\left\langle o b s I \subseteq o b s \equiv L_{i n} M 1(t s N \cup c N) \subseteq L_{i n} M 2(t s N \cup c N)\right.\) )
                \(\prec o b s I_{\Omega} \subseteq o b s_{\Omega} \equiv\left(\bigcup i o \in L_{i n} M 1(t s N \cup c N) .\{i o\} \times B M 1\right.\) io \(\left.\Omega\right)\)
```

```
        by linarith
    next
        assume atc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2
        show obsI\subseteqobs ^ obsI\Omega}\subseteq\mp@subsup{I}{\Omega}{}\subseteqob\mp@subsup{s}{\Omega}{
            using atc-io-reduction-on-sets-to-obs[of M1 <tsN\cupcN\rangle\Omega M2]
                <atc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2>
                    <obsI\subseteqobs\equivL Lin M1 (tsN\cupcN)\subseteq L Lin M2 (tsN\cupcN)>
                    <obs\mp@subsup{I}{\Omega}{}\subseteqobs\Omega}\equiv(\bigcupio\in\mp@subsup{L}{in}{}M1(tsN\cupcN).{io}\timesBM1 io \Omega
                        \subseteq(\bigcupio\inLin M2 (tsN\cupcN).{io} × B M2 io \Omega)>
        by blast
    qed
    then show ?thesis
        using<M1 \preceqM2 = atc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2> by linarith
next
    case False
    then have }\negobsI\subseteqobs\vee\negobs\mp@subsup{I}{\Omega}{}\subseteqobs
        using <cN}={}\vee\vee\negobsI\subseteqobs\vee\negobs\mp@subsup{I}{\Omega}{}\subseteqobs\mp@subsup{\Omega}{\Omega}{}\rangle\mathrm{ by auto
    have ᄀatc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2
        using atc-io-reduction-on-sets-to-obs[of M1 tsN\cupcN \Omega M2]
            \negobsI\subseteqobs}\vee\negobs\mp@subsup{I}{\Omega}{}\subseteqob\mp@subsup{s}{\Omega}{}\rangle\mathrm{ precond
        by fastforce
    have \negM1 \preceq M2
    proof
        assume M1 \preceq M2
        have atc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2
        using asc-soundness[OF〈OFSM M1〉\langleOFSM M2〉]〈M1 \preceq M2〉 by blast
        then show False
            using<\neg atc-io-reduction-on-sets M1 (tsN\cupcN) \Omega M2` by blast
    qed
    then show ?thesis
        using \prec\negobsI\subseteqobs\vee\negobs\mp@subsup{I}{\Omega}{}\subseteqobs\Omega\rangle
    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 \(\times\left({ }^{\prime}\right.\) in \(\times\) 'out \() \times\) 'state \()\) set \(\Rightarrow{ }^{\prime}\) 'state \(\Rightarrow\) ('in, 'out, 'state) FSM where
from-rel rel \(q 0=0\) succ \(=\lambda\) io \(p .\{q .(p, i o, q) \in\) rel \(\}\),
inputs \(=\) image \((f s t \circ f s t \circ s n d) r e l\),
outputs \(=\) image \((\) snd \(\circ f\) st \(\circ\) snd \()\) rel,
initial \(=q 0\) )
```

```
lemma nodes-from-rel : nodes (from-rel rel q0) \subseteqinsert q0 (image (snd \circ snd) rel)
    (is nodes ?M \subseteq insert q0 (image (snd \circ snd) rel))
proof -
    have \q io p.q\in succ ?M io p\Longrightarrowq\in image(snd o snd) rel
        by force
    have }\bigwedgeq.q\in\mathrm{ nodes ?M }\Longrightarrowq=q0\veeq\in\mathrm{ image (snd }\circ\mathrm{ snd) rel
    proof -
        fix q}\mathrm{ assume }q\in\mathrm{ nodes ?M
        then show }q=q0\veeq\in\mathrm{ image (snd ○ snd) rel
        proof (cases rule: FSM.nodes.cases)
            case initial
            then show ?thesis by auto
        next
            case (execute p a)
            then show ?thesis
                using <\ q io p . q\in succ ?M io p\Longrightarrowq\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 $\Rightarrow$ bool where
well-formed-rel rel $=($ finite rel

$$
\begin{aligned}
\wedge(\forall \text { s1 } x y & (x \notin \text { image }(f s t \circ f \text { fst } \circ \text { snd }) \text { rel } \\
& \vee y \notin \text { image }(\text { snd } \circ \text { fst } \circ \text { snd }) \text { rel }) \\
& \longrightarrow(\exists \text { s2 } \cdot(\text { s1, }(x, y), s 2) \in \text { rel }))
\end{aligned}
$$

$$
\wedge r e l \neq\{ \})
$$

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 $q 0$ (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 $\bigwedge s 1 x y .(x \notin$ inputs $? M \vee y \notin$ outputs $? M) \longrightarrow$ succ $? M(x, y) s 1=\{ \}$
using assms by auto
ultimately show ?thesis
by auto
qed

```
fun completely-specified-rel-over :: ('state \(\times\) ('in \(\times\) 'out) \(\times\) 'state) set \(\Rightarrow\) 'state set \(\Rightarrow\) 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.
\(\quad \exists\) s2. \((s 1,(x, y), s 2) \in\) 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 \in nodes (from-rel rel q0)
    show }\forallx\in\mathrm{ inputs ?M. ヨy€outputs ?M. ヨs2. s2 }\in\mathrm{ succ ?M (x,y) s1
    proof
        fix }x\mathrm{ assume }x\in\mathrm{ inputs (from-rel rel q0)
        then have }x\in\mathrm{ image (fst ०fst ० snd) rel
            using assms by auto
        obtain y s2 where y f image (snd \circ fst \circ snd) rel (s1,(x,y),s2) \in rel
            using assms «s1\in nodes (from-rel rel q0)\rangle\langlex\in image (fst \circfst \circ snd) rel>
            by (meson completely-specified-rel-over.elims(2))
        then have }y\in\mathrm{ outputs (from-rel rel q0) s2 }\in\mathrm{ succ (from-rel rel q0) (x,y) s1
            by auto
        then show \exists y\inoutputs (from-rel rel q0). \exists s2. s2 \in succ (from-rel rel q0) (x,y) s1
        by blast
    qed
qed
```

fun observable-rel :: ('state $\times($ 'in $\times$ 'out $) \times$ 'state) set $\Rightarrow$ bool where
observable-rel rel $=(\forall$ io s1 $\cdot\{s 2 \cdot(s 1$, io,s2 $) \in$ rel $\}=\{ \}$
$\left.\vee\left(\exists s 2 \cdot\left\{s 2^{\prime} \cdot\left(s 1, i o, s 2^{\prime}\right) \in \operatorname{rel}\right\}=\{s 2\}\right)\right)$
lemma observable-from-rel :
assumes observable-rel rel
shows observable (from-rel rel q0) (is observable ?M)
proof -
have $\bigwedge$ io s1 . \{ s2 . $(s 1, i o, s 2) \in$ rel $\}=$ succ ? M io s1 by auto
then show ?thesis using assms by auto
qed
abbreviation OFSM-rel rel q0 $\equiv$ well-formed-rel rel
$\wedge$ completely-specified-rel-over rel (nodes (from-rel rel q0))
$\wedge$ observable-rel rel
lemma OFMS-from-rel :
assumes OFSM-rel rel q0
shows OFSM (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 $M_{S}$-rel $::(n a t \times(n a t \times n a t) \times n a t)$ set $\equiv\{(0,(0,0), 1),(0,(0,1), 1),(1,(0,2), 1)\}$
abbreviation $M_{S}::(n a t, n a t, n a t) F S M \equiv$ from-rel $M_{S}$-rel 0
abbreviation $M_{I}$-rel $::(n a t \times(n a t \times n a t) \times n a t)$ set $\equiv\{(0,(0,0), 1),(0,(0,1), 1),(1,(0,2), 0)\}$
abbreviation $M_{I}::($ nat,nat,nat $) F S M \equiv$ from-rel $M_{I}$-rel 0
lemma example-nodes:
nodes $M_{S}=\{0,1\}$ nodes $M_{I}=\{0,1\}$
proof -

```
    have \(0 \in\) nodes \(M_{S}\) by auto
    have \(1 \in \operatorname{succ} M_{S}(0,0) 0\) by auto
    have \(1 \in\) nodes \(M_{S}\)
        by (meson \(\left\langle 0 \in\right.\) nodes \(\left.M_{S}\right\rangle\left\langle 1 \in \operatorname{succ} M_{S}(0,0) 0\right\rangle\) succ-nodes)
    have \(\{0,1\} \subseteq\) nodes \(M_{S}\)
    using \(\left\langle 0 \in\right.\) nodes \(\left.M_{S}\right\rangle\left\langle 1 \in\right.\) nodes \(\left.M_{S}\right\rangle\) by auto
    moreover have nodes \(M_{S} \subseteq\{0,1\}\)
    using nodes-from-rel[of \(M_{S}\)-rel 0] by auto
    ultimately show nodes \(M_{S}=\{0,1\}\)
    by blast
next
    have \(0 \in\) nodes \(M_{I}\) by auto
    have \(1 \in \operatorname{succ} M_{I}(0,0) 0\) by auto
    have \(1 \in\) nodes \(M_{I}\)
        by (meson \(\left\langle 0 \in\right.\) nodes \(\left.M_{I}\right\rangle\left\langle 1 \in \operatorname{succ} M_{I}(0,0) 0\right\rangle\) succ-nodes)
    have \(\{0,1\} \subseteq\) nodes \(M_{I}\)
        using \(\left\langle 0 \in\right.\) nodes \(\left.M_{I}\right\rangle\left\langle 1 \in\right.\) nodes \(\left.M_{I}\right\rangle\) by auto
    moreover have nodes \(M_{I} \subseteq\{0,1\}\)
        using nodes-from-rel[of \(M_{I}\)-rel 0] by auto
    ultimately show nodes \(M_{I}=\{0,1\}\)
        by blast
qed
lemma example-OFSM :
    OFSM M \(M_{S}\) OFSM M \(M_{I}\)
proof -
    have well-formed-rel \(M_{S}\)-rel
        unfolding well-formed-rel.simps by auto
    moreover have completely-specified-rel-over \(M_{S}\)-rel (nodes (from-rel \(M_{S}\)-rel 0))
        unfolding completely-specified-rel-over.simps
    proof
        fix \(s 1\) assume ( \(s 1:: n a t) \in\) nodes (from-rel \(M_{S}-\) rel 0\()\)
        then have s1 \(\in\) (insert 0 (image (snd \(\circ\) snd) \(M_{S}\)-rel) )
            using nodes-from-rel \(\left[\right.\) of \(M_{S}\)-rel 0\(]\) by blast
        moreover have completely-specified-rel-over \(M_{S}\)-rel (insert 0 (image (snd \(\circ\) snd) \(M_{S}\)-rel) )
            unfolding completely-specified-rel-over.simps by auto
        ultimately show \(\forall x \in(f s t \circ f s t \circ\) snd \()\) ' \(M_{S}-r e l\).
                        \(\exists y \in(\) snd \(\circ f s t \circ\) snd \() ' M_{S}-r e l . \exists s 2 .(s 1,(x, y), s 2) \in M_{S}-r e l\)
        by \(\operatorname{simp}\)
    qed
    moreover have observable-rel \(M_{S}\)-rel
        by auto
    ultimately have \(O F S M\)-rel \(M_{S}\)-rel 0
        by auto
    then show OFSM \(M_{S}\)
        using OFMS-from-rel[of \(M_{S}\)-rel 0] by linarith
next
    have well-formed-rel \(M_{I}\)-rel
        unfolding well-formed-rel.simps by auto
    moreover have completely-specified-rel-over \(M_{I}\)-rel (nodes (from-rel \(M_{I}\)-rel 0))
        unfolding completely-specified-rel-over.simps
    proof
        fix \(s 1\) assume ( \(s 1:: n a t) \in\) nodes (from-rel \(M_{I}\)-rel 0\()\)
        then have s1 \(\in\left(\right.\) insert \(0\left(\right.\) image \((\) snd \(\circ\) snd \() M_{I}\)-rel \(\left.)\right)\)
        using nodes-from-rel[ of \(M_{I}\)-rel 0] by blast
        have completely-specified-rel-over \(M_{I}\)-rel (insert 0 (image (snd \(\circ\) snd) \(M_{I}\)-rel))
            unfolding completely-specified-rel-over.simps by auto
```

```
    show }\forallx\in(fst\circfst\circsnd) ' M M -rel.
        \existsy\in(snd\circfst ○ snd) ' M M -rel. \existss\mathcal{Z. (s1, (x,y), s\mathcal{Z})\in M M-rel}
    by (meson <completely-specified-rel-over M M -rel (insert 0 ((snd \circ snd) ' }\mp@subsup{M}{I}{}\mathrm{ -rel))>
        «s1 \in insert 0 ((snd ○ snd) ' M M -rel)> completely-specified-rel-over.elims(2))
    qed
    moreover have observable-rel M}\mp@subsup{M}{I}{}\mathrm{ -rel
    by auto
    ultimately have OFSM-rel M M rel 0
    by auto
    then show OFSM M M
    using OFMS-from-rel[of M}\mp@subsup{M}{I}{}\mathrm{ -rel 0] by linarith
qed
```

lemma example-fault-domain : asc-fault-domain $M_{S} M_{I}$ 2
proof -
have inputs $M_{S}=$ inputs $M_{I}$
by auto
moreover have card (nodes $\left.M_{I}\right) \leq 2$
using example-nodes(2) by auto
ultimately show asc-fault-domain $M_{S} M_{I}$ 2
by auto
qed
abbreviation $F A I L_{I}::(n a t \times n a t) \equiv(3,3)$
abbreviation $P M_{I}::(n a t$, nat, nat $\times n a t) F S M \equiv 0$
succ $=\left(\lambda a(p 1, p 2) .\left(\right.\right.$ if $\left(p 1 \in\right.$ nodes $M_{S} \wedge p 2 \in$ nodes $M_{I} \wedge\left(f s t a \in\right.$ inputs $\left.M_{S}\right)$
$\wedge\left(\right.$ snd $a \in$ outputs $M_{S} \cup$ outputs $\left.\left.M_{I}\right)\right)$
then $\left(\right.$ if $\left(\right.$ succ $M_{S}$ a $p 1=\{ \} \wedge$ succ $M_{I}$ a $\left.p \mathbf{2} \neq\{ \}\right)$
then $\left\{F A I L_{I}\right\}$
else $\left(\operatorname{succ} M_{S}\right.$ a $p 1 \times \operatorname{succ} M_{I}$ a $\left.p 2\right)$ )
else \{\})),
inputs $=$ inputs $M_{S}$,
outputs $=$ outputs $M_{S} \cup$ outputs $M_{I}$,
initial $=\left(\right.$ initial $M_{S}$, initial $\left.M_{I}\right)$
D

```
lemma example-productF : productF \(M_{S} M_{I} F A I L_{I} P M_{I}\)
proof -
    have inputs \(M_{S}=\) inputs \(M_{I}\)
        by auto
    moreover have \(f s t F A I L_{I} \notin\) nodes \(M_{S}\)
        using example-nodes(1) by auto
    moreover have snd \(F A I L_{I} \notin\) nodes \(M_{I}\)
        using example-nodes(2) by auto
    ultimately show ?thesis
        unfolding productF.simps by blast
qed
```

abbreviation $V_{I}::$ nat list set $\equiv\{[],[0]\}$
lemma example-det-state-cover : is-det-state-cover $M_{S} V_{I}$
proof -
have $d$-reaches $M_{S}\left(\right.$ initial $\left.M_{S}\right)[]\left(\right.$ initial $\left.M_{S}\right)$
by auto
then have initial $M_{S} \in d$-reachable $M_{S}\left(\right.$ initial $\left.M_{S}\right)$
unfolding $d$-reachable.simps by blast
have $d$-reached-by $M_{S}\left(\right.$ initial $\left.M_{S}\right)[0] 1$ [1] [0]
proof
show length $[0]=$ length $[0] \wedge$
length $[0]=$ length $[1] \wedge$ path $M_{S}(([0] \|[0]) \|[1])\left(\right.$ initial $\left.M_{S}\right)$
$\wedge \operatorname{target}(([0] \|[0]) \|[1])\left(\right.$ initial $\left.M_{S}\right)=1$
by auto

```
have \(\bigwedge y s 2 \operatorname{tr2}\).
    length \([0]=\) length ys2
        \(\wedge\) length \([0]=\) length tr2
        \(\wedge\) path \(M_{S}(([0] \| y s 2) \| t r 2)\left(\right.\) initial \(\left.M_{S}\right)\)
            \(\longrightarrow \operatorname{target}(([0] \| y s \mathcal{Z}) \| \operatorname{tr} \mathcal{Z})\left(\right.\) initial \(\left.M_{S}\right)=1\)
    proof
    fix ys2 tr2 assume length \([0]=\) length ys2 \(\wedge\) length \([0]=\) length tr2
                \(\wedge\) path \(M_{S}(([0] \| y s 2) \|\) tr2 \()\left(\right.\) initial \(\left.M_{S}\right)\)
    then have length ys2 \(=1\) length tr2 \(=1\) path \(M_{S}(([0] \| y s 2) \|\) tr2 \()\left(\right.\) initial \(\left.M_{S}\right)\)
        by auto
    moreover obtain \(y^{2}\) where \(y s 2=[y 2]\)
        using <length ys2 \(=1\) 〉
        by (metis One-nat-def 〈length \([0]=\) length ys2 \(\wedge\) length \([0]=\) length tr2
            \(\wedge\) path \(M_{S}\left(([0] \| y s 2) \|\right.\) tr2) \(\left(\right.\) initial \(\left.\left.M_{S}\right)\right\rangle\) append.simps(1) append-butlast-last-id
            butlast-snoc length-butlast length-greater-0-conv list.size(3) nat.simps(3))
    moreover obtain \(t 2\) where tr2 \(=\) [t2]
        using 〈length tr2 \(=1\rangle\)
        by (metis One-nat-def 〈length \([0]=\) length ys2 \(\wedge\) length \([0]=\) length tr2
            \(\wedge\) path \(M_{S}(([0] \| y s 2) \|\) tr2 \()\left(\right.\) initial \(\left.\left.M_{S}\right)\right\rangle\) append.simps(1) append-butlast-last-id
            butlast-snoc length-butlast length-greater-0-conv list.size(3) nat.simps(3))
    ultimately have path \(M_{S}[((0, y \mathcal{2}), t 2)]\left(\right.\) initial \(\left.M_{S}\right)\)
        by auto
    then have \(t 2 \in \operatorname{succ} M_{S}\left(0, y\right.\) 2) \(\left(\right.\) initial \(\left.M_{S}\right)\)
        by auto
    moreover have \(\bigwedge y\). succ \(M_{S}(0, y)\left(\right.\) initial \(\left.M_{S}\right) \subseteq\{1\}\)
        by auto
    ultimately have \(t 2=1\)
        by blast
    show target \((([0] \| y s \mathcal{Z}) \|\) tr2 \()\left(\right.\) initial \(\left.M_{S}\right)=1\)
        using \(\langle y s 2=[y\) 2 \(]\rangle\langle t r 2=[t 2]\rangle\langle t 2=1\rangle\) by auto
    qed
    then show \(\forall y s 2\) tr2.
        length \([0]=\) length ys2 \(\wedge\) length \([0]=\) length tr2
        \(\wedge\) path \(M_{S}(([0] \| y s 2) \|\) tr2 \()\left(\right.\) initial \(\left.M_{S}\right)\)
            \(\longrightarrow \operatorname{target}(([0] \| y s \mathcal{Z}) \| \operatorname{tr2})\left(\right.\) initial \(\left.M_{S}\right)=1\)
    by auto
qed
then have \(d\)-reaches \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)\) [0] 1
    unfolding \(d\)-reaches.simps by blast
then have \(1 \in d\)-reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)\)
    unfolding \(d\)-reachable.simps by blast
then have \(\{0,1\} \subseteq d\)-reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)\)
    using \(\left\langle\right.\) initial \(M_{S} \in d\)-reachable \(M_{S}\left(\right.\) initial \(\left.\left.M_{S}\right)\right\rangle\) by auto
moreover have \(d\)-reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right) \subseteq\) nodes \(M_{S}\)
proof
    fix \(s\) assume \(s \in d\)-reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)\)
    then have \(s \in\) reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)\)
        using \(d\)-reachable-reachable by auto
    then show \(s \in\) nodes \(M_{S}\)
        by blast
qed
ultimately have \(d\)-reachable \(M_{S}\left(\right.\) initial \(\left.M_{S}\right)=\{0,1\}\)
    using example-nodes(1) by blast
```

fix $f^{\prime}::$ nat $\Rightarrow$ nat list
let ? $f=f^{\prime}(0:=[], 1:=[0])$

```
have is-det-state-cover-ass MS ?f
    unfolding is-det-state-cover-ass.simps
proof
    show ?f (initial MS)= \] by auto
    show }\foralls\ind\mathrm{ -reachable }\mp@subsup{M}{S}{}(\mathrm{ initial }\mp@subsup{M}{S}{})\mathrm{ . d-reaches M M (initial M M) (?f s)s
    proof
        fix s}\mathrm{ assume sfd-reachable }\mp@subsup{M}{S}{(}(\mathrm{ initial MS)
        then have s\in reachable MS (initial MS)
            using d-reachable-reachable by auto
        then have s\in nodes M}\mp@subsup{M}{S}{
            by blast
        then have s=0 \ s=1
            using example-nodes(1) by blast
        then show d-reaches }\mp@subsup{M}{S}{}(\mathrm{ initial }\mp@subsup{M}{S}{})(?f s)
        proof
            assume s=0
            then show d-reaches }\mp@subsup{M}{S}{}(\mathrm{ initial }\mp@subsup{M}{S}{})(?f s)
            using <d-reaches MS (initial M M) [] (initial MS)> by auto
        next
            assume s=1
            then show d-reaches MS (initial MS)(?f s)s
            using 〈d-reaches }\mp@subsup{M}{S}{}(\mathrm{ initial }\mp@subsup{M}{S}{})[0] 1` by aut
        qed
    qed
qed
moreover have }\mp@subsup{V}{I}{}=\mathrm{ image ?f (d-reachable M M (initial MS))
    using <d-reachable M M (initial }\mp@subsup{M}{S}{})={0,1}> by aut
ultimately show ?thesis
    unfolding is-det-state-cover.simps by blast
qed
```

abbreviation $\Omega_{I}::($ nat, nat) ATC set $\equiv\{$ Node $0(\lambda y$. Leaf) $\}$
lemma applicable-set $M_{S} \Omega_{I}$
by auto
lemma example-test-tools : test-tools $M_{S} M_{I}$ FAIL $_{I} P M_{I} V_{I} \Omega_{I}$
using example-productF example-det-state-cover by auto
lemma OFSM-not-vacuous :
$\exists$ M :: (nat,nat,nat) FSM . OFSM M
using example-OFSM(1) by blast
lemma fault-domain-not-vacuous :
ヨ (M2::(nat,nat,nat) FSM) (M1::(nat,nat,nat) FSM) m . asc-fault-domain M2 M1 m
using example-fault-domain by blast
lemma test-tools-not-vacuous :
ヨ (M2::(nat,nat,nat) FSM)
(M1::(nat,nat,nat) FSM)
(FAIL::(nat×nat))
(PM::(nat,nat,nat×nat) FSM)
(V::(nat list set))
( $\Omega::($ nat,nat $)$ ATC set) . test-tools M2 M1 FAIL PM V $\Omega$
proof (rule exI, rule exI)

```
    show \exists FAIL PM V \Omega. test-tools MS M M FAIL PM V \Omega
    using example-test-tools by blast
qed
lemma precondition-not-vacuous:
    shows \exists (M2::(nat,nat,nat) FSM)
        (M1::(nat,nat,nat) FSM)
        (FAIL::(nat\timesnat))
        (PM::(nat,nat,nat\timesnat) FSM)
        (V::(nat list set))
        (\Omega::(nat,nat) ATC set)
        (m :: nat)
            OFSM M1 ^ OFSM M2 ^ asc-fault-domain M2 M1 m ^ test-tools M2 M1 FAIL PM V \Omega
proof (intro exI)
```



```
        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, 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.

