# Büchi Complementation 

Julian Brunner

September 13, 2023


#### Abstract

This entry provides a verified implementation of rank-based Büchi Complementation [1]. The verification is done in three steps: 1. Definition of odd rankings and proof that an automaton rejects a word iff there exists an odd ranking for it. 2. Definition of the complement automaton and proof that it accepts exactly those words for which there is an odd ranking. 3. Verified implementation of the complement automaton using the Isabelle Collections Framework.


## Contents

1 Alternating Function Iteration ..... 2
2 Run Graphs ..... 3
3 Rankings ..... 7
3.1 Rankings ..... 8
3.2 Ranking Implies Word not in Language ..... 8
3.3 Word not in Language Implies Ranking ..... 10
3.3.1 Removal of Endangered Nodes ..... 10
3.3.2 Removal of Safe Nodes ..... 10
3.3.3 Run Graph Interation ..... 11
3.4 Node Ranks ..... 16
3.5 Correctness Theorem ..... 18
4 Complementation ..... 18
4.1 Level Rankings and Complementation States ..... 18
4.2 Word in Complement Language Implies Ranking ..... 21
4.3 Ranking Implies Word in Complement Language ..... 25
4.4 Correctness Theorem ..... 32
5 Complementation Implementation ..... 32
5.1 Phase 1 ..... 32
5.2 Phase 2 ..... 37
5.3 Phase 3 ..... 39
5.4 Phase 4 ..... 41
5.5 Phase 5 ..... 47
5.6 Phase 6 ..... 49
5.7 Phase 7 ..... 50
6 Boolean Formulae ..... 54
7 Final Instantiation of Algorithms Related to Complementa- tion ..... 55
7.1 Syntax ..... 55
7.2 Hashcodes on Complement States ..... 55
7.3 Complementation ..... 56
7.4 Language Subset ..... 57
7.5 Language Equality ..... 58
8 Build and test exported program with MLton ..... 59

## 1 Alternating Function Iteration

## theory Alternate

imports Main
begin

```
    primrec alternate :: \(\left({ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow n a t \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a\right)\) where
        alternate \(\mathrm{f} g 0=\) id \(\mid\) alternate \(f g(\) Suc \(k)=\) alternate \(g f k \circ f\)
    lemma alternate-Suc \([\) simp \(]\) : alternate \(f g(\) Suc \(k)=(\) if even \(k\) then \(f\) else \(g) \circ\)
alternate \(f g k\)
    proof (induct \(k\) arbitrary: \(f g\) )
        case (0)
        show ?case by simp
    next
        case (Suc k)
        have alternate \(f g(\) Suc \((\) Suc \(k))=\) alternate \(g f(S u c k) \circ f\) by auto
        also have \(\ldots=(\) if even \(k\) then \(g\) else \(f) \circ(\) alternate \(g f k \circ f)\) unfolding Suc
by auto
        also have \(\ldots=(\) if even \((\) Suc \(k)\) then \(f\) else \(g) \circ\) alternate \(f g(S u c k)\) by auto
        finally show? case by this
    qed
    declare alternate.simps(2)[simp del]
    lemma alternate-antimono:
```

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


## 2 Run Graphs

theory Graph<br>imports Transition-Systems-and-Automata.NBA<br>begin

type-synonym 'state node $=$ nat $\times$ 'state
abbreviation ginitial $A \equiv\{0\} \times$ initial $A$
abbreviation gaccepting $A \equiv$ accepting $A \circ$ snd
global-interpretation graph: transition-system-initial
const
$\lambda u(k, p) . w!!k \in$ alphabet $A \wedge u \in\{S u c k\} \times \operatorname{transition} A(w!!k) p \cap V$
$\lambda v . v \in$ ginitial $A \cap V$
for $A w V$
defines
gpath $=$ graph.path and grun $=$ graph.run and greachable $=$ graph.reachable and gnodes $=$ graph.nodes
by this
We disable rules that are degenerate due to execute $=(\lambda x-. x)$.
declare graph.reachable.execute[rule del]
declare graph.nodes.execute[rule del]
abbreviation gtarget $\equiv$ graph.target
abbreviation gstates $\equiv$ graph.states
abbreviation gtrace $\equiv$ graph.trace
abbreviation gsuccessors :: ('label, 'state) nba $\Rightarrow$ 'label stream $\Rightarrow$
'state node set $\Rightarrow$ 'state node $\Rightarrow$ 'state node set where
gsuccessors A $w V$ graph.successors TYPE('label) w A V
abbreviation gusuccessors $A w \equiv$ gsuccessors $A$ w UNIV
abbreviation gupath $A w \equiv$ gpath $A w$ UNIV
abbreviation gurun $A w \equiv$ grun $A w U N I V$
abbreviation gureachable $A w$ greachable $A$ w UNIV
abbreviation gunodes $A w \equiv$ gnodes $A w U N I V$
lemma gtarget-alt-def: gtarget $r v=$ last $(v \# r)$ using fold-const by this
lemma gstates-alt-def: gstates $r v=r$ by simp
lemma gtrace-alt-def: gtrace $r v=r$ by simp
lemma gpath-elim[elim?]:
assumes gpath $A w V s v$
obtains $r k p$
where $s=[$ Suc $k . .<$ Suc $k+$ length $r] \| r v=(k, p)$
proof -
obtain $t r$ where $1: s=t \| r$ length $t=$ length $r$
using zip-map-fst-snd[of s] by (metis length-map)
obtain $k p$ where 2: $v=(k, p)$ by force
have 3: $t=[$ Suc $k . .<$ Suc $k+$ length $r]$
using assms 12
proof (induct arbitrary: trkp)
case (nil v)
then show ?case by (metis add-0-right le-add1 length-0-conv length-zip
min.idem upt-conv-Nil)
next
case (cons uvs)
have 1: $t \| r=(h d t, h d r) \#(t l t \| t l r)$
by (metis cons.prems(1) hd-Cons-tl neq-Nil-conv zip.simps(1) zip-Cons-Cons zip-Nil)
have 2: $s=t l t \| t l r$ using cons 1 by simp
have $t=h d t \#$ tl $t$ using cons(4) by (metis hd-Cons-tl list.simps(3)zip-Nil)
also have $h d t=$ Suc $k$ using 1 cons.hyps(1) cons.prems(1) cons.prems(3)
by auto
also have $t l t=[S u c(S u c k) . .<S u c(S u c k)+$ length $(t l r)]$
using $\operatorname{cons}(3)[O F$ 2] using $1\langle h d t=$ Suc $k\rangle$ cons.prems(1) cons.prems(2)
by auto
finally show ?case using cons.prems(2) upt-rec by auto
qed
show ?thesis using that 123 by simp
qed
lemma gpath-path[symmetric]: path $A$ (stake (length $r$ ) (sdrop $k w) \| r) p \longleftrightarrow$ gpath A w UNIV ([Suc $k . .<$ Suc $k+$ length $r] \| r)(k, p)$

```
proof (induct r arbitrary: \(k p\) )
    case (Nil)
    show ?case by auto
next
    case (Cons q r)
    have 1: path \(A\) (stake (length \(r\) ) (sdrop \((\) Suc \(k) w) \| r) q \longleftrightarrow\)
        gpath A wUNIV ([Suc (Suc k)..<Suc \(k+\operatorname{length}(q \# r)] \| r)(S u c k, q)\)
        using Cons [of Suc \(k q\) ] by simp
    have stake (length \((q \# r))(\) sdrop \(k w) \| q \# r=\)
        \((w!!k, q) \#(\) stake \((\) length \(r)(s d r o p(S u c k) w) \| r)\) by simp
    also have path \(A \ldots p \longleftrightarrow\)
        gpath A wUNIV ((Suc k,q) \# ([Suc (Suc k) ..<Suc \(k+\operatorname{length}(q \# r)] \|\)
r)) ( \(k, p\) )
        using 1 by auto
    also have \((\) Suc \(k, q) \#([\operatorname{Suc}(S u c k) . .<\operatorname{Suc} k+\operatorname{length}(q \# r)] \| r)=\)
        Suc \(k\) \# [Suc (Suc \(k\) ) .. \(<\) Suc \(k+\) length \((q \# r)] \| q \# r\) unfolding
zip-Cons-Cons by rule
    also have Suc \(k \#[\) Suc \((\) Suc \(k) . .<\) Suc \(k+\) length \((q \# r)]=[\) Suc \(k . .<\) Suc
\(k+\) length \((q \# r)]\)
        by (simp add: upt-rec)
    finally show ?case by this
qed
lemma grun-elim[elim?]:
    assumes grun \(A w V s v\)
    obtains \(r k p\)
    where \(s=\) from \(N(\) Suc \(k)\|\| v=(k, p)\)
proof -
    obtain \(t r\) where \(1: s=t| | \mid r\) using szip-smap by metis
    obtain \(k p\) where 2: \(v=(k, p)\) by force
    have 3: \(t=\) from \(N\) (Suc \(k\) )
        using assms unfolding 12
        by (coinduction arbitrary: trkp) (force iff: eq-scons elim: graph.run.cases)
    show ?thesis using that 123 by simp
qed
lemma run-grun:
    assumes run \(A(\) sdrop \(k w||\mid r) p\)
    shows gurun \(A w(\) from \(N(\) Suc \(k)\|\mid\|)(k, p)\)
    using assms by (coinduction arbitrary: \(k p r\) ) (auto elim: nba.run.cases)
lemma grun-run:
    assumes grun A w \(V(\) fromN \((\) Suc \(k)||\mid r)(k, p)\)
    shows run \(A(\) sdrop \(k w \mid \| r) p\)
proof -
    have 2: \(\exists\) ka wa. sdrop \(k\) (stl \(w::{ }^{\prime} a\) stream \()=\) sdrop \(k a w a \wedge P\) ka wa if \(P\)
(Suc k) \(w\) for \(P k w\)
    using that by (metis sdrop.simps(2))
    show ?thesis using assms by (coinduction arbitrary: \(k p w r\) ) (auto intro!: 2
```

```
elim: graph.run.cases)
    qed
    lemma greachable-reachable:
    fixes lqk p
    defines u\equiv(l,q)
    defines v\equiv(k,p)
    assumes u\in greachable A wVv
    shows q\in reachable A p
using assms(3, 1, 2)
proof (induct arbitrary: l q k p)
    case reflexive
    then show ?case by auto
next
    case (execute u)
    have 1:q e successors A (snd u) using execute by auto
    have snd u \in reachable A p using execute by auto
    also have q\in reachable A (snd u) using 1 by blast
    finally show ?case by this
qed
lemma gnodes-nodes: gnodes A w V\subseteqUNIV }\times\mathrm{ nodes A
proof
    fix v
    assume v\in gnodes A wV
    then show v\inUNIV }\times\mathrm{ nodes A by induct auto
qed
lemma gpath-subset:
    assumes gpath A wVrv
    assumes set (gstates r v)\subseteqU
    shows gpath A w Urv
    using assms by induct auto
lemma grun-subset:
    assumes grun A w Vrv
    assumes sset (gtrace r v)\subseteqU
    shows grun A w Urv
using assms
proof (coinduction arbitrary: r v)
    case (run as rv)
    have 1: grun A w V s a using run(1, 2) by fastforce
    have 2:a g gusuccessors A wv using run(1, 2) by fastforce
    show ?case using 12 run(1, 3) by force
qed
lemma greachable-subset: greachable A wVv\subseteqinsert vV
proof
    fix }
    assume u\in greachable A wVv
```

```
    then show }u\in\mathrm{ insert }vV\mathrm{ by induct auto
qed
    lemma gtrace-infinite:
    assumes grun A w Vrv
    shows infinite (sset (gtrace r v))
    using assms by (metis grun-elim gtrace-alt-def infinite-Ici sset-fromN sset-szip-finite)
    lemma infinite-greachable-gtrace:
    assumes grun A w Vrv
    assumes u\in sset (gtrace r v)
    shows infinite (greachable A w Vu)
proof -
    obtain i where 1: u = gtrace r v !! i using sset-range imageE assms(2) by
metis
    have 2: gtarget (stake (Suc i) r) v=u unfolding 1 sscan-snth by rule
    have infinite (sset (sdrop (Suc i) (gtrace r v)))
        using gtrace-infinite[OF assms(1)]
        by (metis List.finite-set finite-Un sset-shift stake-sdrop)
    also have sdrop (Suc i) (gtrace r v) = gtrace (sdrop (Suc i) r) (gtarget (stake
(Suc i) r) v)
            by simp
    also have sset ...\subseteq greachable A w Vu
            using assms(1) 2 by (metis graph.reachable.reflexive graph.reachable-trace
graph.run-sdrop)
    finally show ?thesis by this
    qed
    lemma finite-nodes-gsuccessors:
        assumes finite (nodes A)
        assumes v\in gunodes A w
        shows finite (gusuccessors A wv
    proof -
    have gusuccessors A wv\subseteqgureachable A wv by rule
    also have ...\subseteqgunodes A w using assms(2) by blast
    also have ...\subseteqUNIV }\times\mathrm{ nodes A using gnodes-nodes by this
    finally have 3: gusuccessors A wv\subseteqUNIV }\times\mathrm{ nodes A by this
    have gusuccessors A wv\subseteq{Suc (fst v)} }\times\mathrm{ nodes A using 3 by auto
    also have finite ... using assms(1) by simp
    finally show ?thesis by this
    qed
end
```


## 3 Rankings

theory Ranking
imports
Alternate

Graph
begin

### 3.1 Rankings

type-synonym 'state ranking $=$ 'state node $\Rightarrow$ nat
definition ranking :: ('label, 'state) nba $\Rightarrow$ 'label stream $\Rightarrow$ 'state ranking $\Rightarrow$ bool where
ranking $A w f \equiv$
$(\forall v \in$ gunodes $A w . f v \leq 2 * \operatorname{card}($ nodes $A)) \wedge$
( $\forall v \in$ gunodes $A w . \forall u \in$ gusuccessors $A$ w v. $f u \leq f v) \wedge$
( $\forall v \in$ gunodes $A w$. gaccepting $A v \longrightarrow$ even $(f v)) \wedge$
$(\forall v \in$ gunodes $A w . \forall r k$. gurun $A w r v \longrightarrow \operatorname{smap} f($ gtrace $r v)=$ sconst
$k \longrightarrow$ odd $k$ )

### 3.2 Ranking Implies Word not in Language

lemma ranking-stuck:
assumes ranking $A w f$
assumes $v \in$ gunodes $A w$ gurun $A w r v$
obtains $n k$
where smap $f($ gtrace $($ sdrop $n r)($ gtarget $($ stake $n r) v))=$ sconst $k$
proof -
have 0: $f u \leq f v$ if $v \in$ gunodes $A w u \in$ gusuccessors $A w v$ for $v u$ using assms(1) that unfolding ranking-def by auto
have 1: shd ( $v \# \#$ gtrace $r v) \in$ gunodes $A$ wing assms(2) by auto
have 2: sdescending ( $\operatorname{smap} f(v \# \#$ gtrace $r v)$ )
using 1 assms(3)
proof (coinduction arbitrary: r v rule: sdescending.coinduct)
case sdescending
obtain $u s$ where 1: $r=u \# \# s$ using stream.exhaust by blast
have 2: $v \in$ gunodes $A$ wing sdescending(1) by simp
have 3: gurun $A w(u \# \# s) v$ using sdescending(2) 1 by auto
have 4: $u \in$ gusuccessors $A w v$ using 3 by auto
have 5: u g gureachable A wv using graph.reachable-successors 4 by blast show ?case
unfolding 1
proof (intro exI conjI disjI1)
show $f u \leq f v$ using 024 by this
show shd $(u \# \#$ gtrace $s u) \in$ gunodes $A w$ using 25 by auto show gurun $A$ ws using 3 by auto
qed auto
qed
obtain $s k$ where 3: smap $f(v \# \#$ gtrace $r v)=s$ @- sconst $k$
using sdescending-stuck[OF 2] by metis
have gtrace (sdrop (Suc (length s)) r) (gtarget (stake (Suc (length s)) r) v)= sdrop (Suc (length s)) (gtrace r v)
using sscan-sdrop by rule
also have smap $f \ldots=\operatorname{sdrop}($ length $s)(\operatorname{smap} f(v \# \#$ gtrace $r v))$
by (metis 3 id-apply sdrop-simps(2) sdrop-smap sdrop-stl shift-eq siterate. $\operatorname{simps(2)~stream.sel(2))~}$
also have $\ldots=$ sconst $k$ unfolding 3 using shift-eq by metis
finally show? thesis using that by blast
qed
lemma ranking-stuck-odd:
assumes ranking $A w f$
assumes $v \in$ gunodes $A$ w gurun $A w r v$
obtains $n$
where Ball (sset (smap $f($ gtrace $(\operatorname{sdrop} n r)(g t a r g e t(s t a k e ~ n r) v)))$ odd
proof -
obtain $n k$ where 1: smap $f($ gtrace $($ sdrop $n r)($ gtarget $($ stake $n r) v))=$ sconst $k$
using ranking-stuck assms by this
have 2: gtarget (stake n $r$ ) $v \in$ gunodes $A w$
using $\operatorname{assms}(2,3)$ by (simp add: graph.nodes-target graph.run-stake)
have 3: gurun A $w($ sdrop $n r)($ gtarget $($ stake $n r) v)$
using $\operatorname{assms}(2,3)$ by (simp add: graph.run-sdrop)
have 4: odd $k$ using 123 assms(1) unfolding ranking-def by meson
have 5: Ball (sset (smap $f$ (gtrace (sdrop n r) (gtarget (stake $n r) v)$ )) odd unfolding 1 using 4 by simp
show ?thesis using that 5 by this
qed
lemma ranking-language:
assumes ranking $A w f$
shows $w \notin$ language $A$
proof
assume 1: $w \in$ language $A$
obtain $r p$ where 2: run $A(w \mid \| r) p p \in \operatorname{initial} A \operatorname{infs}(\operatorname{accepting} A)(p \# \#$
$r$ ) using 1 by rule
let $? r=$ from $N 1 \| \mid r$
let ? $v=(0, p)$
have 3: ?v $\in$ gunodes $A$ w gurun $A$ w ?r ?v using 2(1, 2) by (auto intro: run-grun)
obtain $n$ where 4: Ball (sset (smap f (gtrace (sdrop $n$ ? $r$ ) (gtarget (stake $n$
? $r$ ) ? $v())$ ) odd
using ranking-stuck-odd assms 3 by this
let ?s = stake $n$ ?r
let ? $t=s d r o p n ? r$
let $? u=$ gtarget $? s ? v$
have sset (gtrace ?t ?u) $\subseteq$ gureachable $A w$ ?v
proof (intro graph.reachable-trace graph.reachable-target graph.reachable.reflexive) show gupath $A w$ ?s ?v using graph.run-stake 3(2) by this show gurun $A w$ ?t ? u using graph.run-sdrop 3(2) by this
qed
also have $\ldots \subseteq$ gunodes $A$ w using 3(1) by blast
finally have 7: sset (gtrace ?t ?u) $\subseteq$ gunodes $A$ wy this
have 8: $\wedge p . p \in$ gunodes $A w \Longrightarrow$ gaccepting $A p \Longrightarrow$ even ( $f p$ )
using assms unfolding ranking-def by auto
have 9: $\bigwedge p . p \in \operatorname{sset}$ (gtrace ?t ? $u$ ) $\Longrightarrow$ gaccepting $A p \Longrightarrow$ even (f $p$ ) using 78 by auto
have 19: infs (accepting A) (smap snd ?r) using 2(3) by simp
have 18: infs (gaccepting A) ?r using 19 by simp
have 17: infs (gaccepting A) (gtrace ?r ? $v$ ) using 18 unfolding gtrace-alt-def by this
have 16: infs (gaccepting A) (gtrace (?s @- ?t) ?v) using 17 unfolding stake-sdrop by this
have 15: infs (gaccepting A) (gtrace ?t ?u) using 16 by simp
have 13: infs (even $\circ f$ ) (gtrace ?t ? u) using infs-mono[OF - 15] 9 by simp
have 12: infs even (smap $f$ (gtrace ?t ?u)) using 13 by (simp add: comp-def)
have 11: Bex (sset (smapf (gtrace ?t ?u))) even using 12 infs-any by metis
show False using 411 by auto
qed

### 3.3 Word not in Language Implies Ranking

### 3.3.1 Removal of Endangered Nodes

definition clean $::$ ('label, 'state) $n b a \Rightarrow$ 'label stream $\Rightarrow$ 'state node set $\Rightarrow$ 'state node set where
clean $A w V \equiv\{v \in V$. infinite (greachable $A w V v)\}$
lemma clean-decreasing: clean $A w V \subseteq V$ unfolding clean-def by auto
lemma clean-successors:
assumes $v \in V u \in$ gusuccessors $A w v$
shows $u \in$ clean $A w V \Longrightarrow v \in$ clean $A w V$
proof -
assume 1: $u \in$ clean $A w V$
have 2: $u \in V$ infinite (greachable $A w V u$ ) using 1 unfolding clean-def by auto
have 3: u g greachable A $w V v$ using graph.reachable.execute assms(2) 2(1) by blast
have 4: greachable $A w V u \subseteq$ greachable $A w V v$ using 3 by blast
have 5: infinite (greachable $A \bar{w}$ ) using 2(2) 4 by (simp add: infinite-super)
show $v \in$ clean $A w V$ unfolding clean-def using assms(1) 5 by simp
qed

### 3.3.2 Removal of Safe Nodes

definition prune :: ('label, 'state) nba $\Rightarrow$ 'label stream $\Rightarrow$ 'state node set $\Rightarrow$ 'state node set where
prune $A w V \equiv\{v \in V . \exists u \in$ greachable $A w V v$. gaccepting $A u\}$
lemma prune-decreasing: prune $A w V \subseteq V$ unfolding prune-def by auto lemma prune-successors:
assumes $v \in V u \in$ gusuccessors $A w v$
shows $u \in$ prune $A w V \Longrightarrow v \in$ prune $A w V$
proof -
assume 1: $u \in$ prune $A w V$
have 2: $u \in V \exists x \in$ greachable $A w V u$. gaccepting $A x$ using 1 unfolding prune-def by auto
have 3: $u \in$ greachable $A w V v$ using graph.reachable.execute assms(2) 2(1) by blast
have 4: greachable $A w V u \subseteq$ greachable $A w V v$ using 3 by blast
show $v \in$ prune $A w V$ unfolding prune-def using assms(1) 2(2) 4 by auto qed

### 3.3.3 Run Graph Interation

definition graph $::$ ('label, 'state) nba $\Rightarrow$ 'label stream $\Rightarrow$ nat $\Rightarrow$ 'state node set where
graph $A w k \equiv$ alternate $($ clean $A w)($ prune $A w) k($ gunodes $A w)$
abbreviation level $A w k l \equiv\{v \in$ graph $A w k . f s t v=l\}$
lemma graph- $0[$ simp $]$ : graph $A w 0=$ gunodes $A w$ unfolding graph-def by simp
lemma graph-Suc[simp]: graph $A w($ Suc $k)=($ if even $k$ then clean $A w$ else prune Aw) (graph Awk)
unfolding graph-def by simp
lemma graph-antimono: antimono (graph $A$ w)
using alternate-antimono clean-decreasing prune-decreasing
unfolding monotone-def le-fun-def graph-def
by metis
lemma graph-nodes: graph $A w k \subseteq$ gunodes $A$ w using graph-0 graph-antimono le0 antimonoD by metis
lemma graph-successors:
assumes $v \in$ gunodes $A w u \in$ gusuccessors $A w v$
shows $u \in \operatorname{graph} A w k \Longrightarrow v \in \operatorname{graph} A w k$
using assms
proof (induct $k$ arbitrary: $u v$ )
case 0
show ? case using 0 (2) by simp
next
case (Suc k)
have 1: v graph A $w k$ using Suc using antimono-iff-le-Suc graph-antimono rev-subsetD by blast
show ?case using Suc(2) clean-successors[OF 1 Suc(4)] prune-successors[OF 1 Suc(4)] by auto
qed

## lemma graph-level-finite:

assumes finite (nodes A)
shows finite (level $A w k l$ )
proof -
have level $A$ wkl$\subseteq\{v \in$ gunodes $A w$. fst $v=l\}$ by (simp add: graph-nodes subset-CollectI)
also have $\{v \in$ gunodes $A w$. fst $v=l\} \subseteq\{l\} \times$ nodes $A$ using gnodes-nodes by force
also have finite ( $\{l\} \times$ nodes $A$ ) using assms(1) by simp
finally show ?thesis by this
qed
lemma find-safe:
assumes $w \notin$ language $A$
assumes $V \neq\{ \} V \subseteq$ gunodes $A w$
assumes $\bigwedge v . v \in V \Longrightarrow$ gsuccessors $A w V v \neq\{ \}$
obtains $v$
where $v \in V \forall u \in$ greachable $A w V v$. $\neg$ gaccepting $A u$ proof (rule ccontr)
assume 1: $\neg$ thesis
have 2: $\bigwedge v . v \in V \Longrightarrow \exists u \in$ greachable $A w V$. gaccepting $A u$ using that 1 by auto
have 3: $\bigwedge r v . v \in$ initial $A \Longrightarrow$ run $A(w \| r) v \Longrightarrow$ fins $($ accepting $A) r$ using assms(1) by auto
obtain $v$ where $4: v \in V$ using $\operatorname{assms}(2)$ by force
obtain $x$ where 5: $x \in$ greachable $A w V$ gaccepting $A x$ using 24 by blast
obtain $y$ where 50: gpath $A w V y v x=$ gtarget $y v$ using 5(1) by rule
obtain $r$ where 6: grun $A w \operatorname{Vrx}$ infs $(\lambda x . x \in V \wedge$ gaccepting $A x) r$
proof (rule graph.recurring-condition)
show $x \in V \wedge$ gaccepting $A x$ using greachable-subset 45 by blast
next
fix $v$
assume 1: $v \in V \wedge$ gaccepting $A v$
obtain $v^{\prime}$ where 20: $v^{\prime} \in$ gsuccessors $A w V v \mathbf{u s i n g} \operatorname{assms}(4) 1$ by (meson IntE equals0I)
have 21: $v^{\prime} \in V$ using 20 by auto
have 22: $\exists u \in$ greachable $A w V v^{\prime} . u \in V \wedge$ gaccepting $A u$
using greachable-subset 221 by blast
obtain $r$ where 30: gpath $A w V r v^{\prime}$ gtarget $r v^{\prime} \in V \wedge$ gaccepting $A$ (gtarget $\left.r v^{\prime}\right)$
using 22 by blast
show $\exists r . r \neq[] \wedge$ gpath $A w V r v \wedge$ gtarget $r v \in V \wedge$ gaccepting $A$ (gtarget $r v)$
proof (intro exI conjI)
show $v^{\prime} \# r \neq[]$ by $\operatorname{simp}$
show gpath $A w V\left(v^{\prime} \# r\right) v$ using 2030 by auto
show gtarget ( $\left.v^{\prime} \# r\right) v \in V$ using 30 by simp
show gaccepting $A\left(\right.$ gtarget $\left.\left(v^{\prime} \# r\right) v\right)$ using 30 by simp
qed
qed auto
obtain $u$ where 100: $u \in$ ginitial $A v \in$ gureachable $A w u$ using $4 \operatorname{assms}(3)$
by blast
have 101: gupath $A w y v$ using gpath-subset 50(1) subset-UNIV by this
have 102: gurun $A w r x$ using grun-subset 6 (1) subset-UNIV by this obtain $t$ where 103: gupath $A$ w $t u v=$ gtarget $t u$ using 100(2) by rule have 104: gurun $A w(t @-y @-r) u$ using 10110210350 (2) by auto obtain $s q$ where 7: $t$ @-y@-r=fromN (Suc 0) \|\| su=(0,q)
using grun-elim[OF 104] 100(1) by blast
have 8: run $A(w||\mid s) q$ using grun-run[OF 104 [unfolded 7]] by simp
have 9: $q \in$ initial $A$ using 100(1) 7(2) by auto
have 91: sset $(\operatorname{trace}(w\|\| s) q) \subseteq$ reachable $A q$
using nba.reachable-trace nba.reachable.reflexive 8 by this
have 10: fins (accepting A) susing 398 by this
have 12: infs (gaccepting A) r using infs-mono[OF - 6(2)] by simp
have $s=\operatorname{smap}$ snd ( $t$ @-y @-r) unfolding 7(1) by simp
also have infs (accepting A) ... using 12 by (simp add: comp-def)
finally have 13: infs (accepting A) s by this
show False using 1013 by simp
qed
lemma remove-run:
assumes finite (nodes A) $w \notin$ language $A$
assumes $V \subseteq$ gunodes $A$ w clean $A w V \neq\{ \}$
obtains $v r$
where
grun $A w \operatorname{Vrv}$
sset $($ gtrace $r v) \subseteq$ clean A $w V$
sset (gtrace r $v$ ) $\subseteq-$ prune $A w($ clean $A w V)$
proof -
obtain $u$ where 1: $u \in$ clean $A w V \forall x \in$ greachable $A w($ clean $A w V) u$. $\neg$ gaccepting $A x$
proof (rule find-safe)
show $w \notin$ language $A$ using assms(2) by this
show clean $A w V \neq\{ \}$ using assms(4) by this
show clean $A w V \subseteq$ gunodes $A$ wing assms(3) by (meson clean-decreasing subset-iff)
next
fix $v$
assume 1: $v \in$ clean $A w V$
have 2: $v \in V$ using 1 clean-decreasing by blast
have 3: infinite (greachable A $w V v$ ) using 1 clean-def by auto
have gsuccessors $A w V v \subseteq$ gusuccessors $A w v$ by auto
also have finite ... using $2 \operatorname{assms}(1,3)$ finite-nodes-gsuccessors by blast
finally have 4 : finite (gsuccessors $A w V v$ ) by this
have 5: infinite (insert $v(\bigcup(($ greachable $A w V)$ '(gsuccessors A w V v) )) ) using graph.reachable-step 3 by metis
obtain $u$ where $6: u \in$ gsuccessors $A w V$ vinfinite (greachable A $w V u$ ) using 45 by auto

```
    have 7: u\in clean A w V using 6 unfolding clean-def by auto
    show gsuccessors A w (clean A wV)v\not={} using 6(1) 7 by auto
    qed auto
    have 2: u}\inV\mathrm{ using 1(1) unfolding clean-def by auto
    have 3: infinite (greachable A w Vu) using 1(1) unfolding clean-def by simp
    have 4: finite (gsuccessors A wVv) if v\ingreachable AwVu for v
    proof -
    have 1:v\inV using that greachable-subset 2 by blast
    have gsuccessors A wVv\subseteqgusuccessors A wv by auto
    also have finite ... using 1 assms(1, 3) finite-nodes-gsuccessors by blast
    finally show ?thesis by this
    qed
    obtain r where 5: grun A w V r u using graph.koenig[OF 3 4] by this
    have 6: greachable A wVu\subseteqV using 2 greachable-subset by blast
    have 7: sset (gtrace r u)\subseteqV
        using graph.reachable-trace[OF graph.reachable.reflexive 5(1)] 6 by blast
    have 8: sset (gtrace ru)\subseteq clean A w V
        unfolding clean-def using 7 infinite-greachable-gtrace[OF 5(1)] by auto
    have 9: sset (gtrace ru)\subseteqgreachable A w (clean AwV)u
    using 58 by (metis graph.reachable.reflexive graph.reachable-trace grun-subset)
    show ?thesis
    proof
        show grun A wVru using 5(1) by this
        show sset (gtrace r u)\subseteq clean A wV using 8 by this
        show sset (gtrace r u)\subseteq- prune A w (clean A wV)
        proof (intro subsetI ComplI)
            fix p
            assume 10:p\in sset (gtrace r u) p\in prune A w (clean A w V)
            have 20: \existsx\in greachable Aw (clean A wV) p.gaccepting A x
            using 10(2) unfolding prune-def by auto
        have 30:greachable A w(clean A wV) p\subseteqgreachable Aw(clean AwV)
u
                using 10(1) 9 by blast
        show False using 1(2) 20 30 by force
    qed
    qed
qed
lemma level-bounded:
    assumes finite (nodes A) w\not\in language A
    obtains n
    where \l.l\geqn\Longrightarrowcard (level A w (2*k)l)\leqcard (nodes A) -k
proof (induct k arbitrary: thesis)
    case (0)
    show ?case
    proof (rule 0)
    fix l :: nat
    have finite ({l} }\times\mathrm{ nodes A) using assms(1) by simp
    also have level A w 0l\subseteq{l} \times nodes A using gnodes-nodes by force
```

```
        also (card-mono) have card ... = card (nodes A) using assms(1) by simp
        finally show card (level A w (2*0)l)\leqcard (nodes A) - 0 by simp
        qed
    next
        case (Suc k)
    show ?case
    proof (cases graph A w (Suc (2*k)) ={})
        case True
        have 3: graph A w (2*Suc k)={} using True prune-decreasing by simp
blast
    show ?thesis using Suc(2) 3 by simp
    next
        case False
        obtain vr where 1:
            grun A w (graph A w (2*k)) rv
            sset (gtrace r v)\subseteq graph A w (Suc (2*k))
            sset (gtrace r v)\subseteq-graph A w (Suc (Suc (2*k)))
    proof (rule remove-run)
            show finite (nodes A) w\not\in language A using assms by this
            show clean A w (graph A w (2*k))\not={} using False by simp
            show graph A w (2*k)\subseteqgunodes A w using graph-nodes by this
    qed auto
    obtain lq where 2: v=(l,q) by force
    obtain n where 90: \l.n\leql\Longrightarrowcard (level A w (2*k)l)\leqcard (nodes
A) - k
            using Suc(1) by blast
    show ?thesis
    proof (rule Suc(2))
            fix }
            assume 100: n+Sucl\leqj
            have 6: graph A w (Suc (Suc (2*k)))\subseteqgraph Aw (Suc (2*k))
            using graph-antimono antimono-iff-le-Suc by blast
        have 101:gtrace r v !! (j - Suc l) \in graph A w (Suc (2*k)) using 1(2)
snth-sset by auto
            have 102: gtrace r v !! (j - Suc l)\not\in graph A w (Suc (Suc (2 * k))) using
1(3) snth-sset by blast
            have 103: gtrace r v !! (j - Suc l) \inlevel A w (Suc (2*k)) j
            using 1(1) 100 101 2 by (auto elim: grun-elim)
    have 104:gtrace rv !! (j - Suc l)\not\in level A w (Suc (Suc (2*k))) jusing
1 0 0 ~ 1 0 2 ~ b y ~ s i m p
    have level A w (2*Suck) j= level A w (Suc (Suc (2*k))) j by simp
    also have ...\subset level A w (Suc (2*k)) jusing 103 104 6 by blast
    also have ...\subseteq level A w (2*k) j by (simp add: Collect-mono clean-def)
    finally have 105: level A w (2*Suck) j\subset level A w (2*k) j by this
    have card (level A w (2*Suc k) j)<card (level A w (2*k) j)
        using assms(1) 105 by (simp add: graph-level-finite psubset-card-mono)
    also have ... \leqcard (nodes A) - k using 90 100 by simp
    finally show card (level A w (2*Suc k)j) \leqcard (nodes A) - Suc k by
simp
```

```
    qed
    qed
qed
lemma graph-empty:
    assumes finite (nodes A) w\not\in language A
    shows graph A w (Suc (2 * card (nodes A)))={}
proof -
    obtain n where 1: \bigwedgel. l\geqn\Longrightarrow card (level A w (2 * card (nodes A)) l)=0
        using level-bounded[OF assms(1, 2), of card (nodes A)] by auto
    have graph A w (2* card (nodes A)) =
        (\bigcupl\in{..<n}.level A w (2* card (nodes A)) l) \cup
        (\bigcupl\in{n..}.level A w (2* card (nodes A)) l)
        by auto
    also have (Ul\in{n ..}. level A w (2 * card (nodes A)) l)={}
        using graph-level-finite assms(1) 1 by fastforce
    also have finite ((\bigcupl\in{..<n}. level A w (2* card (nodes A)) l) \cup{})
    using graph-level-finite assms(1) by auto
    finally have 100: finite (graph A w (2 * card (nodes A))) by this
    have 101: finite (greachable A w (graph A w (2* card (nodes A))) v) for v
    using 100 greachable-subset[of A w graph A w (2 * card (nodes A)) v]
    using finite-insert infinite-super by auto
    show ?thesis using 101 by (simp add: clean-def)
qed
lemma graph-le:
    assumes finite (nodes A) w\not\in language A
    assumes v\in graph A wk
    shows k\leq2* card (nodes A)
    using graph-empty graph-antimono assms
    by (metis Suc-leI empty-iff monotone-def not-le-imp-less rev-subsetD)
```


### 3.4 Node Ranks

```
definition rank :: ('label, 'state) nba \(\Rightarrow\) 'label stream \(\Rightarrow\) 'state node \(\Rightarrow\) nat where rank \(A w v \equiv\) GREATEST \(k . v \in \operatorname{graph} A w k\)
lemma rank-member:
assumes finite (nodes A) \(w \notin\) language \(A v \in\) gunodes \(A w\)
shows \(v \in\) graph \(A w(r a n k A w v)\)
unfolding rank-def
proof (rule GreatestI-nat)
show \(v \in\) graph \(A w 0\) using \(\operatorname{assms}(3)\) by simp
show \(k \leq 2 * \operatorname{card}(\) nodes \(A)\) if \(v \in \operatorname{graph} A w k\) for \(k\)
using graph-le assms(1, 2) that by blast
qed
lemma rank-removed:
assumes finite (nodes A) \(w \notin\) language \(A\)
shows \(v \notin\) graph A \(w(\) Suc (rank A w \(v\) ) )
proof
assume \(v \in \operatorname{graph} A w(\operatorname{Suc}(\operatorname{rank} A w v))\)
```

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

## qed

lemma language-ranking:
assumes finite (nodes A) w $\neq$ language $A$
shows ranking $A w(r a n k A w)$
unfolding ranking-def
proof (intro conjI ballI allI impI)
fix $v$
assume 1:v gunodes $A w$
have 2: $v \in$ graph $A w(r a n k A w v)$ using rank-member assms 1 by this
show rank $A w v \leq 2 *$ card (nodes $A$ ) using graph-le assms 2 by this
next
fix $v u$
assume 1: $v \in$ gunodes $A w u \in$ gusuccessors $A w v$
show rank $A w u \leq r a n k A w v$ using rank-le assms 1 by this
next
fix $v$
assume 1: $v \in$ gunodes $A w$ gaccepting $A v$
have 2: $v \in \operatorname{graph} A w(\operatorname{rank} A w v)$ using rank-member assms 1(1) by this
have 3: v $\notin$ graph $A w(S u c(r a n k A w v))$ using rank-removed assms by this
have 4: v $\operatorname{l}$ prune $A w($ graph $A w(\operatorname{rank} A w v))$ using 2 1(2) unfolding prune-def by auto
have 5: graph $A w(\operatorname{Suc}(\operatorname{rank} A w v)) \neq \operatorname{prune} A w(\operatorname{graph} A w(\operatorname{rank} A w v))$ using 34 by blast
show even (rank $A w v$ ) using 5 by auto
next
fix $v r k$
assume 1: $v \in$ gunodes $A w$ gurun $A w r v \operatorname{smap}(\operatorname{rank} A w)($ gtrace $r v)=$ sconst $k$
have sset (gtrace $r v) \subseteq$ gureachable $A w v$
using 1(2) by (metis graph.reachable.reflexive graph.reachable-trace)
then have 6: sset (gtrace $r v) \subseteq$ gunodes $A w$ using 1 (1) by blast
have 60: rank $A w$ 'sset (gtrace r $v$ ) $\subseteq\{k\}$
using $1(3)$ by (metis equalityD1 sset-sconst stream.set-map)
have 50: sset (gtrace $r v$ ) $\subseteq$ graph $A w k$
using rank-member[OF assms] subsetD[OF 6] 60 unfolding image-subset-iff by auto
have 70: grun $A w($ graph $A w k) r v$ using grun-subset 1(2) 50 by this
have 7: sset (gtrace rv) $\subseteq$ clean $A w($ graph $A w k)$
unfolding clean-def using 50 infinite-greachable-gtrace[OF 70] by auto
have 8: sset (gtrace r $v$ ) $\cap$ graph $A w($ Suc $k)=\{ \}$ using rank-removed $[O F$ assms] 60 by blast
have 9: sset (gtrace $r v) \neq\{ \}$ using stream.set-sel(1) by auto
have 10: graph $A w(S u c k) \neq$ clean $A w($ graph $A w k)$ using 789 by blast show odd $k$ using 10 unfolding graph-Suc by auto
qed

### 3.5 Correctness Theorem

theorem language-ranking-iff:
assumes finite (nodes A)
shows $w \notin$ language $A \longleftrightarrow(\exists f$. ranking $A w f)$
using ranking-language language-ranking assms by blast
end

## 4 Complementation

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


### 4.1 Level Rankings and Complementation States

type-synonym 'state lr $=$ 'state $\rightharpoonup$ nat
definition lr-succ :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state $l r \Rightarrow$ 'state lr set where
lr-succ A af $\equiv\{g$.
$\operatorname{dom} g=\bigcup\left(\right.$ transition $\left.A a^{\prime} \operatorname{dom} f\right) \wedge$
$(\forall p \in \operatorname{dom} f . \forall q \in$ transition $A$ a $p$. the $(g q) \leq$ the $(f p)) \wedge$
$(\forall q \in$ dom g. accepting $A q \longrightarrow$ even $($ the $(g q)))\}$
type-synonym 'state st $=$ 'state set
definition st-succ :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state $l r \Rightarrow$ 'state st $\Rightarrow$ 'state st where
st-succ $A$ a $g P \equiv\left\{q \in\right.$ if $P=\{ \}$ then dom $g$ else $\bigcup\left(\right.$ transition $\left.A a^{\prime} P\right)$. even (the $(g q))\}$
type-synonym 'state $c s=$ 'state $l r \times$ 'state st
definition complement-succ :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state $c s \Rightarrow$ 'state cs set where
complement-succ $A a \equiv \lambda(f, P) .\{(g$, st-succ $A$ a $g P) \mid g . g \in l r$-succ A a $f\}$
definition complement :: ('label, 'state) nba $\Rightarrow$ ('label, 'state cs) nba where
complement $A \equiv n b a$ (alphabet A) $(\{$ const $($ Some $(2 *$ card $($ nodes $A))) \mid '$ initial $A\} \times\{\{ \}\})$ (complement-succ A)

$$
(\lambda(f, P) \cdot P=\{ \})
$$

lemma dom-nodes:
assumes $f P \in$ nodes (complement $A$ )
shows dom $(f s t f P) \subseteq$ nodes $A$
using assms unfolding complement-def complement-succ-def lr-succ-def by
(induct) (auto, blast)
lemma ran-nodes:
assumes $f P \in$ nodes (complement $A$ )
shows ran $(f s t f P) \subseteq\{0$.. $2 *$ card $($ nodes $A)\}$
using assms
proof induct
case (initial fP)
show ?case
using initial unfolding complement-def by (auto) (metis eq-refl option.inject ran-restrictD)
next
case (execute fP agQ)
obtain $f P$ where 1: $f P=(f, P)$ by force
have 2: $\operatorname{ran} f \subseteq\{0$.. $2 *$ card (nodes $A$ ) $\}$ using execute(2) unfolding 1 by auto
obtain $a g Q$ where 3: agQ $=(a,(g, Q))$ using prod-cases3 by this
have 4: $p \in \operatorname{dom} f \Longrightarrow q \in$ transition $A$ a $p \Longrightarrow$ the $(g q) \leq$ the $(f p)$ for $p q$ using execute(3)
unfolding 13 complement-def nba.simps complement-succ-def lr-succ-def by $\operatorname{simp}$
have 8: dom $g=\bigcup(($ transition $A$ a)' $(\operatorname{dom} f))$
using execute(3)
unfolding 13 complement-def nba.simps complement-succ-def lr-succ-def
by $\operatorname{simp}$
show ?case
unfolding 13 ran-def
proof safe
fix $q k$
assume 5: fst $(\operatorname{snd}(a,(g, Q))) q=$ Some $k$
have $6: q \in$ dom $g$ using 5 by auto
obtain $p$ where 7: $p \in \operatorname{dom} f q \in \operatorname{transition~} A$ a $p$ using 6 unfolding 8 by auto
have $k=$ the $(g q)$ using 5 by auto
also have $\ldots \leq$ the ( $f p$ ) using 47 by this
also have $\ldots \leq 2 *$ card (nodes $A$ ) using 27 (1) by (simp add: domD ranI subset-eq)
finally show $k \in\{0$.. $2 *$ card $($ nodes $A)\}$ by auto
qed
qed
lemma states-nodes:
assumes $f P \in$ nodes (complement $A$ )
shows snd $f P \subseteq$ nodes $A$
using assms
proof induct
case (initial fP)
show ?case using initial unfolding complement-def by auto
next
case (execute fP agQ)
obtain $f P$ where 1: $f P=(f, P)$ by force
have 2: $P \subseteq$ nodes $A$ using execute(2) unfolding 1 by auto
obtain $\operatorname{ag} Q$ where 3: ag $Q=(a,(g, Q))$ using prod-cases3 by this
have 11: a $\in$ alphabet $A$ using execute(3) unfolding 3 complement-def by auto
have 10: $(g, Q) \in$ nodes (complement $A$ ) using execute $(1,3)$ unfolding 13 by auto
have 4: dom $g \subseteq$ nodes $A$ using dom-nodes[OF 10] by simp
have 5: $\bigcup\left(\right.$ transition $\left.A a^{\prime} P\right) \subseteq$ nodes $A$ using 211 by auto
have 6: $Q \subseteq$ nodes $A$
using execute(3)
unfolding 13 complement-def nba.simps complement-succ-def st-succ-def
using 45
by (auto split: if-splits)
show ?case using 6 unfolding 3 by auto
qed
theorem complement-finite:
assumes finite (nodes A)
shows finite (nodes (complement A))
proof -
let ?lrs $=\{f . \operatorname{dom} f \subseteq$ nodes $A \wedge \operatorname{ran} f \subseteq\{0 . .2 * \operatorname{card}($ nodes $A)\}\}$
have 1: finite ?lrs using finite-set-of-finite-maps' assms by auto
let ?states $=$ Pow $($ nodes $A)$
have 2: finite ?states using assms by simp
have nodes (complement $A$ ) $\subseteq$ ?lrs $\times$ ?states by (force dest: dom-nodes ran-nodes states-nodes)
also have finite ... using 12 by simp

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


### 4.2 Word in Complement Language Implies Ranking

lemma complement-ranking:
assumes $w \in$ language (complement $A$ )
obtains $f$
where ranking $A w f$
proof -
obtain $r p$ where 1:
run (complement $A$ ) $(w \| \mid r) p$
$p \in$ initial (complement A)
infs (accepting (complement A)) ( $p$ \#\# r)
using assms by rule
let ? $m=p \# \# r$
obtain 100:
fst $(? m$ !! Suc $k) \in l r-s u c c A(w!!k)(f s t(? m!!k))$
snd $(? m$ !! Suc $k)=$ st-succ $A(w!!k)(f s t(? m$ !! Suc $k))($ snd $(? m$ !! k) $)$
for $k$ using complement-trace-snth 1 (1) unfolding nba.trace-alt-def szip-smap-snd by metis
define $f$ where $f \equiv \lambda(k, q)$. the $(f s t(? m!!k) q)$
define $P$ where $P k \equiv \operatorname{snd}(? m!!k)$ for $k$
have 2: snd $v \in \operatorname{dom}(f s t(? m!!f s t v))$ if $v \in$ gunodes $A w$ for $v$
using that
proof induct
case (initial v)
then show? case using 1(2) unfolding complement-def by auto
next
case (execute $v u$ )
have snd $u \in \bigcup($ transition $A(w!!$ fst $v)$ ' $\operatorname{dom}(f s t(? m!!f s t v)))$
using execute $(2,3)$ by auto

```
        also have ... = dom(fst (?m !! Suc (fst v)))
            using }100\mathrm{ unfolding lr-succ-def by simp
        also have Suc (fst v)=fst u using execute(3) by auto
        finally show ?case by this
    qed
    have 3: fu\leqfv if 10:v\in gunodes A w and 11:u\ingusuccessors A wv for
uv
    proof -
        have 15: snd u\in transition A (w !! fst v) (snd v) using 11 by auto
        have 16: snd v d dom (fst (?m !! fst v)) using 2 10 by this
        have fu}=\mathrm{ the (fst (?m !! fst u) (snd u)) unfolding f-def by (simp add:
case-prod-beta)
    also have fst u=Suc (fstv) using 11 by auto
    also have the (fst (?m !! ...) (snd u))\leq the (fst (?m !! fst v) (snd v))
            using 100 1516 unfolding lr-succ-def by auto
    also have ... = fv unfolding f-def by (simp add: case-prod-beta)
    finally show fu\leqfv by this
    qed
    have 4:\existsl\geqk. Pl={} for k
    proof -
        have 15: infs ( }\lambda(k,P).P={})?m using 1(3) unfolding complement-de
by auto
    obtain l where 17:l\geqk snd (?m !! l) ={} using 15 unfolding infs-snth
by force
    have 19: Pl={} unfolding P-def using 17 by auto
    show ?thesis using 19 17(1) by auto
qed
show ?thesis
proof (rule that, unfold ranking-def, intro conjI ballI impI allI)
    fix v
    assume v\in gunodes A w
    then show fv\leq2* card (nodes A)
    proof induct
        case (initial v)
        then show ?case using 1(2) unfolding complement-def f-def by auto
    next
        case (execute v u)
        have fu\leqfv using 3[OF execute(1)] execute(3) by simp
        also have ... \leq2 * card (nodes A) using execute(2) by this
        finally show ?case by this
        qed
next
    fix vu
    assume 10:v\ingunodes A w
    assume 11:u G gusuccessors A wv
    show fu\leqfv using 31011 by this
next
    fix v
    assume 10:v\ingunodes A w
```

assume 11: gaccepting $A v$
show even ( $f v$ )
using 10
proof cases
case (initial)
then show ?thesis using 1 (2) unfolding complement-def $f$-def by auto
next
case (execute $u$ )
have 12: snd $v \in \operatorname{dom}(f s t(? m$ !! fst $v))$ using execute graph.nodes.execute 2 by blast
have 12: snd $v \in \operatorname{dom}(f s t(? m$ !! Suc (fst u))) using 12 execute(2) by auto
have 13: accepting $A(s n d v)$ using 11 by auto
have $f v=$ the $(f s t(? m!!f s t v)(s n d v))$ unfolding $f$-def by (simp add: case-prod-beta)
also have $f s t v=S u c(f s t u)$ using execute(2) by auto
also have even (the (fst (?m !! Suc (fst u)) (snd v)))
using 1001213 unfolding lr-succ-def by simp
finally show ?thesis by this
qed
next
fix $v s k$
assume 10: $v \in$ gunodes $A w$
assume 11: gurun $A w s$
assume 12: smap $f($ gtrace s $v)=$ sconst $k$
show odd $k$
proof
assume 13: even $k$
obtain $t u$ where 14: $u \in$ ginitial $A$ gupath $A w t u v=$ gtarget $t u$ using 10 by auto
obtain $l$ where $15: l \geq$ length $t P l=\{ \}$ using 4 by auto
have 30: gurun $A w(t @-s) u$ using 1114 by auto
have 21: fst (gtarget (stake (Suc l) $(t @-s)) u)=$ Suc $l$ for $l$
unfolding sscan-snth[symmetric] using 3014 (1) by (auto elim!: grun-elim) have 17: snd (gtarget (stake $(S u c l+i)(t @-s)) u) \in P(S u c l+i)$ for $i$ proof (induct $i$ )
case (0)
have 20: gtarget (stake (Suc l) $(t$ @-s)) u gunodes A w
using 1411 by (force simp add: 15(1) le-SucI graph.run-stake stake-shift)
have snd (gtarget (stake (Suc l) $(t$ @-s)) u) $\in$
$\operatorname{dom}(f s t(? m$ !! fst (gtarget (stake (Suc l) $(t$ @-s)) u)))
using $2[O F$ 20] by this
also have fst (gtarget (stake (Suc l) ( $t$ @ $-s)$ ) u) $=$ Suc l using 21 by this
finally have 22: snd (gtarget (stake (Suc l) (t @-s)) u) $\in \operatorname{dom}(f s t(? m$ !! Suc l)) by this
have gtarget $($ stake $(S u c l)(t @-s)) u=\operatorname{gtrace}(t @-s) u!!l$ unfolding sscan-snth by rule
also have $\ldots=$ gtrace s $v!!(l-$ length $t)$ using $15(1)$ by simp
also have $f \ldots=\operatorname{smap} f($ gtrace $s v)!!(l-$ length $t)$ by $\operatorname{simp}$
also have smap $f(g t r a c e ~ s v)=$ sconst $k$ unfolding 12 by rule
also have sconst $k!!(l-$ length $t)=k$ by $\operatorname{simp}$
finally have 23: even $(f$ (gtarget (stake (Suc $l)(t$ @-s)) u)) using 13 by $\operatorname{simp}$
have snd (gtarget (stake (Suc l) $(t$ @-s)) u) $\in$
$\{p \in \operatorname{dom}(f s t(? m!!$ Suc $l))$. even $(f($ Suc $l, p))\}$
using 212223 by (metis (mono-tags, lifting) mem-Collect-eq prod.collapse)
also have $\ldots=$ st-succ $A(w!!l)(f s t(? m!!S u c l))(P l)$
unfolding $15(2)$ st-succ-def $f$-def by simp
also have $\ldots=P$ (Suc $l$ ) using 100 (2) unfolding $P$-def by rule
finally show ?case by auto
next
case (Suc i)
have 20: $P($ Suc $l+i) \neq\{ \}$ using Suc by auto
have 21: fst (gtarget (stake (Suc l + Suc i) ( $t$ @-s)) u) = Suc $l+$ Suc $i$ using 21 by (simp add: stake-shift)
have gtarget (stake (Suc l+Suc i) (t @-s)) u=gtrace (t @-s)u!! (l + Suc i)
unfolding sscan-snth by simp
also have $\ldots \in$ gusuccessors $A w($ gtarget $(\operatorname{stake}(S u c(l+i))(t @-s))$
u)
using graph.run-snth[OF 30, of $l+$ Suc $i]$ by simp
finally have 220: snd (gtarget (stake (Suc (Suc l+i)) $(t$ @-s)) u) $\in$ transition $A(w!!(S u c l+i))($ snd $($ gtarget $($ stake $(S u c(l+i))(t @-$
s)) $u$ )
using 21 by auto
have 22: snd (gtarget (stake (Suc l+Suc i) (t @-s)) u) $\in$ $\cup\left(\right.$ transition $\left.A(w!!(S u c l+i)){ }^{\prime} P(S u c l+i)\right)$ using 220 Suc by auto
have gtarget (stake $($ Suc $l+S u c i)(t @-s)) u=\operatorname{gtrace}(t @-s) u!!(l$ + Suc i)
unfolding sscan-snth by simp
also have $\ldots=$ gtrace s $v!!(l+$ Suc $i$ - length $t)$ using 15(1)
by (metis add.commute shift-snth-ge sscan-const trans-le-add2)
also have $f \ldots=\operatorname{smap} f($ gtrace $s v)!!(l+$ Suc $i-$ length $t)$ by simp
also have smap $f($ gtrace $s v)=$ sconst $k$ unfolding 12 by rule
also have sconst $k!!(l+$ Suc $i-$ length $t)=k$ by simp
finally have 23: even ( $f$ (gtarget (stake (Suc l+Suc i) ( $t$ @-s)) u))
using 13 by auto
have snd (gtarget (stake $($ Suc $l+S u c i)(t @-s)) u) \in$
$\{p \in \bigcup$ (transition $A(w!!($ Suc $l+i))$ ' $P($ Suc $l+i))$. even $(f(S u c$ $(S u c l+i), p))\}$
using 212223 by (metis (mono-tags) add-Suc-right mem-Collect-eq prod.collapse)
also have $\ldots=$ st-succ $A(w!!($ Suc $l+i))(f s t(? m$ !! Suc $($ Suc $l+i)))$ (P(Sucl $+i)$ )
unfolding st-succ-def $f$-def using 20 by simp
also have $\ldots=P(S u c(S u c l+i))$ unfolding $100(2)[$ folded $P$-def] by rule

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


### 4.3 Ranking Implies Word in Complement Language

```
definition reach where
    reach \(A w i \equiv\{\) target \(r p \mid r p\). path \(A r p \wedge p \in\) initial \(A \wedge\) map fst \(r=\) stake
\(i w\}\)
```

lemma reach- $0[$ simp $]$ : reach $A$ w $0=$ initial $A$ unfolding reach-def by auto lemma reach-Suc-empty:
assumes $w!!n \notin$ alphabet $A$
shows reach $A w($ Suc $n)=\{ \}$
proof safe
fix $q$
assume 1: $q \in$ reach $A w($ Suc $n)$
obtain $r p$ where 2: $q=$ target $r p$ path $A r p p \in \operatorname{initial} A$ map fst $r=$ stake
(Suc n) w
using 1 unfolding reach-def by blast
have 3: path A (take n r @ drop n r) p using 2(2) by simp
have 4: map fst $r=$ stake $n w$ @ $[w!!n]$ using 2(4) stake-Suc by auto
have 5: map snd $r=$ take n (map snd $r$ ) @ [q] using 2(1, 4) 4
by (metis One-nat-def Suc-inject Suc-neq-Zero Suc-pred append.right-neutral append-eq-conv-conj drop-map id-take-nth-drop last-ConsR last-conv-nth
length-0-conv
length-map length-stake lessI nba.target-alt-def nba.states-alt-def zero-less-Suc)
have 6: drop $n r=[(w!!n, q)]$ using 45
by (metis append-eq-conv-conj append-is-Nil-conv append-take-drop-id drop-map length-greater-0-conv length-stake stake-cycle-le stake-invert-Nil take-map zip-Cons-Cons zip-map-fst-snd)
show $q \in\}$ using assms 3 unfolding 6 by auto
qed
lemma reach-Suc-succ:
assumes $w!!n \in$ alphabet $A$
shows reach $A w($ Suc $n)=\bigcup(\operatorname{transition~} A(w!!n)$ ‘reach $A w n)$
proof safe
fix $q$
assume 1: $q \in$ reach $A w($ Suc $n)$
obtain $r p$ where 2: $q=$ target $r p$ path $A r p p \in \operatorname{initial} A$ map fst $r=$ stake
(Suc n) w
using 1 unfolding reach-def by blast
have 3: path $A$ (take $n r$ @ drop $n r$ ) pusing 2(2) by simp
have 4: map fst $r=$ stake $n w$ @ $[w!!n]$ using 2(4) stake-Suc by auto
have 5: map snd $r=$ take $n($ map snd $r) @[q]$ using 2(1, 4$) 4$
by (metis One-nat-def Suc-inject Suc-neq-Zero Suc-pred append.right-neutral append-eq-conv-conj drop-map id-take-nth-drop last-ConsR last-conv-nth length-0-conv
length-map length-stake lessI nba.target-alt-def nba.states-alt-def zero-less-Suc)
have 6: drop $n r=[(w!!n, q)]$ using 45
by (metis append-eq-conv-conj append-is-Nil-conv append-take-drop-id drop-map length-greater-0-conv length-stake stake-cycle-le stake-invert-Nil take-map zip-Cons-Cons zip-map-fst-snd)
show $q \in \bigcup(($ transition $A(w!!n)$ '(reach $A w n)))$
unfolding reach-def
proof (intro UN-I CollectI exI conjI)
show target (take $n r$ ) $p=\operatorname{target}$ (take $n r$ ) $p$ by rule
show path $A$ (take $n r$ ) $p$ using 3 by blast
show $p \in \operatorname{initial} A$ using 2(3) by this
show map fst (take n r) = stake $n w$ using 2 by (metis length-stake lessI nat.distinct(1)
stake-cycle-le stake-invert-Nil take-map take-stake)
show $q \in$ transition $A(w!!n)($ target (take $n r) p$ ) using 3 unfolding 6
by auto
qed
next
fix $p q$
assume 1: $p \in$ reach $A w n q \in \operatorname{transition~} A(w!!n) p$
obtain $r x$ where 2: $p=$ target $r x$ path $A r x x \in$ initial $A$ map fst $r=$ stake $n$ w
using 1 (1) unfolding reach-def by blast
show $q \in$ reach $A w($ Suc $n)$
unfolding reach-def
proof (intro CollectI exI conjI)
show $q=\operatorname{target}(r$ @ $[(w!!n, q)]) x$ using 12 by auto
show path $A(r$ @ $[(w!!n, q)]) x$ using assms 1(2) 2(1, 2) by auto
show $x \in$ initial $A$ using 2(3) by this
show map fst $(r$ @ $[(w!!n, q)])=$ stake (Suc n) w using 12
by (metis eq-fst-iff list.simps(8) list.simps (9) map-append stake-Suc)
qed
qed
lemma reach-Suc[simp]: reach $A w($ Suc $n)=($ if $w!!n \in$ alphabet $A$
then $\bigcup$ (transition $A(w!!n)$ ' reach $A w n)$ else $\})$
using reach-Suc-empty reach-Suc-succ by metis
lemma reach-nodes: reach $A w i \subseteq$ nodes $A$ by (induct $i$ ) (auto)
lemma reach-gunodes: $\{i\} \times$ reach $A w i \subseteq$ gunodes $A w$
by (induct i) (auto intro: graph.nodes.execute)
lemma ranking-complement:
assumes finite (nodes A) w streams (alphabet A) ranking Awf
shows $w \in$ language (complement $A$ )
proof -
define $f^{\prime}$ where $f^{\prime} \equiv \lambda(k, p)$. if $k=0$ then $2 *$ card (nodes $A$ ) else $f(k, p)$

```
    have 0: ranking A w f
    unfolding ranking-def
    proof (intro conjI ballI impI allI)
        show \ \.v\in gunodes A w\Longrightarrowf'v\leq2*\operatorname{card (nodes A)}
        using assms(3) unfolding ranking-def f'-def by auto
        show }\vu.v\ingunodes Aw\Longrightarrowu\ingusuccessors Awv\Longrightarrow和u\leqf'
        using assms(3) unfolding ranking-def f'-def by fastforce
    show }\v.v\in\mathrm{ gunodes }Aw\Longrightarrow\mathrm{ gaccepting A v # even (f'v)
        using assms(3) unfolding ranking-def f'-def by auto
    next
    have 1:v\ingunodes A w\Longrightarrowgurun A wrv\Longrightarrowsmap f(gtrace r v)=sconst
k\Longrightarrow odd k
        for vrk using assms(3) unfolding ranking-def by meson
        fix vr 
        assume 2: v G gunodes A w gurun A wrv smap f'(gtrace r v)= sconst k
        have 20: shd r\in gureachable A wv using 2
            by (auto) (metis graph.reachable.reflexive graph.reachable-trace gtrace-alt-def
subsetD shd-sset)
        obtain 3:
            shd r gunodes A w
        gurun A w (stl r) (shd r)
        smap f' (gtrace (stl r) (shd r)) = sconst k
    using 2 20 by (metis (no-types, lifting) eq-id-iff graph.nodes-trans graph.run-scons-elim
                siterate.simps(2) sscan.simps(2) stream.collapse stream.map-sel(2))
    have 4:k\not=0 if (k,p)\in sset r for kp
    proof -
        obtain ra ka pa where 1:r= fromN (Suc ka)|| rav=(ka,pa)
            using grun-elim[OF 2(2)] by this
        have 2: k fsset (fromN (Suc ka)) using 1(1) that
            by (metis image-eqI prod.sel(1) szip-smap-fst stream.set-map)
        show ?thesis using 2 by simp
    qed
    have 5: smap f' (gtrace (stl r) (shd r)) = smap f (gtrace (stl r) (shd r))
    proof (rule stream.map-cong)
        show gtrace (stl r) (shd r) = gtrace (stl r) (shd r) by rule
        next
            fix z
            assume 1:z\in sset (gtrace (stl r) (shd r))
            have 2: fst z}\not=0\mathrm{ using 4 1 by (metis gtrace-alt-def prod.collapse stl-sset)
            show f'z=fzusing 2 unfolding f'-def by (auto simp: case-prod-beta)
        qed
        show odd k using 135 by simp
    qed
    define g}\mathrm{ where g i p 三if pG reach A wi then Some (f'(i,p)) else None for
i p
    have g-dom[simp]: dom (gi) = reach A wi for i
        unfolding g-def by (auto) (metis option.simps(3))
    have g-0[simp]: g 0 = const (Some (2 * card (nodes A)))|`initial A
```

unfolding $g$-def $f^{\prime}$-def by auto
have $g$-Suc $[\operatorname{simp}]: g($ Suc $n) \in l r$-succ $A(w!!n)(g n)$ for $n$
unfolding lr-succ-def
proof (intro CollectI conjI ballI impI)
show $\operatorname{dom}(g(S u c n))=\bigcup($ transition $A(w!!n)$ 'dom $(g n))$ using snth-in assms(2) by auto
next
fix $p q$
assume 100: $p \in \operatorname{dom}(g n) q \in \operatorname{transition~} A(w!!n) p$
have 101: $q \in$ reach $A w$ (Suc n) using snth-in assms(2) 100 by auto
have 102: $(n, p) \in$ gunodes A $w$ using $100(1)$ reach-gunodes $g$-dom by blast
have 103: (Suc $n, q) \in$ gusuccessors $A w(n, p)$ using snth-in assms(2) 102 100(2) by auto
have 104: $p \in$ reach $A w n$ using $100(1)$ by simp
have $g($ Suc $n) q=S o m e\left(f^{\prime}(S u c n, q)\right)$ using 101 unfolding $g$-def by simp
also have the $\ldots=f^{\prime}($ Suc $n, q)$ by simp
also have $\ldots \leq f^{\prime}(n, p)$ using 0 unfolding ranking-def using 102103 by simp
also have $\ldots=$ the $\left(\operatorname{Some}\left(f^{\prime}(n, p)\right)\right)$ by simp also have Some $\left(f^{\prime}(n, p)\right)=g n p$ using 104 unfolding $g$-def by simp finally show the $(g(S u c n) q) \leq$ the $(g n p)$ by this
next
fix $p$
assume 100: $p \in \operatorname{dom}(g($ Suc $n))$ accepting A $p$
have 101: $p \in$ reach $A w$ (Suc n) using 100 (1) by simp
have 102: (Suc $n, p) \in$ gunodes $A$ w using 101 reach-gunodes by blast have 103: gaccepting $A$ (Suc n, p) using 100(2) by simp
have the $(g$ (Suc $n) p)=f^{\prime}($ Suc $n, p)$ using 101 unfolding $g$-def by simp also have even ... using 0 unfolding ranking-def using 102103 by auto finally show even (the $(g(S u c n) p)$ ) by this
qed
define $P$ where $P \equiv$ rec-nat $\}(\lambda n$. st-succ $A(w!!n)(g($ Suc $n)))$
have $P-0[$ simp $]: P 0=\{ \}$ unfolding $P$-def by simp
have $P$-Suc $[$ simp $]$ : $P($ Suc $n)=\operatorname{st-succ} A(w!!n)(g(S u c n))(P n)$ for $n$ unfolding $P$-def by simp
have $P$-reach: $P n \subseteq$ reach $A w n$ for $n$
using snth-in assms(2) by (induct n) (auto simp add: st-succ-def)
have $P n \subseteq$ reach $A w n$ for $n$ using $P$-reach by auto
also have $\ldots n \subseteq$ nodes $A$ for $n$ using reach-nodes by this
also have finite (nodes A) using assms(1) by this
finally have $P$-finite: finite $(P n)$ for $n$ by this
define $s$ where $s \equiv$ smap $g$ nats $\|\|$ smap $P$ nats
show ?thesis
proof
show run (complement $A)(w||\mid s t l s)(s h d s)$
proof (intro nba.snth-run conjI, simp-all del: stake.simps stake-szip)
fix $k$
show $w!!k \in$ alphabet (complement A) using snth-in assms(2) unfolding complement-def by auto
have stl $s!!k=s!!$ Suc $k$ by $\operatorname{simp}$
also have $\ldots \in$ complement-succ $A(w!!k)(s!!k)$
unfolding complement-succ-def $s$-def using $P$-Suc by simp
also have $\ldots=$ complement-succ $A(w!!k)(\operatorname{target}($ stake $k(w \mid \|$ stl s)) (shd s))
unfolding sscan-scons-snth[symmetric] nba.trace-alt-def by simp
also have $\ldots=$ transition (complement $A$ ) ( $w!!k$ ) (target (stake $k$ ( $w \|$
stl s)) (shd s))
unfolding complement-def nba.sel by rule
finally show stl $s!!k \in$
transition (complement $A)(w!!k)($ target $(\operatorname{stake} k(w||\mid ~ s t l ~ s))(s h d s))$
by this
qed
show shd $s \in$ initial (complement A) unfolding complement-def s-def using $P-0$ by simp
show infs (accepting (complement A)) (shd s\#\# stl s)
proof -
have 10: $\forall n . \exists k \geq n . P k=\{ \}$
proof (rule ccontr)
assume 20: $\neg(\forall n . \exists k \geq n . P k=\{ \})$
obtain $k$ where 22: $P(k+n) \neq\{ \}$ for $n$ using 20 using le-add1 by blast
define $m$ where $m n S \equiv\{p \in \bigcup$ (transition $A(w!!n)$ ' $S$ ). even (the ( $g($ Suc $n) p)$ ) $\}$ for $n S$
define $R$ where $R$ in $S \equiv$ rec-nat $S(\lambda i . m(n+i))$ ifor $i n S$
have $R-O[$ simp $]: R 0 n=i d$ for $n$ unfolding $R$-def by auto
have $R$-Suc [simp]: $R$ (Suc $i$ ) $n=m(n+i) \circ R i n$ for $i n$ unfolding $R$-def by auto
have $R$-Suc': $R($ Suc $i) n=R i(S u c n) \circ m n$ for $i n$ unfolding $R$-Suc by (induct $i$ ) (auto)
have $R$-reach: $R$ in $S \subseteq$ reach $A w(n+i)$ if $S \subseteq$ reach $A w n$ for $i n S$ using snth-in assms(2) that m-def by (induct i) (auto)
have $P-R: P(k+i)=R i k(P k)$ for $i$
using 22 by (induct $i$ ) (auto simp add: case-prod-beta' m-def st-succ-def)
have 50: $R$ in $S=(\bigcup p \in S . R$ in $\{p\})$ for $i n S$
by (induct i) (auto simp add: m-def prod.case-eq-if)
have 51: $R(i+j) n S=\{ \}$ if $R i n S=\{ \}$ for $i j n S$
using that by (induct $j$ ) (auto simp add: m-def prod.case-eq-if)
have 52: $R j n S=\{ \}$ if $i \leq j R i n S=\{ \}$ for $i j n S$
using 51 by (metis le-add-diff-inverse that(1) that(2))
have 1: $\exists p \in S . \forall$ i. $R$ in $\{p\} \neq\{ \}$
if assms: finite $S \wedge i . R$ in $S \neq\{ \}$ for $n S$
proof (rule ccontr)
assume 1: $\neg(\exists p \in S . \forall$ i. $R$ in $\{p\} \neq\{ \})$
obtain $f$ where $3: \wedge p . p \in S \Longrightarrow R(f p) n\{p\}=\{ \}$ using 1 by metis have $4: R\left(S u p\left(f^{\prime} S\right)\right) n\{p\}=\{ \}$ if $p \in S$ for $p$
proof (rule 52)
show $f p \leq S u p(f$ ' $S$ ) using assms(1) that by (auto intro: le-cSup-finite)
show $R(f p) n\{p\}=\{ \}$ using 3 that by this
qed
have $R(S u p(f ‘ S)) n S=(\bigcup p \in S . R(S u p(f$ ' $S)) n\{p\})$ using 50
by this
also have $\ldots=\{ \}$ using 4 by simp
finally have $5: R\left(S u p\left(f^{\prime} S\right)\right) n S=\{ \}$ by this
show False using that(2) 5 by auto
qed
have 2: $\wedge$ i. R $i(k+0)(P k) \neq\{ \}$ using $22 P-R$ by simp
obtain $p$ where 3: $p \in P k \wedge i$. $R i k\{p\} \neq\{ \}$ using 1 [OF P-finite 2] by auto
define $Q$ where $Q n p \equiv(\forall i . R i(k+n)\{p\} \neq\{ \}) \wedge p \in P(k+n)$

## for $n p$

have 5: $\exists q \in$ transition $A(w!!(k+n)) p . Q(S u c n) q$ if $Q n p$ for $n p$ proof -
have 11: $p \in P(k+n) \bigwedge i . R i(k+n)\{p\} \neq\{ \}$ using that unfolding $Q$-def by auto
have 12: $R($ Suc $i)(k+n)\{p\} \neq\{ \}$ for $i$ using 11(2) by this
have 13: $R i(k+$ Suc $n)(m(k+n)\{p\}) \neq\{ \}$ for $i$ using 12 unfolding $R$-Suc ${ }^{\prime}$ by simp
have $\{p\} \subseteq P(k+n)$ using $11(1)$ by auto
also have $\ldots \subseteq$ reach $A w(k+n)$ using $P$-reach by this
finally have $R 1(k+n)\{p\} \subseteq$ reach $A w(k+n+1)$ using $R$-reach by blast
also have $\ldots \subseteq$ nodes $A$ using reach-nodes by this
also have finite (nodes $A$ ) using assms(1) by this
finally have 14: finite $(m(k+n)\{p\})$ by simp
obtain $q$ where 14: $q \in m(k+n)\{p\} \bigwedge i . R i(k+$ Suc $n)\{q\} \neq\{ \}$
using $1[O F 1413]$ by auto
show ?thesis
unfolding $Q$-def prod.case
proof (intro bexI conjI allI)
show $\wedge i . R i(k+S u c n)\{q\} \neq\{ \}$ using 14(2) by this
show $q \in P(k+$ Suc $n)$
using 14 (1) 11(1) 22 unfolding $m$-def by (auto simp add: st-succ-def)
show $q \in$ transition $A(w!!(k+n)) p$ using $14(1)$ unfolding $m$-def by $\operatorname{simp}$
qed
qed
obtain $r$ where 23:
run $A \operatorname{rp}$ ^i. $Q i((p \# \#$ trace $r p)!!i) \bigwedge i . f s t(r!!i)=w!!(k+i)$
proof (rule nba.invariant-run-index[of $Q 0 p A \lambda n p a . f s t a=w!!(k+$ n)])
show $Q 0 p$ unfolding $Q$-def using 3 by auto
show $\exists a$. (fst $a \in$ alphabet $A \wedge$ snd $a \in \operatorname{transition~} A($ fst $a) p) \wedge$ $Q($ Suc $n)($ snd $a) \wedge f s t a=w!!(k+n)$ if $Q n p$ for $n p$ using snth-in assms(2) 5 that by fastforce
qed auto
have 20: smap fst $r=$ sdrop $k w$ using 23(3) by (intro eqI-snth) (simp add: case-prod-beta)
have 21: $(p \# \#$ smap snd $r)!!i \in P(k+i)$ for $i$
using 23(2) unfolding $Q$-def unfolding nba.trace-alt-def by simp
obtain $r$ where 23: run $A($ sdrop $k w||\mid$ stl $r)($ shd $r) \bigwedge i . r!!i \in P(k$
using 2021 23(1) by (metis stream.sel(1) stream.sel(2) szip-smap)
let $? v=(k, s h d r)$
let $? r=$ from $N($ Suc $k)\|\|$ stl $r$
have shd $r=r$ !! 0 by simp
also have $\ldots \in P k$ using 23(2)[of 0] by simp
also have $\ldots \subseteq$ reach $A w k$ using $P$-reach by this
finally have 24: ?v $\in$ gunodes $A w$ using reach-gunodes by blast
have 25: gurun $A w$ ?r ?v using run-grun 23(1) by this
obtain $l$ where 26: Ball (sset (smap f' (gtrace (sdrop l ?r) (gtarget (stake $l$ ? $r$ ) ?v) )) ) odd
using ranking-stuck-odd 02425 by this
have 27: $f^{\prime}($ Suc $(k+l), r$ !! Suc $l)=$
shd (smap $f^{\prime}$ (gtrace (sdrop l ?r) (gtarget (stake l ?r) ?v))) by (simp add:
algebra-simps)
also have $\ldots \in \operatorname{sset}\left(\operatorname{smap} f^{\prime}(\right.$ gtrace $($ sdrop $l$ ? $r)$ (gtarget (stake l ? $r$ )
?v))
using shd-sset by this
finally have 28: odd ( $f^{\prime}(S u c(k+l), r$ !! Suc $l)$ ) using 26 by auto
have $r!!$ Suc $l \in P($ Suc $(k+l))$ using 23(2) by (metis add-Suc-right)
also have $\ldots=\{p \in \bigcup$ (transition $A(w!!(k+l))$ ' $P(k+l))$.
even (the $(g(S u c(k+l)) p))\}$ using 23(2) by (auto simp: st-succ-def)
also have $\ldots \subseteq\{p$. even $($ the $(g(S u c(k+l)) p))\}$ by auto
finally have 29: even (the $(g(S u c(k+l))(r$ !! Suc $l))$ ) by auto
have 30:r !! Suc $l \in$ reach $A w($ Suc $(k+l))$
using 23(2) P-reach by (metis add-Suc-right subsetCE)
have 31: even $\left(f^{\prime}(\right.$ Suc $(k+l), r$ !! Suc $l)$ ) using 2930 unfolding $g$-def by $\operatorname{simp}$
show False using 2831 by simp
qed
have 11: infs ( $\lambda k . P k=\{ \})$ nats using 10 unfolding infs-snth by simp
have infs $(\lambda S . S=\{ \})($ smap snd (smap g nats ||| smap $P$ nats))
using 11 by (simp add: comp-def)
then have infs $(\lambda x$. snd $x=\{ \})$ (smap $g$ nats ||| smap $P$ nats)
by (simp add: comp-def del: szip-smap-snd)
then have infs $(\lambda(f, P) . P=\{ \})$ (smap g nats ||| smap $P$ nats)
by (simp add: case-prod-beta ${ }^{\prime}$ )
then have $\operatorname{infs}(\lambda(f, P) . P=\{ \})($ stl (smap g nats \||| smap $P$ nats $)$ ) by blast

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


### 4.4 Correctness Theorem

```
theorem complement-language:
    assumes finite (nodes A)
    shows language (complement A) = streams (alphabet A) - language A
proof (safe del: notI)
    have 1: alphabet (complement A) = alphabet A unfolding complement-def
nba.sel by rule
    show }w\in\mathrm{ streams (alphabet A) if w}\mathrm{ (language (complement A) for w
        using nba.language-alphabet that 1 by force
    show }w\not\in\mathrm{ language A if w}\in\mathrm{ language (complement A) for w
        using complement-ranking ranking-language that by metis
    show w\in language (complement A) if w\in streams (alphabet A) w\not\in language
A for w
        using language-ranking ranking-complement assms that by blast
    qed
```

end

## 5 Complementation Implementation

theory Complementation-Implement
imports
Transition-Systems-and-Automata.NBA-Implement
Complementation
begin
unbundle lattice-syntax
type-synonym item $=$ nat $\times$ bool
type-synonym 'state items $=$ 'state $\rightharpoonup$ item
type-synonym state $=($ nat $\times$ item $)$ list
abbreviation item-rel $\equiv$ nat-rel $\times_{r}$ bool-rel
abbreviation state-rel $\equiv\langle n a t-r e l$, item-rel $\rangle$ list-map-rel
abbreviation pred $A$ a $q \equiv\{p . q \in$ transition $A$ a $p\}$

### 5.1 Phase 1

definition $c s-l r$ :: 'state items $\Rightarrow$ 'state $l r$ where

$$
c s-l r f \equiv \text { map-option fst } \circ f
$$

definition cs-st :: 'state items $\Rightarrow$ 'state st where
cs-st $f \equiv f-$ 'Some'snd -' $\{$ True $\}$
abbreviation cs-abs :: 'state items $\Rightarrow$ 'state cs where
cs-abs $f \equiv(c s-l r f, c s-s t f)$
definition cs-rep :: 'state cs $\Rightarrow$ 'state items where
$c s$-rep $\equiv \lambda(g, P) p$. map-option $(\lambda k .(k, p \in P))(g p)$
lemma cs-abs-rep[simp]: cs-rep $(c s-a b s f)=f$
proof
show cs-rep (cs-absf) $x=f x$ for $x$
unfolding $c s$-lr-def cs-st-def cs-rep-def by (cases f $x$ ) (force+)

## qed

lemma $c s$-rep-lr $[\operatorname{simp}]: c s-l r(c s-r e p ~(g, P))=g$
proof
show cs-lr (cs-rep $(g, P)) x=g x$ for $x$
unfolding cs-rep-def cs-lr-def by (cases g $x$ ) (auto)
qed
lemma cs-rep-st $[$ simp $]$ :cs-st (cs-rep $(g, P))=P \cap$ dom $g$
unfolding $c s$-rep-def cs-st-def by force
lemma $c s-l r-\operatorname{dom}[s i m p]: \operatorname{dom}(c s-l r f)=\operatorname{dom} f$ unfolding $c s-l r-d e f$ by $\operatorname{simp}$
lemma cs-lr-apply[simp]:
assumes $p \in \operatorname{dom} f$
shows the (cs-lr fp) $=$ fst (the $(f p)$ )
using assms unfolding $c s-l r$-def by auto
lemma cs-rep-dom[simp]: dom (cs-rep $(g, P))=\operatorname{dom} g$ unfolding cs-rep-def by auto
lemma cs-rep-apply[simp]:
assumes $p \in \operatorname{dom} f$
shows $f$ st (the $(c s$-rep $(f, P) p))=$ the $(f p)$
using assms unfolding $c s$-rep-def by auto
abbreviation cs-rel :: ('state items $\times$ 'state cs) set where $c s-r e l \equiv b r c s-a b s t o p$
lemma cs-rel-inv-single-valued: single-valued (cs-rel ${ }^{-1}$ )
by (auto intro!: inj-onI) (metis cs-abs-rep)
definition refresh-1 :: 'state items $\Rightarrow$ 'state items where
refresh-1 $f \equiv$ if True $\in$ snd'ran $f$ then $f$ else map-option (apsnd top) $\circ f$
definition ranks-1 ::
('label, 'state) $n b a \Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where
ranks-1 A a $f \equiv\{g$.
$\operatorname{dom} g=\bigcup(($ transition $A a) ‘(\operatorname{dom} f)) \wedge$
$(\forall p \in \operatorname{dom} f . \forall q \in \operatorname{transition~A~a~p.fst~}($ the $(g q)) \leq f s t($ the $(f p))) \wedge$
$(\forall q \in$ dom $g$. accepting $A q \longrightarrow$ even $($ fst $($ the $(g q)))) \wedge$
cs-st $g=\{q \in \bigcup(($ transition $A a)$ '(cs-st $f))$. even $(f s t($ the $(g q)))\}\}$
definition complement-succ-1 ::
('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where complement-succ-1 A $a=$ ranks-1 A $a \circ$ refresh-1
definition complement-1 :: ('label, 'state) $n b a \Rightarrow$ ('label, 'state items) nba where complement- $1 A \equiv n b a$
(alphabet A)
$\left(\left\{\right.\right.$ const $\left.($ Some $(2 *$ card (nodes A), False $))\right|^{‘}$ initial $\left.\left.A\right\}\right)$
(complement-succ-1 A)
$(\lambda f . c s-s t f=\{ \})$
lemma refresh-1-dom[simp]: dom (refresh-1 f) $=\operatorname{dom} f$ unfolding refresh-1-def by $\operatorname{simp}$
lemma refresh-1-apply $[$ simp $]:$ fst (the (refresh-1 $f$ p $)$ ) $=$ fst $($ the $(f p))$ unfolding refresh-1-def by (cases $f$ p) (auto)
lemma refresh-1-cs-st[simp]: cs-st (refresh-1f) $=($ if cs-st $f=\{ \}$ then dom $f$ else cs-st f)
unfolding refresh-1-def cs-st-def ran-def image-def vimage-def by auto
lemma complement-succ-1-abs:
assumes $g \in$ complement-succ-1 $A$ a $f$
shows cs-abs $g \in$ complement-succ $A$ a (cs-abs $f$ )
unfolding complement-succ-def
proof (simp, rule)
have 1 :
dom $g=\bigcup(($ transition $A a)$ ' $(\operatorname{dom} f))$
$\forall p \in \operatorname{dom} f . \forall q \in$ transition $A$ a $p$.fst $($ the $(g q)) \leq f s t($ the $(f p))$
$\forall p \in$ dom $g$. accepting $A p \longrightarrow$ even (fst $($ the $(g p))$ )
using assms unfolding complement-succ-1-def ranks-1-def by simp-all
show cs-lr $g \in l r$-succ $A$ a (cs-lr $f$ )
unfolding $l r$-succ-def
proof (intro CollectI conjI ballI impI)
show $\operatorname{dom}(c s-l r g)=\bigcup\left(\right.$ transition $\left.A a^{\prime} \operatorname{dom}(c s-l r f)\right)$ using 1 by simp
next
fix $p q$
assume 2: $p \in \operatorname{dom}(c s-l r f) q \in \operatorname{transition~} A$ a $p$
have 3: $q \in \operatorname{dom}(c s-l r g)$ using 12 by auto
show the $(c s-l r g q) \leq$ the (cs-lr $f p$ ) using 123 by simp
next
fix $p$
assume 2: $p \in \operatorname{dom}(c s-l r g)$ accepting $A p$
show even (the (cs-lr g $p$ )) using 12 by auto
qed
have 2: cs-st $g=\left\{q \in \bigcup\right.$ (transition $A a^{\prime}$ cs-st (refresh-1 f)). even (fst (the $(g q)))\}$
using assms unfolding complement-succ-1-def ranks-1-def by simp
show cs-st $g=s t$-succ $A a(c s-l r g)(c s-s t f)$
proof (cases cs-st $f=\{ \}$ )
case True
have 3: the $(\operatorname{cs-lr} g q)=$ fst $($ the $(g q))$ if $q \in \bigcup((\operatorname{transition~Aa)'(\operatorname {dom}f))}$
for $q$
using that 1(1) by simp
show ?thesis using 23 unfolding st-succ-def refresh-1-cs-st True cs-lr-dom 1(1) by force
next
case False
have 3: the $(\operatorname{cs-lr} g q)=f s t($ the $(g q))$ if $q \in \bigcup(($ transition $A a)$ ' $(c s-s t f))$ for $q$
using that 1(1) by (auto intro!: cs-lr-apply) (metis IntE UN-iff cs-abs-rep cs-lr-dom cs-rep-st domD prod.collapse)
have $c s$-st $g=\{q \in \bigcup$ (transition $A a$ 'cs-st (refresh-1 $f)$ ). even (fst (the ( $g$ q))) $\}$
using 2 by this
also have $c s$-st (refresh-1 f) $=c s$-st $f$ using False by simp
also have $\{q \in \bigcup(($ transition $A a)$ ' $(c s$-st $f))$. even $(f s t($ the $(g q)))\}=$
$\{q \in \bigcup(($ transition $A a)$ ' $(c s$-st $f))$. even (the $(c s-l r g q))\}$ using 3 by metis
also have $\ldots=$ st-succ $A$ a (cs-lr g) (cs-st f) unfolding st-succ-def using False by simp
finally show ?thesis by this
qed
qed
lemma complement-succ-1-rep:
assumes $P \subseteq \operatorname{dom} f(g, Q) \in$ complement-succ $A a(f, P)$
shows cs-rep $(g, Q) \in$ complement-succ-1 A a (cs-rep $(f, P))$
unfolding complement-succ-1-def ranks-1-def comp-apply
proof (intro CollectI conjI ballI impI)
have 1 :
$\operatorname{dom} g=\bigcup(($ transition $A a)$ ' $(\operatorname{dom} f))$
$\forall p \in \operatorname{dom} f . \forall q \in$ transition $A$ a $p$. the $(g q) \leq$ the $(f p)$
$\forall p \in$ dom $g$. accepting $A p \longrightarrow$ even (the ( $g p)$ )
using assms(2) unfolding complement-succ-def lr-succ-def by simp-all
have 2: $Q=\{q \in$ if $P=\{ \}$ then dom $g$ else $\bigcup(($ transition $A a)$ ' $P)$. even (the $(g q))\}$
using assms(2) unfolding complement-succ-def st-succ-def by simp
have 3: $Q \subseteq$ dom $g$ unfolding 21 (1) using assms(1) by auto
show dom (cs-rep $(g, Q))=\bigcup$ (transition A a'dom (refresh-1 (cs-rep ( $f$, $P)$ )) ) using 1 by $\operatorname{simp}$
show $\bigwedge p q \cdot p \in \operatorname{dom}($ refresh-1 $($ cs-rep $(f, P))) \Longrightarrow q \in \operatorname{transition~A~a~} p \Longrightarrow$
fst $($ the $(\operatorname{cs-rep}(g, Q) q)) \leq f s t($ the $($ refresh-1 $($ cs-rep $(f, P)) p))$
using $1(1,2)$ by (auto) (metis UN-I cs-rep-apply domI option.sel)
show $\wedge p . p \in \operatorname{dom}($ cs-rep $(g, Q)) \Longrightarrow$ accepting $A p \Longrightarrow$ even (fst (the (cs-rep $(g, Q) p))$ )
using $1(1,3)$ by auto
show cs-st (cs-rep $(g, Q))=\{q \in \bigcup$ (transition Aa'cs-st (refresh-1 (cs-rep $(f, P)))$ ).
even $(f s t$ the (cs-rep $(g, Q) q)))\}$
proof (cases $P=\{ \}$ )

```
    case True
    have cs-st (cs-rep (g,Q)) =Q using 3 by auto
    also have \ldots. ={q\in dom g. even (the (gq))} unfolding 2 using True by
auto
            also have ... = {q\in dom g. even (fst (the (cs-rep (g,Q) q)))} using
cs-rep-apply by metis
    also have dom g=\bigcup((transition A a)'(domf)) using 1(1) by this
    also have dom f=cs-st (refresh-1 (cs-rep (f,P))) using True by simp
    finally show ?thesis by this
    next
    case False
    have 4:fst (the (cs-rep (g,Q)q))= the (gq) if q\inU((transition A a)'P)
for q
            using 1(1) that assms(1) by (fast intro: cs-rep-apply)
    have cs-st (cs-rep (g,Q)) =Q using 3 by auto
    also have ... = {q\in\bigcup((transition A a)'P). even (the (gq))} unfolding
2 using False by auto
    also have ... ={q\in\bigcup((transition A a)'P). even (fst (the (cs-rep (g,Q)
q)))} using 4 by force
    also have P=(cs-st (refresh-1 (cs-rep (f,P)))) using assms(1) False by
auto
    finally show ?thesis by simp
    qed
qed
lemma complement-succ-1-refine: (complement-succ-1, complement-succ) \in
    Id }->\mathrm{ Id }->\mathrm{ cs-rel }->\langlecs-rel\rangle set-rel
proof (clarsimp simp: br-set-rel-alt in-br-conv)
    fix }A::('a,'b) nb
    fix af
    show complement-succ A a (cs-abs f) = cs-abs'complement-succ-1 A a f
    proof safe
            fix g Q
            assume 1:(g,Q)\in complement-succ A a (cs-absf)
            have 2: Q\subseteqdom g
            using 1 unfolding complement-succ-def lr-succ-def st-succ-def
            by (auto) (metis IntE cs-abs-rep cs-lr-dom cs-rep-st)
        have 3:cs-st f\subseteqdom (cs-lr f) unfolding cs-st-def by auto
        show (g,Q)\incs-abs' complement-succ-1 A a f
        proof
            show (g,Q) = cs-abs (cs-rep (g,Q)) using 2 by auto
            have cs-rep (g,Q)\incomplement-succ-1 A a (cs-rep (cs-abs f))
                    using complement-succ-1-rep 3 1 by this
            also have cs-rep (cs-abs f)=f by simp
            finally show cs-rep (g,Q)\in complement-succ-1 A af by this
            qed
        next
            fix g
            assume 1:g\in complement-succ-1 A af
```

show cs-abs $g \in$ complement-succ $A a(c s-a b s f)$ using complement-succ-1-abs 1 by this
qed
qed
lemma complement-1-refine: (complement-1, complement) $\in\langle I d$, Id $\rangle$ nba-rel $\rightarrow$ $\langle I d$, cs-rel〉 nba-rel
unfolding complement-1-def complement-def proof parametricity
fix $A B::\left({ }^{\prime} a,{ }^{\prime} b\right) n b a$
assume 1: $(A, B) \in\langle I d, I d\rangle$ nba-rel
have 2: (const (Some (2 * card (nodes B), False)) |' initial B, const (Some (2 * card (nodes B)))|' initial B, \{\}) $\in$ cs-rel unfolding $c s$-lr-def $c s$-st-def in-br-conv by (force simp: restrict-map-def)
show (complement-succ-1 $A$, complement-succ $B) \in I d \rightarrow$ cs-rel $\rightarrow\langle$ cs-rel $\rangle$ set-rel
using complement-succ-1-refine 1 by parametricity auto
show (\{const (Some (2 * card (nodes A), False)) |' initial A\},
$\left\{\right.$ const $\left.($ Some $(2 *$ card $($ nodes $B)))\right|^{\prime}$ initial $\left.\left.B\right\} \times\{\{ \}\}\right) \in\langle$ cs-rel $\rangle$ set-rel using 12 by simp parametricity
show $(\lambda f$. cs-st $f=\{ \}, \lambda(f, P) . P=\{ \}) \in$ cs-rel $\rightarrow$ bool-rel by (auto simp: in-br-conv)
qed

### 5.2 Phase 2

definition ranks-2 :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where
ranks-2 A a $f \equiv\{g$.
$\operatorname{dom} g=\bigcup(($ transition $A a) '(\operatorname{dom} f)) \wedge$
$(\forall$ q ld. $g q=$ Some $(l, d) \longrightarrow$
$l \leq \Pi(f s t$ 'Some -'f'pred A a q) $\wedge$
$(d \longleftrightarrow \bigsqcup($ snd 'Some -'f'pred A a q) $\wedge$ even $l) \wedge$
(accepting $A q \longrightarrow$ even $l)$ ) \}
definition complement-succ-2 ::
('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where
complement-succ-2 A $a \equiv$ ranks-2 A $a \circ$ refresh-1
definition complement-2 :: ('label, 'state) nba $\Rightarrow$ ('label, 'state items) nba where
complement-2 $A \equiv n b a$
(alphabet A)
$\left(\left.\{\right.$ const (Some $(2 \times$ card (nodes A), False $))\right|^{‘}$ initial $\left.\left.A\right\}\right)$
(complement-succ-2 A)
( $\lambda f$. True $\notin$ snd' $\operatorname{ran} f$ )
lemma ranks-2-refine: ranks-2 $=$ ranks-1 proof (intro ext)
fix $A::\left(' a,{ }^{\prime} b\right) n b a$ and $a f$
show ranks-2 A af ranks-1 A af
proof safe
fix $g$
assume 1: $g \in$ ranks-2 A a $f$
have 2: $\operatorname{dom} g=\bigcup(($ transition $A a)$ '( $\operatorname{dom} f))$ using 1 unfolding ranks-2-def by auto
have 3: $g q=\operatorname{Some}(l, d) \Longrightarrow l \leq \Pi(f s t$ 'Some -' $f$ 'pred $A$ a $q)$ for $q l d$ using 1 unfolding ranks-2-def by auto
have 4:g $q=\operatorname{Some}(l, d) \Longrightarrow d \longleftrightarrow \bigsqcup($ snd'Some -'f'pred A a q) $\wedge$ even $l$ for $q l d$
using 1 unfolding ranks-2-def by auto
have 5: $g q=\operatorname{Some}(l, d) \Longrightarrow$ accepting $A q \Longrightarrow$ even $l$ for $q l d$ using 1 unfolding ranks-2-def by auto
show $g \in$ ranks-1 A af
unfolding ranks-1-def
proof (intro CollectI conjI balli impI)
show $\operatorname{dom} g=\bigcup(($ transition $A$ a)' $(\operatorname{dom} f))$ using 2 by this
next
fix $p q$
assume 10: $p \in \operatorname{dom} f q \in \operatorname{transition~} A$ a $p$
obtain $k c$ where 11: f $p=\operatorname{Some}(k, c)$ using $10(1)$ by auto
have 12: $q \in \operatorname{dom} g$ using 102 by auto
obtain $l d$ where 13: $g q=\operatorname{Some}(l, d)$ using 12 by auto
have fst $($ the $(g q))=l$ unfolding 13 by simp
also have $\ldots \leq \Pi$ ( $f s t$ 'Some -' $f$ ' pred $A$ a q) using 313 by this
also have $\ldots \leq k$
proof (rule cInf-lower)
show $k \in f$ st'Some -' $f$ 'pred $A$ a $q$ using 11 10(2) by force
show bdd-below (fst'Some -' $f$ ' pred $A$ a q) by simp
qed
also have $\ldots=$ fst (the $(f p)$ ) unfolding 11 by simp
finally show $f$ st $($ the $(g q)) \leq f s t(t h e(f p))$ by this
next
fix $q$
assume 10: $q \in \operatorname{dom} g$ accepting $A q$
show even (fst (the $(g q))$ ) using 105 by auto
next
show cs-st $g=\{q \in \bigcup(($ transition $A a)$ '(cs-st f)). even $(f s t($ the $(g q)))\}$ proof
show cs-st $g \subseteq\{q \in \bigcup(($ transition $A a)$ '(cs-st $f))$. even $(f s t($ the $(g q)))\}$ using 4 unfolding cs-st-def image-def vimage-def by auto metis+ show $\{q \in \bigcup(($ transition $A a)$ ' $(c s$-st $f))$. even $(f s t($ the $(g q)))\} \subseteq$ cs-st $g$ proof safe
fix $p q$
assume 10: even $(f s t($ the $(g q))) p \in \operatorname{cs-st} f q \in \operatorname{transition~A~a~} p$
have 12: $q \in \operatorname{dom} g$ using 102 unfolding $c s$-st-def by auto
show $q \in c s-s t g$ using 10412 unfolding $c s$-st-def image-def by force qed
qed
qed
next
fix $g$
assume 1: g e ranks-1 A af
have 2: $\operatorname{dom} g=\bigcup(($ transition $A a)$ '( $\operatorname{dom} f))$ using 1 unfolding ranks-1-def by auto
have 3: $\wedge p q \cdot p \in \operatorname{dom} f \Longrightarrow q \in \operatorname{transition~A~a~} p \Longrightarrow f s t($ the $(g q)) \leq f s t$ (the ( $f$ p $)$ )
using 1 unfolding ranks-1-def by auto
have $4: \bigwedge q . q \in \operatorname{dom} g \Longrightarrow$ accepting $A q \Longrightarrow$ even $(f s t($ the $(g q)))$
using 1 unfolding ranks-1-def by auto
have 5: cs-st $g=\{q \in \bigcup(($ transition $A a)$ '(cs-st $f))$. even $(f s t($ the $(g q)))\}$ using 1 unfolding ranks-1-def by auto
show $g \in$ ranks-2 A af
unfolding ranks-2-def
proof (intro CollectI conjI allI impI)
show $\operatorname{dom} g=\bigcup(($ transition $A a)$ ' $(\operatorname{dom} f))$ using 2 by this
next
fix $q l d$
assume 10: g $q=\operatorname{Some}(l, d)$
have 11: $q \in$ dom $g$ using 10 by auto
show $l \leq \Pi(f s t ' S o m e-' f ' p r e d A$ a $q)$
proof (rule cInf-greatest)
show fst 'Some -' $f$ ' pred $A$ a $q \neq\{ \}$ using 11 unfolding 2 image-def vimage-def by force
show $\bigwedge x . x \in f s t$ 'Some -' $f$ 'pred $A$ a $q \Longrightarrow l \leq x$
using 310 by (auto) (metis domI fst-conv option.sel)
qed
have $d \longleftrightarrow q \in c s$-st $g$ unfolding cs-st-def by (force simp: 10)
also have cs-st $g=\{q \in \bigcup(($ transition $A a)$ '(cs-st f)). even (fst (the ( $g$
$q))$ ) $\}$ using 5 by this
also have $q \in \ldots \longleftrightarrow(\exists x \in$ cs-st $f . q \in$ transition $A$ a $x) \wedge$ even $l$
unfolding mem-Collect-eq 10 by simp
also have $\ldots \longleftrightarrow \bigsqcup($ snd'Some - ' $f$ ' pred $A$ a q) $\wedge$ even $l$
unfolding cs-st-def image-def vimage-def by auto metis+
finally show $d \longleftrightarrow \bigsqcup($ snd 'Some -'f'pred A a q) $\wedge$ even $l$ by this show accepting $A q \Longrightarrow$ even $l$ using 41011 by force
qed
qed
qed
lemma complement-2-refine: (complement-2, complement-1) $\in\langle I d, I d\rangle$ nba-rel $\rightarrow\langle I d, I d\rangle$ nba-rel
unfolding complement-2-def complement-1-def complement-succ-2-def comple-ment-succ-1-def
unfolding ranks-2-refine cs-st-def image-def vimage-def ran-def by auto

### 5.3 Phase 3

definition bounds-3 :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items where
bounds-3 A af $\equiv \lambda$ q. let $S=$ Some -'f' pred A a q in
if $S=\{ \}$ then None else Some $(\Pi(f s t$ ' $S), \bigsqcup($ snd ' $S))$
definition items-3 :: ('label, 'state) nba $\Rightarrow$ 'state $\Rightarrow$ item $\Rightarrow$ item set where items-3 A $p \equiv \lambda(k, c) .\{(l, c \wedge$ even $l) \mid l . l \leq k \wedge($ accepting $A p \longrightarrow$ even $l)\}$ definition get-3 :: ('label, 'state) nba $\Rightarrow$ 'state items $\Rightarrow$ ('state - item set) where
get-3 A $f \equiv \lambda$ p. map-option (items-3 A p) (f p)
definition complement-succ-3 ::
('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where
complement-succ-3 A $a \equiv$ expand-map $\circ$ get-3 $A \circ$ bounds-3 A $a \circ$ refresh-1
definition complement-3 :: ('label, 'state) $n b a \Rightarrow$ ('label, 'state items) nba where complement-3 $A \equiv n b a$
(alphabet A)
$\left(\left\{\left.(\right.\right.$ Some $\circ($ const $(2 *$ card $($ nodes $A)$, False $)))\right|^{‘}$ initial $\left.\left.A\right\}\right)$
(complement-succ-3 A)
$(\lambda f . \forall(p, k, c) \in$ map-to-set $f . \neg c)$
lemma bounds-3-dom[simp]: dom (bounds-3 A af) $=\bigcup(($ transition $A$ a)' (dom f))
unfolding bounds-3-def Let-def dom-def by (force split: if-splits)
lemma items-3-nonempty[intro!, simp]: items-3 A ps $\neq\{ \}$ unfolding items-3-def by auto
lemma items-3-finite[intro!, simp]: finite (items-3 A p s) unfolding items-3-def by (auto split: prod.splits)
lemma get-3-dom[simp]: dom (get-3 Af) $=\operatorname{dom} f$ unfolding get-3-def by (auto split: bind-splits)
lemma get-3-finite[intro, simp]: $S \in$ ran (get-3 $A f) \Longrightarrow$ finite $S$
unfolding get-3-def ran-def by auto
lemma get-3-update[simp]: get-3 $A(f(p \mapsto s))=($ get-3 $A f)(p \mapsto$ items-3 A p s)
unfolding get-3-def by auto
lemma expand-map-get-bounds-3: expand-map $\circ$ get-3 $A \circ$ bounds-3 $A a=$ ranks-2 A a
proof (intro ext set-eqI, unfold comp-apply)
fix $f g$
have 1: $(\forall x S y$. get-3 $A$ (bounds-3 A af) $x=$ Some $S \longrightarrow g x=$ Some $y \longrightarrow$ $y \in S) \longleftrightarrow$
$(\forall q S l d$. get-3 $A$ (bounds-3 A af) $q=$ Some $S \longrightarrow g q=\operatorname{Some}(l, d) \longrightarrow$ $(l, d) \in S)$
by auto
have 2: $(\forall$ S. get-3 $A$ (bounds-3 A a f) $q=$ Some $S \longrightarrow g q=\operatorname{Some}(l, d)$ $\longrightarrow(l, d) \in S) \longleftrightarrow$
$(g q=\operatorname{Some}(l, d) \longrightarrow l \leq \Pi(f s t ‘($ Some $-‘ f$ 'pred $A$ a $q)) \wedge$
$(d \longleftrightarrow \bigsqcup($ snd ' $($ Some $-‘ f$ 'pred $A$ a $q)) \wedge$ even $l) \wedge($ accepting $A q \longrightarrow$ even $l$ ))
if 3: $\operatorname{dom} g=\bigcup((\operatorname{transition} A a)$ ' $(\operatorname{dom} f))$ for $q l d$
proof -
have 4: $q \notin \operatorname{dom} g$ if Some $-{ }^{\prime} f$ 'pred $A$ a $q=\{ \}$ unfolding 3 using that by force
show ?thesis unfolding get-3-def items-3-def bounds-3-def Let-def using 4 by auto
qed
show $g \in$ expand-map (get-3 $A($ bounds-3 $A$ af $)) \longleftrightarrow g \in$ ranks-2 $A$ a $f$
unfolding expand-map-alt-def ranks-2-def mem-Collect-eq unfolding get-3-dom bounds-3-dom 1 using 2 by blast
qed
lemma complement-succ-3-refine: complement-succ-3 = complement-succ-2
unfolding complement-succ-3-def complement-succ-2-def expand-map-get-bounds-3 by rule
lemma complement-initial-3-refine: \{const (Some (2 * card (nodes A), False)) $\mid$ ' initial $A\}=$
$\left\{\left.(\right.$ Some $\circ($ const $(2 *$ card $($ nodes $A)$, False $)))\right|^{\prime}$ initial $\left.A\right\}$
unfolding comp-apply by rule
lemma complement-accepting-3-refine: True $\notin$ snd ' $\operatorname{ran} f \longleftrightarrow(\forall(p, k, c) \in$ map-to-set $f . \neg c$ )
unfolding map-to-set-def ran-def by auto
lemma complement-3-refine: (complement-3, complement-2) $\in\langle I d$, Id $\rangle$ nba-rel $\rightarrow\langle I d, I d\rangle$ nba-rel
unfolding complement-3-def complement-2-def
unfolding complement-succ-3-refine complement-initial-3-refine complement-accepting-3-refine by auto

### 5.4 Phase 4

definition items-4 :: ('label, 'state) nba $\Rightarrow$ 'state $\Rightarrow$ item $\Rightarrow$ item set where items-4 $A p \equiv \lambda(k, c) .\{(l, c \wedge$ even $l) \mid l . k \leq S u c l \wedge l \leq k \wedge($ accepting $A p$ $\longrightarrow$ even l)\}
definition get-4 :: ('label, 'state) nba $\Rightarrow$ 'state items $\Rightarrow$ ('state $\rightharpoonup$ item set)

## where

get-4 $A f \equiv \lambda p$ map-option (items-4 A p) (f p)
definition complement-succ-4 ::
('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items set where
complement-succ-4 $A a \equiv$ expand-map $\circ$ get-4 $A \circ$ bounds-3 $A a \circ$ refresh-1
definition complement-4 :: ('label, 'state) $n b a \Rightarrow$ ('label, 'state items) nba where
complement-4 $A \equiv n b a$
(alphabet A)
$\left(\left\{\left.(\right.\right.$ Some $\circ($ const $(2 *$ card $($ nodes $A)$, False $)))\right|^{‘}$ initial $\left.\left.A\right\}\right)$
(complement-succ-4 A)
$(\lambda f . \forall(p, k, c) \in$ map-to-set $f . \neg c)$
lemma get-4-dom[simp]: $\operatorname{dom}($ get-4 $A f)=\operatorname{dom} f$ unfolding get-4-def by (auto split: bind-splits)
definition $R$ :: 'state items rel where

```
    R\equiv{(f,g).
    domf=\operatorname{dom}g\wedge
    (\forall p\indom f. fst (the (f p)) \leq fst (the (g p))) ^
```



```
lemma bounds-R:
    assumes (f,g)\inR
    assumes bounds-3 A a (refresh-1 f) p=Some (n,e)
    assumes bounds-3 A a (refresh-1 g) p=Some (k, c)
    shows }n\leqke\longleftrightarrow
proof -
    have 1:
    domf=\operatorname{dom}g
    \forallp\in\operatorname{dom}f.fst (the (f p)) \leqfst (the (g p))
    \forallp\in\operatorname{dom}f.snd (the (f p))\longleftrightarrow snd (the (g p))
    using assms(1) unfolding }R\mathrm{ -def by auto
    have }n=\Pi(fst'(Some -' refresh-1 f'pred A a p)
    using assms(2) unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
    also have fst'Some -'refresh-1 f'pred A a p=fst'Some -'f'pred A a p
    proof
    show fst'Some -'refresh-1 f'pred A a p\subseteqfst'Some -'f'pred A a p
        unfolding refresh-1-def image-def
        by (auto simp: map-option-case split: option.split) (force)
    show fst'Some -'f'pred A a p\subseteqfst'Some -'refresh-1 f'pred A a p
        unfolding refresh-1-def image-def
    by (auto simp: map-option-case split: option.split) (metis fst-conv option.sel)
    qed
    also have ...= fst'Some -'f'(pred A a p\cap\operatorname{dom f)}
    unfolding dom-def image-def Int-def by auto metis
    also have ... = fst'the 'f'(pred A a p\cap domf)
    unfolding dom-def by force
    also have ...= (fst \circ the \circf)'(pred A a p\cap dom f) by force
    also have }\Pi((fst\circ\mathrm{ the ○f)'(pred A a p П dom f))}
        \Pi((fst \circ the \circ g)'(pred A a p \cap domg))
    proof (rule cINF-mono)
    show pred A a p\capdom g\not={}
        using assms(2) 1(1) unfolding bounds-3-def refresh-1-def
        by (auto simp: Let-def split: if-splits) (force+)
    show bdd-below ((fst \circ the \circf)'(pred A a p \cap dom f)) by rule
    show }\existsn\in\mathrm{ pred A a p ค dom f. (fst ○ the ○f) n < (fst ○ the ○g)m
        if m\in pred A a p\capdom g for m using 1 that by auto
    qed
    also have (fst \circ the ○ g)'(pred A a p \cap dom g) =fst' the 'g'(pred A a p \cap
dom g) by force
    also have ... =fst'Some -' g'(pred A a p \cap dom g)
        unfolding dom-def by force
    also have ... = fst'Some -' g'pred A a p
        unfolding dom-def image-def Int-def by auto metis
    also have ... = fst'Some -'refresh-1 g'pred A a p
```


## proof

show $f_{s t}$ 'Some -' $g$ ' pred $A$ a $p \subseteq f_{s t}$ 'Some -'refresh-1 g 'pred $A$ ap unfolding refresh-1-def image-def
by (auto simp: map-option-case split: option.split) (metis fst-conv option.sel)
show $f$ st 'Some -' refresh- 1 g ' pred $A$ a $p \subseteq f_{s t}$ 'Some -' $g$ 'pred A a $p$ unfolding refresh-1-def image-def by (auto simp: map-option-case split: option.split) (force)
qed
also have $\Pi(f s t$ ' $($ Some -' refresh-1 $g$ ' pred $A$ a $p))=k$
using $\operatorname{assms}(3)$ unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
finally show $n \leq k$ by this
have $e \longleftrightarrow \bigsqcup($ snd ' (Some -' refresh-1 f'pred A a p))
using assms(2) unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
also have snd 'Some -' refresh-1 $f$ 'pred A a $p=$ snd 'Some -'refresh-1 $f$ '
(pred A a $p \cap \operatorname{dom}$ (refresh-1 f))
unfolding dom-def image-def Int-def by auto metis
also have $\ldots=$ snd 'the 'refresh- 1 f ' (pred A a $p \cap \operatorname{dom}$ (refresh- 1 f$)$ )
unfolding dom-def by force
also have $\ldots=($ snd $\circ$ the $\circ$ refresh-1 $f)$ ' $($ pred $A$ a $p \cap \operatorname{dom}($ refresh-1 $f))$
by force
also have $\ldots=($ snd $\circ$ the $\circ$ refresh $-1 \mathrm{~g})$ ' $($ pred $A$ a $p \cap \operatorname{dom}($ refresh $-1 \mathrm{~g})$ )
proof (rule image-cong)
show pred $A$ a $p \cap \operatorname{dom}($ refresh-1 $f)=$ pred $A$ a $p \cap \operatorname{dom}($ refresh $-1 g)$
unfolding refresh-1-dom 1(1) by rule
show (snd $\circ$ the $\circ$ refresh-1 f) $q \longleftrightarrow($ snd $\circ$ the $\circ$ refresh-1 $g) q$
if 2: $q \in$ pred $A$ a $p \cap \operatorname{dom}$ (refresh-1 $g$ ) for $q$
proof
have $3: \forall x \in \operatorname{ran} f . \neg$ snd $x \Longrightarrow(n$, True $) \in \operatorname{ran} g \Longrightarrow g q=$ Some $(k$,
$c) \Longrightarrow c$ for $n k c$
using $1(1,3)$ unfolding dom-def ran-def
by (auto dest!: Collect-inj) (metis option.sel snd-conv)
have 4: $g q=$ Some ( $n$, True) $\Longrightarrow f q=$ Some ( $k, c$ ) $\Longrightarrow c$ for $n k c$
using 1 (3) unfolding dom-def by force
have 5: $\forall x \in$ ran $g . \neg$ snd $x \Longrightarrow(k$, True $) \in \operatorname{ran} f \Longrightarrow$ False for $k$
using $1(1,3)$ unfolding dom-def ran-def
by (auto dest!: Collect-inj) (metis option.sel snd-conv)
show (snd $\circ$ the $\circ$ refresh-1 $f) q \Longrightarrow($ snd $\circ$ the $\circ$ refresh $-1 g) q$
using $1(1,3) 23$ unfolding refresh- 1 -def by (force split: if-splits)
show (snd $\circ$ the $\circ$ refresh-1 $g) q \Longrightarrow($ snd $\circ$ the $\circ$ refresh-1 f) $q$
using $1(1,3) 245$ unfolding refresh-1-def
by (auto simp: map-option-case split: option.splits if-splits) (force+)
qed
qed
also have $\ldots=$ snd 'the 'refresh-1 g ' (pred A a $p \cap \operatorname{dom}($ refresh $-1 \mathrm{~g})$ ) by force
also have $\ldots=$ snd'Some -'refresh-1 $g$ ' (pred $A$ a $p \cap \operatorname{dom}($ refresh-1 $g)$ )
unfolding dom-def by force
also have $\ldots=$ snd'Some -' refresh-1 g 'pred $A$ a $p$
unfolding dom-def image-def Int-def by auto metis

```
        also have }\bigsqcup(snd'(Some -'refresh-1 g'pred A a p))\longleftrightarrow
        using assms(3) unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
    finally show }e\longleftrightarrowc\mathrm{ by this
qed
```

lemma complement-4-language-1: language (complement-3 A) $\subseteq$ language (complement-4
A)
proof (rule simulation-language)
show alphabet (complement-3 A) $\subseteq$ alphabet (complement-4 A)
unfolding complement-3-def complement-4-def by simp
show $\exists q \in$ initial (complement-4 $A$ ). $(p, q) \in R$ if $p \in$ initial (complement-3
A) for $p$
using that unfolding complement-3-def complement-4-def R-def by simp
show $\exists g^{\prime} \in$ transition (complement-4 A) a $g .\left(f^{\prime}, g^{\prime}\right) \in R$
if $f^{\prime} \in$ transition (complement-3 $A$ ) a $f(f, g) \in R$
for $a f f^{\prime} g$
proof -
have $1: f^{\prime} \in$ expand-map (get-3 A (bounds-3 A a (refresh-1 $f$ )) )
using that(1) unfolding complement-3-def complement-succ-3-def by auto
have 2:
$\operatorname{dom} f=\operatorname{dom} g$
$\forall p \in \operatorname{dom} f$.fst $($ the $(f p)) \leq$ fst $($ the $(g p))$
$\forall p \in \operatorname{dom} f$. snd (the $(f p)) \longleftrightarrow$ snd (the $(g p)$ )
using that(2) unfolding $R$-def by auto
have dom $f^{\prime}=\operatorname{dom}$ (get-3 A (bounds-3 A a (refresh-1 f))) using ex-
pand-map-dom 1 by this
also have $\ldots=\operatorname{dom}$ (bounds-3 A a (refresh-1 f)) by simp
finally have $3: \operatorname{dom} f^{\prime}=\operatorname{dom}(b o u n d s-3 A$ a (refresh-1 $f$ )) by this
define $g^{\prime}$ where $g^{\prime} p \equiv d o$
\{
$(k, c) \leftarrow$ bounds-3 A a (refresh-1 g) $p ;$
$(l, d) \leftarrow f^{\prime} p ;$
Some (if even $k=$ even $l$ then $k$ else $k-1, d$ )
$\}$ for $p$
have $4: g^{\prime} p=d o$
\{
$k c \leftarrow$ bounds-3 A a (refresh-1 g) p;
$l d \leftarrow f^{\prime} p ;$
Some (if even $(f s t k c)=$ even $(f s t l d)$ then fst kc else fst $k c-1$, snd ld)
$\}$ for $p$ unfolding $g^{\prime}$-def case-prod-beta by rule
have dom $g^{\prime}=\operatorname{dom}($ bounds-3 A a $($ refresh-1 $g)) \cap \operatorname{dom} f^{\prime}$ using 4 bind-eq-Some-conv
by fastforce
also have $\ldots=\operatorname{dom} f^{\prime}$ using 23 by $\operatorname{simp}$
finally have 5: $\operatorname{dom} g^{\prime}=\operatorname{dom} f^{\prime}$ by this
have $6:(l, d) \in$ items-3 A $p(k, c)$
if bounds-3 A a (refresh-1 f) $p=\operatorname{Some}(k, c) f^{\prime} p=\operatorname{Some}(l, d)$ for $p k c l$
$d$
using 1 that unfolding expand-map-alt-def get-3-def by blast
show ?thesis
unfolding complement-4-def nba.sel complement-succ-4-def comp-apply proof
show $\left(f^{\prime}, g^{\prime}\right) \in R$
unfolding $R$-def mem-Collect-eq prod.case
proof (intro conjI ballI)
show $\operatorname{dom} f^{\prime}=\operatorname{dom} g^{\prime}$ using 5 by rule
next
fix $p$
assume 10: $p \in \operatorname{dom} f^{\prime}$
have 11: $p \in \operatorname{dom}$ (bounds-3 A a (refresh-1 g)) using 2(1) 310 by simp obtain $k c$ where 12: bounds-3 A a (refresh-1 g) $p=\operatorname{Some}(k, c)$ using
11 by fast
obtain $l d$ where 13: $f^{\prime} p=\operatorname{Some}(l, d)$ using 10 by auto
obtain $n e$ where 14: bounds-3 A a (refresh-1f) $p=\operatorname{Some}(n, e)$ using
103 by fast
have 15: $(l, d) \in$ items-3 A $p(n, e)$ using 61413 by this
have 16: $n \leq k$ using bounds- $R(1)$ that(2) 1412 by this
have 17: $l \leq k$ using 1516 unfolding items-3-def by simp
have 18: even $k \longleftrightarrow$ odd $l \Longrightarrow l \leq k \Longrightarrow l \leq k-1$ by presburger
have 19: $e \longleftrightarrow c$ using bounds-R(2) that(2) 1412 by this
show fst $\left(\right.$ the $\left.\left(f^{\prime} p\right)\right) \leq f s t\left(\right.$ the $\left.\left(g^{\prime} p\right)\right)$ using 1718 unfolding 41213
by $\operatorname{simp}$
show snd $\left(\right.$ the $\left.\left(f^{\prime} p\right)\right) \longleftrightarrow$ snd $\left(\right.$ the $\left.\left(g^{\prime} p\right)\right)$ using 19 unfolding 41213 by $\operatorname{simp}$
qed
show $g^{\prime} \in$ expand-map $($ get-4 $A($ bounds-3 A a (refresh-1 $\left.g))\right)$
unfolding expand-map-alt-def mem-Collect-eq
proof (intro conjI allI impI)
show dom $g^{\prime}=\operatorname{dom}($ get-4 $A($ bounds-3 $A$ a (refresh-1 g) $))$ using 2(1) 3 5 by $\operatorname{simp}$
fix $p S x y$
assume 10: get-4 $A$ (bounds-3 A a (refresh-1 g)) $p=$ Some $S$
assume 11: $g^{\prime} p=$ Some $x y$
obtain $k c$ where 12: bounds-3 A a (refresh-1 g) $p=\operatorname{Some}(k, c) S=$ items-4 A p $(k, c)$
using 10 unfolding get-4-def by auto
obtain $l d$ where 13: $f^{\prime} p=\operatorname{Some}(l, d) x y=($ if even $k \longleftrightarrow$ even $l$ then $k$ else $k-1, d)$
using 1112 unfolding $g^{\prime}$-def by (auto split: bind-splits)
obtain $n e$ where 14: bounds-3 A a (refresh-1f) $p=$ Some ( $n, e$ ) using 13(1) 3 by fast
have 15: $(l, d) \in$ items-3 A $p(n, e)$ using $61413(1)$ by this
have 16: $n \leq k$ using bounds-R(1) that(2) 14 12(1) by this
have 17: $e \longleftrightarrow c$ using bounds-R(2) that(2) 14 12(1) by this
show $x y \in S$ using 151617 unfolding 12(2) 13(2) items-3-def items-4-def by auto
qed
qed
qed

```
    show }^pq.(p,q)\inR\Longrightarrow\mathrm{ accepting (complement-3 A) p ב accepting
(complement-4 A) q
    unfolding complement-3-def complement-4-def R-def map-to-set-def
    by (auto) (metis domIff eq-snd-iff option.exhaust-sel option.sel)
    qed
    lemma complement-4-less: complement-4 A \leq complement-3 A
    unfolding less-eq-nba-def
    unfolding complement-4-def complement-3-def nba.sel
    unfolding complement-succ-4-def complement-succ-3-def
    proof (safe intro!: le-funI, unfold comp-apply)
    fix afg
    assume g expand-map (get-4 A (bounds-3 A a (refresh-1 f)))
    then show g\in expand-map (get-3 A (bounds-3 A a (refresh-1 f)))
        unfolding get-4-def get-3-def items-4-def items-3-def expand-map-alt-def by
blast
    qed
    lemma complement-4-language-2: language (complement-4 A)\subseteq language (complement-3
A)
    using language-mono complement-4-less by (auto dest: monoD)
    lemma complement-4-language: language (complement-3 A) = language (complement-4
A)
    using complement-4-language-1 complement-4-language-2 by blast
    lemma complement-4-finite[simp]:
    assumes finite (nodes A)
    shows finite (nodes (complement-4 A))
proof -
    have (nodes (complement-3 A), nodes (complement-2 A)) \in \langleId\rangle set-rel
        using complement-3-refine by parametricity auto
    also have (nodes (complement-2 A), nodes (complement-1 A)) \in\langleId\rangle set-rel
        using complement-2-refine by parametricity auto
    also have (nodes (complement-1 A), nodes (complement A)) \in\langlecs-rel\rangle set-rel
        using complement-1-refine by parametricity auto
    finally have 1:(nodes (complement-3 A), nodes (complement A)) \in\langlecs-rel\rangle
set-rel by simp
    have 2: finite (nodes (complement A)) using complement-finite assms(1) by
this
    have 3: finite (nodes (complement-3 A))
        using finite-set-rel-transfer-back 1 cs-rel-inv-single-valued 2 by this
    have 4: nodes (complement-4 A)\subseteq nodes (complement-3 A)
        using nodes-mono complement-4-less by (auto dest: monoD)
    show finite (nodes (complement-4 A)) using finite-subset 4 3 by this
    qed
    lemma complement-4-correct:
    assumes finite (nodes A)
    shows language (complement-4 A) = streams (alphabet A) - language A
proof -
    have language (complement-4 A) = language (complement-3 A)
        using complement-4-language by rule
```

also have (language (complement-3 A), language (complement-2 A)) $\in\langle\langle I d\rangle$ stream-rel $\rangle$ set-rel
using complement-3-refine by parametricity auto
also have (language (complement-2 A), language (complement-1 A)) $\in\langle\langle I d\rangle$ stream-rel〉 set-rel
using complement-2-refine by parametricity auto
also have (language (complement-1 A), language (complement $A)) \in\langle\langle I d\rangle$ stream-rel〉 set-rel
using complement-1-refine by parametricity auto
also have language (complement $A$ ) streams (alphabet $A$ ) - language $A$
using complement-language assms(1) by this
finally show language (complement-4 A) streams (alphabet $A$ ) - language $A$ by simp
qed

### 5.5 Phase 5

definition refresh-5 :: 'state items $\Rightarrow$ 'state items nres where
refresh-5 $f \equiv$ if $\exists(p, k, c) \in$ map-to-set $f . c$
then RETURN $f$
else do
\{
ASSUME (finite $(\operatorname{dom} f)$ );
FOREACH (map-to-set $f)(\lambda(p, k, c)$ m. do
\{
ASSERT $(p \notin \operatorname{dom} m)$;
$\operatorname{RETURN}(m(p \mapsto(k$, True $)))$
\}
) Map.empty
\}
definition merge-5 :: item $\Rightarrow$ item option $\Rightarrow$ item where
merge- $5 \equiv \lambda(k, c) . \lambda$ None $\Rightarrow(k, c) \mid \operatorname{Some}(l, d) \Rightarrow(k \sqcap l, c \sqcup d)$
definition bounds-5 :: ('label, 'state) nba $\Rightarrow$ 'label $\Rightarrow$ 'state items $\Rightarrow$ 'state items nres where
bounds-5 A a $f \equiv$ do
\{
ASSUME (finite (dom f));
ASSUME $(\forall$ p. finite (transition A a p));
FOREACH (map-to-set $f)(\lambda(p, s) m$.
FOREACH (transition A a p) ( $\lambda q f$.
RETURN $(f(q \mapsto$ merge-5 $s(f q))))$
m)

Map.empty
\}
definition items-5 :: ('label,'state) nba $\Rightarrow$ 'state $\Rightarrow$ item $\Rightarrow$ item set where items-5 A $p \equiv \lambda(k, c)$.do \{
let values $=$ if accepting $A p$ then Set.filter even $\{k-1$.. $k\}$ else $\{k-1$.. $k\} ;$

```
        let item = \lambdal. (l, c ^ even l);
        item'values
    }
    definition get-5 :: ('label,''state) nba => 'state items }=>\mathrm{ ('state }-\mathrm{ item set)
where
        get-5 A f 三 \ p.map-option (items-5 A p) (f p)
    definition expand-5 :: ('a \rightharpoonup'b set) => ('a \rightharpoonup'b) set nres where
        expand-5 f 三FOREACH (map-to-set f) ( }\lambda(x,S)X.do 
            ASSERT ( }\forall\textrm{g}\inX.x\not\indom g)
            ASSERT (\foralla\inS.\forallb\inS.a\not=b\longrightarrow(\lambday. (\lambda g. g (x\mapstoy))'X)a\cap (\lambda
y. (\lambdag.g(x\mapstoy))'X) b={});
            RETURN (U y \inS. (\lambda g. g(x\mapstoy))`X)
        }) {Map.empty}
definition complement-succ-5 ::
    ('label, 'state) nba => 'label }=>\mathrm{ 'state items }=>\mathrm{ 'state items set nres where
    complement-succ-5 A a f 三do
    {
        f\leftarrowrefresh-5 f;
        f\leftarrowbounds-5 A a f;
        ASSUME (finite (dom f));
        expand-5 (get-5 A f)
    }
```

lemma bounds－3－empty：bounds－3 A a Map．empty $=$ Map．empty unfolding bounds－3－def Let－def by auto
lemma bounds－3－update：bounds－3 $A$ a $(f(p \mapsto s))=$
override－on（bounds－3 A a f）（Some $\circ$ merge－5 $s$ ○ bounds－3 A a $(f(p:=$
None）））（ （ransition A a p）

## proof

note fun－upd－image［simp］
fix $q$
show bounds－3 A a $(f(p \mapsto s)) q=$
override－on（bounds－3 A a f）（Some $\circ$ merge－5 $s \circ$ bounds－3 A a $(f(p:=$
None）））（transition A a p）q
proof（cases $q \in$ transition $A$ a $p$ ）
case True
define $S$ where $S \equiv$ Some $-{ }^{\prime} f$＇$($ pred $A$ a $q-\{p\})$
have 1：Some－＇f $(p:=$ Some $s)$＇pred $A$ a $q=$ insert s $S$ using True
unfolding $S$－def by auto
have 2：Some－＇$f(p:=$ None $)$＇pred $A$ a $q=S$ unfolding $S$－def by auto
have bounds－3 $A$ a $(f(p \mapsto s)) q=$ Some $(\Pi(f s t$＇$($ insert $s ~ S)), ~ \bigsqcup($ snd＇ （insert $s S$ ））
unfolding bounds－3－def 1 by simp
also have $\ldots=$ Some（merge－5 $s($ bounds－3 A a $(f(p:=$ None $)) q))$
unfolding 2 bounds－3－def merge－5－def by（cases s）（simp－all add：cInf－insert）
also have $\ldots=$ override－on（bounds－3 A a f）（Some ○ merge－5 $s \circ$ bounds－3
A $a(f(p:=$ None $)))$
（transition $A$ a $p$ ）$q$ using True by simp
finally show ？thesis by this

```
    next
            case False
            then have pred A a q\cap{x.x\not=p}= pred A a q
                by auto
            with False show ?thesis by (simp add: bounds-3-def)
        qed
    qed
```

    lemma refresh-5-refine: \((\) refresh-5, \(\lambda \mathrm{f}\). RETURN \((\) refresh-1 \(f)) \in I d \rightarrow\langle I d\rangle\)
    nres-rel
proof safe
fix $f::{ }^{\prime} a \Rightarrow$ item option
have $1:(\exists(p, k, c) \in$ map-to-set $f . c) \longleftrightarrow$ True $\in$ snd ' $\operatorname{ran} f$
unfolding image-def map-to-set-def ran-def by force
show (refresh-5 f, RETURN (refresh-1 $f)) \in\langle$ Id $\rangle$ nres-rel
unfolding refresh-5-def refresh-1-def 1
by (refine-vcg FOREACH-rule-map-eq[where $X=\lambda$ m. map-option (apsnd
T) $\circ m]$ ( auto)
qed
lemma bounds-5-refine: (bounds-5 A a, $\lambda \mathrm{f}$. RETURN (bounds-3 A af)) $\in I d$
$\rightarrow\langle I d\rangle$ nres-rel
unfolding bounds-5-def by
(refine-vcg FOREACH-rule-map-eq[where $X=$ bounds-3 A a] FOREACH-rule-insert-eq)
(auto simp: override-on-insert bounds-3-empty bounds-3-update)
lemma items-5-refine: items-5 = items-4
unfolding items-5-def items-4-def by (intro ext) (auto split: if-splits)
lemma get-5-refine: get-5 $=$ get-4
unfolding get-5-def get-4-def items-5-refine by rule
lemma expand-5-refine: (expand-5 $f, A S S E R T$ (finite $(\operatorname{dom} f)) \gg R E T U R N$
$($ expand-map $f)) \in\langle I d\rangle$ nres-rel
unfolding expand-5-def
by (refine-vcg FOREACH-rule-map-eq[where $X=$ expand-map]) (auto dest!:
expand-map-dom map-upd-eqD1)
lemma complement-succ-5-refine: (complement-succ-5, RETURN o๐० comple-ment-succ-4) $\in$

Id $\rightarrow$ Id $\rightarrow$ Id $\rightarrow\langle$ Id $\rangle$ nres-rel
unfolding complement-succ-5-def complement-succ-4-def get-5-refine comp-apply
by (refine-vcg vcg1 [OF refresh-5-refine $]$ vcg $1[O F$ bounds-5-refine $]$ vcg $0[O F$ ex-pand-5-refine]) (auto)

### 5.6 Phase 6

definition expand-map-get-6 :: ('label, 'state) nba $\Rightarrow$ 'state items $\Rightarrow$ 'state items set nres where
expand-map-get-6 A $f \equiv$ FOREACH (map-to-set $f)(\lambda(k, v) X$. do \{
ASSERT $(\forall g \in X . k \notin \operatorname{dom} g)$;
ASSERT $(\forall a \in($ items-5 A $k v) . \forall b \in($ items-5 A $k v) . a \neq b \longrightarrow(\lambda y .(\lambda$ $g . g(k \mapsto y))$ ' $X) a \cap(\lambda y .(\lambda g . g(k \mapsto y))$ ' $X) b=\{ \})$;

```
    RETURN(Uy\initems-5 A kv. (\lambda g.g(k\mapstoy))`X)
    }) {Map.empty}
    lemma expand-map-get-6-refine: (expand-map-get-6, expand-5 o० get-5) }\inId
Id }->\langleId\rangle nres-re
    unfolding expand-map-get-6-def expand-5-def get-5-def by (auto intro: FORE-
ACH-rule-map-map[param-fo])
definition complement-succ-6 ::
    ('label,'state) nba = 'label = 'state items }=>\mathrm{ 'state items set nres where
    complement-succ-6 A a f 三do
    {
        f\leftarrowrefresh-5 f;
        f\leftarrowbounds-5 A a f;
        ASSUME (finite (dom f));
        expand-map-get-6 A f
    }
```

lemma complement-succ-6-refine:
(complement-succ-6, complement-succ-5) $\in I d \rightarrow I d \rightarrow I d \rightarrow\langle$ Id $\rangle$ nres-rel unfolding complement-succ-6-def complement-succ-5-def
by (refine-vcg vcg2[OF expand-map-get-6-refine]) (auto intro: refine-IdI)

### 5.7 Phase 7

interpretation autoref-syn by this

## context

fixes $f f$
assumes $f[$ autoref-rules $]:(f i, f) \in$ state-rel
begin
private lemma [simp]: finite (dom f)
using list-map-rel-finite $f$ unfolding finite-map-rel-def by force
schematic-goal refresh-7: (?f :: ?'a, refresh-5 f) $\in$ ?R
unfolding refresh-5-def by (autoref-monadic (plain))
end
concrete-definition refresh-7 uses refresh-7
lemma refresh-7-refine: $(\lambda f$. RETURN (refresh-7f), refresh-5) $\in$ state-rel $\rightarrow$ $\langle$ state-rel $\rangle$ nres-rel
using refresh-7.refine by fast
context
fixes $A$ :: ('label, nat) nba
fixes succi a $f f$
assumes succi[autoref-rules]: $($ succi, transition $A a) \in$ nat-rel $\rightarrow\langle n a t-r e l\rangle$ list-set-rel
assumes $f[$ autoref-rules $]:(f i, f) \in$ state-rel
begin
private lemma [simp]: finite (transition A a p)
using list-set-rel-finite succi[param-fo] unfolding finite-set-rel-def by blast
private lemma [simp]: finite ( $\operatorname{dom} f$ ) using $f i$ by force
private lemma [autoref-op-pat]: transition $A a \equiv O P(\operatorname{transition~} A$ a) by simp
private lemma [autoref-rules]: $(\min , \min ) \in$ nat-rel $\rightarrow$ nat-rel $\rightarrow$ nat-rel by simp
schematic-goal bounds-7:
notes $t y$ - $R E L[$ where $R=\langle n a t-r e l$, item-rel $\rangle$ dftt-ahm-rel, autoref-tyrel]
shows (?f :: ?'a, bounds-5 A af) $\in$ ?R
unfolding bounds-5-def merge-5-def sup-bool-def inf-nat-def by (autoref-monadic (plain))
end
concrete-definition bounds-7 uses bounds-7
lemma bounds-7-refine: $($ si, transition $A$ a) $\in$ nat-rel $\rightarrow\langle$ nat-rel $\rangle$ list-set-rel $\Longrightarrow$ ( $\lambda$ p. RETURN (bounds-7 si p), bounds-5 A a) $\in$ state-rel $\rightarrow\langle\langle$ nat-rel, item-rel $\rangle$ dftt-ahm-rel $\rangle$ nres-rel
using bounds-7.refine by auto

## context

fixes $A::$ ('label, nat) nba
fixes acci
assumes [autoref-rules]: (acci, accepting A) $\in$ nat-rel $\rightarrow$ bool-rel

## begin

private lemma [autoref-op-pat]: accepting $A \equiv O P($ accepting $A)$ by simp
private lemma [autoref-rules]: $((d v d),(d v d)) \in$ nat-rel $\rightarrow$ nat-rel $\rightarrow$ bool-rel by $\operatorname{simp}$
private lemma [autoref-rules]: $(\lambda k l$. upt $k$ (Suc l), atLeastAtMost $) \in$ nat-rel $\rightarrow$ nat-rel $\rightarrow\langle$ nat-rel $\rangle$ list-set-rel
by (auto simp: list-set-rel-def in-br-conv)
schematic-goal items-7: (?f :: ?'a, items-5 A) $\in$ ? $R$
unfolding items-5-def Let-def Set.filter-def by autoref
end
concrete-definition items-7 uses items-7

```
context
    fixes A :: ('label, nat) nba
    fixes ai
    fixes fif
    assumes ai:(ai, accepting A) \in nat-rel }->\mathrm{ bool-rel
    assumes fi[autoref-rules]: (fi,f)\in\langlenat-rel, item-rel\rangle dflt-ahm-rel
begin
    private lemma [simp]: finite (dom f)
    using dflt-ahm-rel-finite-nat fi unfolding finite-map-rel-def by force
    private lemma [simp]:
    assumes \m. m S S\Longrightarrowx\not\in dom m
    shows inj-on (\lambda m.m(x\mapstoy))S
            using assms unfolding dom-def inj-on-def by (auto) (metis fun-upd-triv
fun-upd-upd)
    private lemmas [simp] =op-map-update-def[abs-def]
    private lemma [autoref-op-pat]: items-5 A \equivOP (items-5 A) by simp
    private lemmas [autoref-rules]= items-7.refine[OF ai]
    schematic-goal expand-map-get-7: (?f, expand-map-get-6 A f)\in
        <<state-rel\rangle list-set-rel\rangle nres-rel
        unfolding expand-map-get-6-def by (autoref-monadic (plain))
end
concrete-definition expand-map-get-7 uses expand-map-get-7
lemma expand-map-get-7-refine:
    assumes (ai, accepting A) \in nat-rel }->\mathrm{ bool-rel
    shows (\lambda fi. RETURN (expand-map-get-7 ai fi),
        \lambdaf. ASSUME (finite (dom f))> expand-map-get-6 A f) }
            \nat-rel, item-rel\rangle dflt-ahm-rel }->\langle\langle\mathrm{ state-rel> list-set-rel> nres-rel
    using expand-map-get-7.refine[OF assms] by auto
context
    fixes A :: ('label, nat) nba
    fixes a :: 'label
    fixes p :: nat items
    fixes }
    fixes ai
    fixes pi
    assumes Ai: (Ai,A)\in\langleId,Id\rangle nbai-nba-rel
    assumes ai: (ai,a) \inId
    assumes pi[autoref-rules]:(pi,p)\in state-rel
begin
```

private lemmas succi $=$ nbai-nba-param(4)[THEN fun-relD, OF Ai, THEN fun-relD, OF ai]
private lemmas acceptingi $=$ nbai-nba-param(5)[THEN fun-relD, OF Ai]
private lemma [autoref-op-pat]: ( $\lambda$ g. ASSUME (finite (dom g)) >ex-pand-map-get-6 A g) $\equiv$
$O P(\lambda$ g. ASSUME $($ finite $($ dom $g)) \gg$ expand-map-get- 6 A g) by simp
private lemma [autoref-op-pat]: bounds-5 A $a \equiv O P$ (bounds-5 A a) by simp
private lemmas [autoref-rules] $=$ refresh-7-refine
bounds-7-refine $[O F$ succi $]$
expand-map-get-7-refine[OF acceptingi]
schematic-goal complement-succ-7: (?f :: ?'a, complement-succ-6 A a p) $\in$ ?R
unfolding complement-succ-6-def by (autoref-monadic (plain))
end
concrete-definition complement-succ-7 uses complement-succ-7
lemma complement-succ-7-refine:
(RETURN о०० complement-succ-7, complement-succ-6) $\in$ $\langle I d$, Id $\rangle$ nbai-nba-rel $\rightarrow$ Id $\rightarrow$ state-rel $\rightarrow$ $\langle\langle$ state-rel $\rangle$ list-set-rel〉 nres-rel
using complement-succ-7.refine unfolding comp-apply by parametricity
context
fixes $A$ :: ('label, nat) nba
fixes $A i$
fixes $n n i$ :: nat
assumes $A i:(A i, A) \in\langle I d, I d\rangle n b a i-n b a-r e l$
assumes ni[autoref-rules]: $(n i, n) \in I d$
begin
private lemma [autoref-op-pat]: initial $A \equiv O P($ initial $A)$ by $\operatorname{simp}$
private lemmas $[$ autoref-rules $]=$ nbai-nba-param(3)[THEN fun-relD, OF Ai]
schematic-goal complement-initial-7:
$\left(? f,\left\{\left.(\right.\right.$ Some $\circ($ const $(2 * n$, False $)))\right|^{\prime}$ initial $\left.\left.A\right\}\right) \in\langle$ state-rel $\rangle$ list-set-rel by autoref
end
concrete-definition complement-initial-7 uses complement-initial-7
schematic-goal complement-accepting-7: $(? f, \lambda f . \forall(p, k, c) \in$ map-to-set $f . \neg$ c) $\in$
state-rel $\rightarrow$ bool-rel
by autoref
concrete-definition complement-accepting-7 uses complement-accepting-7
definition complement-7 :: ('label, nat) nbai $\Rightarrow$ nat $\Rightarrow$ ('label, state) nbai where complement-7 Ai ni $\equiv$ nbai
(alphabeti Ai)
(complement-initial-7 Ai ni)
(complement-succ-7 Ai)
(complement-accepting-7)
lemma complement-7-refine[autoref-rules]:
assumes $(A i, A) \in\langle I d, I d\rangle$ nbai-nba-rel
assumes ( $n i$,
(OP card ::: 〈Id〉 ahs-rel bhc $\rightarrow$ nat-rel) \$
$((O P$ nodes $:::\langle I d, I d\rangle$ nbai-nba-rel $\rightarrow\langle I d\rangle$ ahs-rel bhc $) \$ A)) \in$ nat-rel
shows (complement-7 Ai ni, (OP complement-4 :::
$\langle I d, I d\rangle$ nbai-nba-rel $\rightarrow\langle I d$, state-rel $\rangle$ nbai-nba-rel) $\$ A) \in\langle I d$, state-rel $\rangle$
nbai-nba-rel
proof -
note complement-succ-7-refine
also note complement-succ-6-refine
also note complement-succ-5-refine
finally have 1 : (complement-succ-7, complement-succ-4) $\in$
$\langle I d, I d\rangle$ nbai-nba-rel $\rightarrow I d \rightarrow$ state-rel $\rightarrow\langle$ state-rel $\rangle$ list-set-rel unfolding nres-rel-comp unfolding nres-rel-def unfolding fun-rel-def by
auto
show ?thesis
unfolding complement-7-def complement-4-def
using 1 complement-initial-7.refine complement-accepting-7.refine assms unfolding autoref-tag-defs by parametricity
qed
end

## 6 Boolean Formulae

theory Formula
imports Main
begin

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

```
    Negation 'a formula |
    Conjunction 'a formula 'a formula 
    Disjunction 'a formula 'a formula
primrec satisfies :: 'a set }=>\mathrm{ 'a formula }=>\mathrm{ bool where
    satisfies A False \longleftrightarrowHOL.False |
    satisfies A True \longleftrightarrowHOL.True |
    satisfies A (Variable a)\longleftrightarrow \longleftrightarrowa\inA|
    satisfies A (Negation x) \longleftrightarrow satisfies A x 
    satisfies A (Conjunction x y) \longleftrightarrow satisfies A x ^ satisfies A y 
    satisfies A (Disjunction x y) \longleftrightarrow satisfies A x \vee satisfies A y
```

end

## 7 Final Instantiation of Algorithms Related to Complementation

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

### 7.1 Syntax

no-syntax -do-let :: [pttrn, 'a] $\Rightarrow$ do-bind ((2let - =/ -) [1000, 13] 13)
syntax -do-let $::[$ pttrn, 'a] $\Rightarrow$ do-bind $((2 l e t-=/ ~-) ~ 13) ~$

### 7.2 Hashcodes on Complement States

definition hci $k \equiv$ uint32-of-nat $k * 1103515245+12345$
definition $h c \equiv \lambda(p, q, b)$. hci $p+h c i q * 31+($ if $b$ then 1 else 0$)$
definition list-hash $x s \equiv$ fold (xor $\circ h c$ ) xs 0
lemma list-hash-eq:
assumes distinct $x s$ distinct ys set $x s=$ set $y s$
shows list-hash xs $=$ list-hash ys
proof -
have mset (remdups xs) $=$ mset (remdups ys) using assms(3) using set-eq-iff-mset-remdups-eq by blast
then have mset $x s=m$ set ys using $\operatorname{assms}(1,2)$ by (simp add: distinct-remdups-id)
have fold (xor $\circ h c) x s=$ fold $(x o r \circ h c) y s$ apply (rule fold-multiset-equiv)
apply (simp-all add: fun-eq-iff ac-simps)
using $\langle m s e t$ xs $=m s e t$ ys .
then show ?thesis unfolding list-hash-def by simp
definition state-hash $::$ nat $\Rightarrow$ Complementation-Implement.state $\Rightarrow$ nat where state-hash n $p \equiv$ nat-of-hashcode (list-hash p) mod $n$
lemma state-hash-bounded-hashcode[autoref-ga-rules]: is-bounded-hashcode state-rel (gen-equals (Gen-Map.gen-ball (foldli $\circ$ list-map-to-list)) (list-map-lookup (=)) (prod-eq $(=)(\longleftrightarrow))$ ) state-hash

## proof

show [param]: (gen-equals (Gen-Map.gen-ball (foldli $\circ$ list-map-to-list)) (list-map-lookup (=))
$($ prod-eq $(=)(\longleftrightarrow)),(=)) \in$ state-rel $\rightarrow$ state-rel $\rightarrow$ bool-rel by autoref
show state-hash $n x s=$ state-hash $n$ ys if $x s \in$ Domain state-rel ys $\in$ Domain state-rel
gen-equals (Gen-Map.gen-ball (foldli ○ list-map-to-list))
(list-map-lookup $(=)$ ) (prod-eq $(=)(=))$ xs ys for xs ys $n$
proof -
have 1: distinct (map fst xs) distinct (map fst ys)
using that(1, 2) unfolding list-map-rel-def list-map-invar-def by (auto simp: in-br-conv)
have 2: distinct $x$ s distinct ys using 1 by (auto intro: distinct-mapI)
have 3: (xs, map-of xs) state-rel (ys, map-of ys) $\in$ state-rel
using 1 unfolding list-map-rel-def list-map-invar-def by (auto simp: in-br-conv)
have 4: (gen-equals (Gen-Map.gen-ball (foldli ○ list-map-to-list)) (list-map-lookup (=))
$($ prod-eq $(=)(\longleftrightarrow))$ xs ys, map-of $x s=$ map-of $y s) \in$ bool-rel using 3 by parametricity
have 5: map-to-set (map-of xs) = map-to-set (map-of ys) using that(3) 4 by $\operatorname{simp}$
have 6 : set $x s=$ set ys using map-to-set-map-of 15 by blast
show state-hash $n$ xs $=$ state-hash $n$ ys unfolding state-hash-def using list-hash-eq 26 by metis
qed
show state-hash $n x<n$ if $1<n$ for $n x$ using that unfolding state-hash-def by $\operatorname{simp}$
qed

### 7.3 Complementation

schematic-goal complement-impl:
assumes [simp]: finite (NBA.nodes A)
assumes [autoref-rules]: $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (?f :: ?'c, op-translate (complement-4 A)) $\in$ ? $R$
by (autoref-monadic (plain))
concrete-definition complement-impl uses complement-impl
theorem complement-impl-correct:
assumes finite (NBA.nodes A)
assumes $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows NBA.language (nbae-nba (nbaei-nbae (complement-impl Ai))) = streams (nba.alphabet A) - NBA.language $A$
using op-translate-language[OF complement-impl.refine[OF assms]]
using complement-4-correct[OF assms(1)]
by $\operatorname{simp}$

### 7.4 Language Subset

definition [simp]: op-language-subset $A B \equiv$ NBA.language $A \subseteq$ NBA.language B
lemmas [autoref-op-pat] $=$ op-language-subset-def[symmetric]
schematic-goal language-subset-impl:
assumes [simp]: finite (NBA.nodes B)
assumes [autoref-rules]: $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
assumes [autoref-rules]: $(B i, B) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (?f :: ?' c, do \{
let $A B^{\prime}=$ intersect $^{\prime} A$ (complement-4 $B$ );
ASSERT (finite ( $N G B A$.nodes $A B^{\prime}$ ));
RETURN (NGBA.language $A B^{\prime}=\{ \}$ )
\}) $\in ? R$
by (autoref-monadic (plain))
concrete-definition language-subset-impl uses language-subset-impl
lemma language-subset-impl-refine[autoref-rules]:
assumes SIDE-PRECOND (finite (NBA.nodes A))
assumes SIDE-PRECOND (finite (NBA.nodes B))
assumes SIDE-PRECOND (nba.alphabet $A \subseteq$ nba.alphabet $B$ )
assumes $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
assumes $(B i, B) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (language-subset-impl Ai Bi, (OP op-language-subset :::
$\langle$ Id, nat-rel $\rangle$ nbai-nba-rel $\rightarrow\langle I d$, nat-rel $\rangle$ nbai-nba-rel $\rightarrow$ bool-rel) $\$ A \$ B) \in$ bool-rel
proof -
have (RETURN (language-subset-impl Ai Bi), do \{
let $A B^{\prime}=$ intersect $^{\prime} A$ (complement-4 B);
ASSERT (finite ( $N G B A$.nodes $A B^{\prime}$ ));
RETURN (NGBA.language $\left.A B^{\prime}=\{ \}\right)$
\}) $\in\langle$ bool-rel $\rangle$ nres-rel
using language-subset-impl.refine assms(2, 4, 5) unfolding autoref-tag-defs
by this
also have (do \{
let $A B^{\prime}=$ intersect $^{\prime} A$ (complement-4 $B$ );
ASSERT (finite ( $N G B A$.nodes $A B^{\prime}$ ));
RETURN (NGBA.language $A B^{\prime}=\{ \}$ )
\}, RETURN $($ NBA.language $A \subseteq$ NBA.language $B)) \in\langle$ bool-rel $\rangle$ nres-rel
proof refine-vcg
show finite (NGBA.nodes (intersect' $A$ (complement-4 B))) using assms(1,

## 2) by auto

have 1: NBA.language $A \subseteq$ streams (nba.alphabet $B$ )
using nba.language-alphabet streams-mono2 assms(3) unfolding autoref-tag-defs by blast
have 2: NBA.language (complement-4 B) $=$ streams (nba.alphabet B) NBA.language $B$
using complement-4-correct assms(2) by auto
show (NGBA.language (intersect' $A($ complement-4 $B))=\{ \}$,
NBA.language $A \subseteq$ NBA.language $B) \in$ bool-rel using 12 by auto
qed
finally show ?thesis using RETURN-nres-relD unfolding nres-rel-comp by force
qed

### 7.5 Language Equality

definition [simp]: op-language-equal $A B \equiv$ NBA.language $A=N B A$.language B
lemmas [autoref-op-pat] $=$ op-language-equal-def $[$ symmetric $]$
schematic-goal language-equal-impl:
assumes [simp]: finite (NBA.nodes $A$ )
assumes [simp]: finite (NBA.nodes B)
assumes [simp]: nba.alphabet $A=$ nba.alphabet $B$
assumes [autoref-rules]: $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
assumes [autoref-rules]: $(B i, B) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (?f :: ?'c, NBA.language $A \subseteq N B A$.language $B \wedge N B A$.language $B \subseteq$
NBA.language $A) \in$ ? $R$
by autoref
concrete-definition language-equal-impl uses language-equal-impl
lemma language-equal-impl-refine[autoref-rules]:
assumes SIDE-PRECOND (finite (NBA.nodes A))
assumes SIDE-PRECOND (finite (NBA.nodes B))
assumes SIDE-PRECOND (nba.alphabet $A=$ nba.alphabet B)
assumes $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
assumes $(B i, B) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (language-equal-impl Ai Bi, (OP op-language-equal :::
$\langle I d$, nat-rel $\rangle$ nbai-nba-rel $\rightarrow\langle$ Id, nat-rel $\rangle$ nbai-nba-rel $\rightarrow$ bool-rel $) \$$ A B $) \in$
bool-rel
using language-equal-impl.refine[OF assms[unfolded autoref-tag-defs]] by auto
schematic-goal product-impl:
assumes [simp]: finite (NBA.nodes B)
assumes [autoref-rules]: $(A i, A) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
assumes [autoref-rules]: $(B i, B) \in\langle I d$, nat-rel $\rangle$ nbai-nba-rel
shows (?f :: ?' c, do \{
let $A B^{\prime}=$ intersect $A($ complement-4 $B)$;
ASSERT (finite (NBA.nodes $\left.A B^{\prime}\right)$ );
op－translate $A B^{\prime}$
\}) $\in ? R$
by（autoref－monadic（plain））
concrete－definition product－impl uses product－impl

## export－code

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

## 8 Build and test exported program with MLton

theory Complementation－Build
imports Complementation－Final
begin
external－file 〈code／Autool．mlb〉
external－file 〈code／Prelude．sml〉
external－file 〈code／Autool．sml〉
compile－generated－files
〈code／Complementation．ML〉（in Complementation－Final）
external－files
〈code／Autool．mlb〉
〈code／Prelude．sml〉
〈code／Autool．sml〉
export－files 〈code／Complementation．sml〉 and 〈code／Autool〉（exe）
where $\langle f n$ dir $=>$
let
val exec $=$ Generated－Files．execute（dir + Path．basic code $)$ ；
val $-=$ exec 〈Prepare〉mv Complementation．ML Complementation．sml；
val $-=$ exec $\langle$ Compilation〉（verbatim $\langle$ ISABELLE－MLTON $\$$ ISABELLE－MLTON－OPTIONS
，
－profile time－default－type intinf Autool．mlb）；
val $-=$ exec $\langle$ Test $\rangle$ ．／Autool help；
in（）end＞
end

## References

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

