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 if 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 Iteration 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

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

6 Boolean Formulae .................................................. 54

7 Final Instantiation of Algorithms Related to Complementation

7.1 Syntax .......................................................... 55
7.2 Hashcodes on Complementation States ....................... 55
7.3 Complementation .............................................. 56
7.4 Language Subset ............................................. 57
7.5 Language Equality ........................................... 58

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

1 Alternating Function Iteration

theory Alternate
imports Main
begin

primrec alternate :: ('a ⇒ 'a) ⇒ ('a ⇒ 'a) ⇒ nat ⇒ ('a ⇒ 'a) where
  alternate f g 0 = id | alternate f g (Suc k) = alternate g f k ∘ f

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

declare alternate.simps(2)[simp del]

lemma alternate-antimono:
assumes \( \forall x. f x \leq x \land \forall x. g x \leq x \)

shows antimono \((\text{alternate } f \ g)\)

proof

fix \( k \ l :: \text{nat} \)

assume \( 1: k \leq l \)

obtain \( n \) where \( 2: l = k + n \) using \text{le-Suc-ex} \( 1 \) by \text{auto}

have \( 3: \text{alternate } f \ g \ (k + n) \leq \text{alternate } f \ g \ k \)

proof (induct \( n \))

\begin{itemize}
  \item case \((0)\)
    \begin{itemize}
      \item show \( ?\text{case by simp} \)
    \end{itemize}
  \item next
  \begin{itemize}
    \item case \((\text{Suc } n)\)
    \begin{itemize}
      \item have \( \text{alternate } f \ g \ (k + \text{Suc } n) \leq \text{alternate } f \ g \ (k + n) \) using \text{assms} by (\text{auto intro: le-funI})
      \item also have \( \ldots \leq \text{alternate } f \ g \ k \) using \text{Suc by this}
      \item finally show \( ?\text{case by this} \)
    \end{itemize}
  \end{itemize}
\end{itemize}

qed

show \( \text{alternate } f \ g \ l \leq \text{alternate } f \ g \ k \)

using \text{3 unfolding 2 by this}

qed

end

2 Run Graphs

theory Graph

imports Transition-Systems-and-Automata.NBA

begin

\text{type-synonym} \ '\text{state node} = \text{nat} \times \ '\text{state}

\text{abbreviation} \ \text{ginitial } A \equiv \{0\} \times \text{initial } A

\text{abbreviation} \ \text{gaccepting } A \equiv \text{accepting } A \circ \text{snd}

\text{global-interpretation} \ \text{graph: transition-system-initial}

const

\[ \lambda \ u \ (k, \ p). \ w \ p k \in \text{alphabet } A \land u \in \{\text{Suc } k\} \times \text{transition } A \ (w \ p k) \ p \cap V \]

\[ \lambda \ v. \ v \in \text{ginitial } A \cap V \]

\text{for} \ A \ w \ V

defines

\[ \text{gpath} = \text{graph.path and gran = graph.run and} \]

\[ \text{greachable = graph.reachable and gnodes = graph.nodes} \]

\text{by this}

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

\text{declare} \ \text{graph.reachable.execute}[\text{rule del}]

\text{declare} \ \text{graph.nodes.execute}[\text{rule del}]

\text{abbreviation} \ \text{gtarget} \equiv \text{graph.target}

\text{abbreviation} \ \text{gstates} \equiv \text{graph.states}
abbreviation gtrace ≡ graph.trace

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

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

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

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

lemma gpath-path[symmetric]: path A (stake (length r) (sdrop k w) || r) p ↔ gpath A w UNIV ([Suc k ..< Suc k + length r] || r) (k, p)
proof (induct r arbitrary: k p)
case (Nil)
  show ?case by auto
next
case (Cons q r)
  have 1: path A (stake (length r) (sdrop (Suc k) w) || r) q ←→
gpath A w UNIV ((Suc (Suc k) ..< Suc k + length (q # r)) || r) (Suc k, q)
    using Cons[of Suc k q] by simp
  have stake (length (q # r)) (sdrop k w) || q # r =
    (w !! k, q) # (stake (length r) (sdrop (Suc k) w) || r) by simp
  also have path A . . . p ←→
gpath A w UNIV ((Suc k, q) # ([Suc (Suc k) ..< Suc k + length (q # r)]) || r)
    by simp
  finally show ?case by this
qed

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

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

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

qed

lemma greachable-reachable:
  fixes l q k p
  defines u ≡ (l, q)
  defines v ≡ (k, p)
  assumes u ∈ greachable A w V v
  shows q ∈ 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 ∈ successors A (snd u) using execute by auto
  have snd u ∈ reachable A p using execute by auto
  also have q ∈ reachable A (snd u) using 1 by blast
  finally show ?case by this
qed

lemma gnodes-nodes: gnodes A w V ⊆ UNIV × nodes A
proof
  fix v
  assume v ∈ gnodes A w V
  then show v ∈ UNIV × nodes A by induct auto
qed

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

lemma greachable-subset: greachable A w V v ⊆ insert v V
proof
  fix u
  assume u ∈ greachable A w V v
then show $u \in \text{insert } v V$ by induct auto

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

**lemma infinite-greachable-gtrace:**
- **assumes** grun $A w V r v$
- **assumes** $u \in \text{sset} (\text{gtrace } r v)$
- **shows** infinite $(\text{greachable } A w V u)$
- **proof** –
  - obtain $i$ where $1$: $u = \text{gtrace } r v !! i$ using sset-range imageE assms (2) by metis
  - have $2$: $\text{gtarget} (\text{stake} (\text{Suc } i) \ r) \ v = u$ unfolding 1 sscan-snth by rule
  - have infinite $(\text{sset} (\text{sdrop} (\text{Suc } i) (\text{gtrace } r v)))$
    - using gtrace-infinite[OF assms(1)]
      by (metis List.finite-set finite-Un sset-shift stake-sdrop)
  - also have $\text{sdrop} (\text{Suc } i) (\text{gtrace } r v) = \text{gtrace} (\text{sdrop} (\text{Suc } i) \ r) (\text{gtarget} (\text{stake} (\text{Suc } i) \ r) \ v)$
    - by simp
  - also have $\text{sset} ... \subseteq \text{greachable } A w V u$
    - using assms(1) 2 by (metis graphreachable.reflexive graphreachable-trace graphrun-sdrop)
  - finally show $\text{?thesis}$ by this

**lemma finite-nodes-gsuccessors:**
- **assumes** finite $(\text{nodes } A)$
- **assumes** $v \in \text{gunodes } A w$
- **shows** finite $(\text{gsuccessors } A w v)$
- **proof** –
  - have $\text{gsuccessors } A w v \subseteq \text{gureachable } A w v$ by rule
  - also have $... \subseteq \text{gunodes } A w$ using assms(2) by blast
  - also have $... \subseteq \text{UNIV } \times \text{nodes } A$ using gnodes-nodes by this
  - finally have $3$: $\text{gsuccessors } A w v \subseteq \text{UNIV } \times \text{nodes } A$ by this
  - have $\text{gsuccessors } A w v \subseteq \{\text{Suc (fst } v)\} \times \text{nodes } A$ using 3 by auto
  - also have $\text{finite ...}$ using assms(1) by simp
  - finally show $\text{?thesis}$ by this

**end**

### 3 Rankings

**theory** Ranking

**imports**

*Alternate*
3.1 Rankings

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

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

3.2 Ranking Implies Word not in Language

lemma ranking-stuck:
  assumes ranking A w f
  assumes v ∈ gunodes A w
gurun A w r v
  obtains n k
  where smap f (gtrace (sdrop n r) (gtarget (stake n r) v)) = sconst k
  proof (coinduction arbitrary: r v rule: sdescending.coinduct)
    case sdescending
      obtain u s where 1: r = u ## s using stream.exhaust by blast
      have 2: v ∈ gunodes A w using sdescending(1) by simp
      have 3: gurun A w (u ## s) v using sdescending(2) 1 by auto
      have 4: u ∈ gusuccessors A w v using 3 by auto
      have 5: u ∈ gureachable A w v using graphreachable-successors 4 by blast
    show ?case
    unfolding 1
    proof (intro exI conjI disjI1)
      show f u ≤ f v using 0 2 4 by this
      show shd (u ## gtrace s u) ∈ gunodes A w using 2 5 by auto
      show gurun A w s u using 3 by auto
    qed auto
  qed

obtain s k where 3: smap f (v ## gtrace r v) = s @-- sconst k
  using sdescending-stuck[OF 2] by metis
  have gtrace (sdrop (Suc (length s)) r) (gtarget (stake (Suc (length s)) r) v) =
    sdrop (Suc (length s)) (gtrace r v)
  using sscan-sdrop by rule
  also have smap f ... = sdrop (length s) (smap f (v ## gtrace r v))
by (metis 3 id-apply sdrop-simps(2) sdrop-smap sdrop-stl shift-eq siterate.simps(2) stream.sel(2))
also have \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 (sdrop n r) (gtarget (stake n r) v)))) odd
proof -
  obtain n k where 1: smap f (gtrace (sdrop n r) (gtarget (stake n r) v)) = sconst k
  using ranking-stuck assms by this
  have 2: gtarget (stake n r) v \in gunodes A w
    using assms(2, 3) by (simp add: graph.nodes-target graph.run-stake)
  have 3: gurun A w (sdrop n r) (gtarget (stake n r) v)
    using assms(2, 3) by (simp add: graph.run-sdrop)
  have 4: odd k using 1 2 3 assms(1) unfolding ranking-def by meson
  have 5: Ball (sset (smap f (gtrace (sdrop n r) (gtarget (stake n r) v)))) odd
    unfolding 1 using 4 by simp
  show \?thesis using that 5 by this
qed

lemma ranking-language:
  assumes ranking A w f
  shows w \notin language A
proof
  assume 1: w \in language A
  obtain r p where 2: run A (w ||| r) p p \in initial A infs (accepting A) (p ##
    r) using 1 by rule
  let ?r = fromN I ||| 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 = sdrop n ?r
  let ?u = gtarget ?s ?v
  have sset (gtrace ?t ?u) \subseteq gurachable A w ?v
proof (intro graph.reachable-trace graph.reachable-target graph.reachable.reflective)
  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 \( \ldots \subseteq \text{gunodes} \ A \ w \) using 3(1) by blast

finally have 7: \( \text{sset} (\text{gtrace} \ ?t \ ?u) \subseteq \text{gunodes} \ A \ w \) by this

have 8: \( \bigwedge \ p. \ p \in \text{gunodes} \ A \ w \Longrightarrow \text{gaccepting} \ A \ p \Longrightarrow \text{even} \ (f \ p) \) using

using assms unfolding ranking-def by auto

have 9: \( \bigwedge \ p. \ p \in \text{sset} (\text{gtrace} \ ?t \ ?u) \Longrightarrow \text{gaccepting} \ A \ p \Longrightarrow \text{even} \ (f \ p) \) using

7 8 by auto

have 19: \( \text{infs} (\text{accepting} \ A) \ (\text{smap} \ \text{snd} \ ?r) \) using 2(3) by simp

have 18: \( \text{infs} (\text{gaccepting} \ A) \ ?r \) using 19 by simp

have 17: \( \text{infs} (\text{gaccepting} \ A) \ (\text{gtrace} \ ?r \ ?v) \) using 18 unfolding gtrace-alt-def by this

have 16: \( \text{infs} (\text{gaccepting} \ A) \ (\text{gtrace} \ (\text{?s} \ @\text{?t}) \ ?v) \) using 17 unfolding stake-sdrop by this

have 15: \( \text{infs} (\text{gaccepting} \ A) \ (\text{gtrace} \ ?t \ ?u) \) using 16 by simp

have 13: \( \text{infs} (\text{even} \circ f) \ (\text{gtrace} \ ?t \ ?u) \) using infs-mono[OF - 15] 9 by simp

have 12: \( \text{infs} \text{ even} \ (\text{smap} \ f (\text{gtrace} \ ?t \ ?u)) \) using 13 by (simp add: comp-def)

have 11: \( \text{Bex (sset (smap f (gtrace ?t ?u))) even} \) using 12 infs-any by metis

show False using 4 11 by auto

qed

3.3 Word not in Language Implies Ranking

3.3.1 Removal of Endangered Nodes

definition clean :: \((\text{'label}, \text{'state}) \text{nba} \Rightarrow \text{'label stream} \Rightarrow \text{'state node set} \Rightarrow \text{'state node set where}

clean A \ w \ V \equiv \{v \in V. \ \text{infinite} (\text{greachable} \ A \ w \ V \ v)\}

lemma clean-decreasing: clean A \ w \ V \subseteq V unfolding clean-def by auto

lemma clean-successors:
assumes \( v \in V \ u \in \text{gusuccessors} \ A \ w \ v \)
shows \( u \in \text{clean} \ A \ w \ V \Longrightarrow v \in \text{clean} \ A \ w \ V \)

proof -
assume 1: \( u \in \text{clean} \ A \ w \ V \)
have 2: \( u \in V \ \text{infinite} (\text{greachable} \ A \ w \ V \ u) \) using 1 unfolding clean-def by auto

have 3: \( u \in \text{greachable} \ A \ w \ V \ v \) using graph.reachable.execute assms(2) 2(1)
by blast

have 4: \( \text{greachable} \ A \ w \ V \ u \subseteq \text{greachable} \ A \ w \ V \ v \) using 3 by blast

have 5: \( \text{infinite} (\text{greachable} \ A \ w \ V \ v) \) using 2(2) 4 by (simp add: infinite-super)

show \( v \in \text{clean} \ A \ w \ V \) unfolding clean-def using assms(1) 5 by simp

qed

3.3.2 Removal of Safe Nodes

definition prune :: \((\text{'label}, \text{'state}) \text{nba} \Rightarrow \text{'label stream} \Rightarrow \text{'state node set} \Rightarrow \text{'state node set where}

prune A \ w \ V \equiv \{v \in V. \ \exists \ u \in \text{greachable} \ A \ w \ V \ v. \ \text{gaccepting} \ A \ u\}
lemma prune-decreasing: prune A w V ⊆ V unfolding prune-def by auto

lemma prune-successors:
  assumes v ∈ V u ∈ gusuccessors A w v
  shows u ∈ prune A w V =⇒ v ∈ prune A w V

proof -
  assume 1: u ∈ prune A w V
  have 2: u ∈ V ⊃ ∃ x ∈ greachable A w V u. gaccepting A x using 1 unfolding
    prune-def by auto
  have 3: u ∈ greachable A w V v using graph.reachable.execute assms(2) 2(1)
    by blast
  have 4: greachable A w V u ⊆ greachable A w V v using 3 by blast
  show v ∈ prune A w V unfolding prune-def using assms(1) 2(2) 4 by auto
qed

3.3.3 Run Graph Iteration

definition graph :: ('label, 'state) nba ⇒ 'label stream ⇒ nat ⇒ 'state node set
  where
    graph A w k ≡ alternate (clean A w) (prune A w) k (gunodes A w)

abbreviation level A w k l ≡ {v ∈ graph A w k. fst v = l}

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

lemma graph-antimono: antimono (graph A w)
  using alternate-antimono clean-decreasing prune-decreasing
  unfolding monotone-def le-fun-def graph-def
  by metis
lemma graph-nodes: graph A w k ⊆ gunodes A w using graph-0 graph-antimono
  le0 antimonoD by metis
lemma graph-successors:
  assumes v ∈ gunodes A w u ∈ gusuccessors A w v
  shows u ∈ graph A w k =⇒ v ∈ 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 w k l ⊆ \{v ∈ gunodes A w. fst v = l\} by (simp add: graph-nodes subset-CollectI)
    also have \{v ∈ gunodes A w. fst v = l\} ⊆ \{l\} × nodes A using gnodes-nodes by force
    also have finite \{l\} × nodes A using assms(1) by simp
    finally show ?thesis by this
  qed

lemma find-safe:
  assumes w /∈ language A
  assumes V \neq {} V ⊆ gunodes A w
  assumes \( \forall v. v ∈ V =⇒ gsuccessors A w V v =\neq {} \)
  obtains v
  where v ∈ V \( \forall u ∈ greachable A w V v. \neg gaccepting A u \)
  proof (rule ccontr)
    assume 1: \( \neg \thesis \)
    have 2: \( \forall v. v ∈ V =⇒ \exists u ∈ greachable A w V v. gaccepting A u \)
      using assms(1) by auto
    have 3: \( \forall r v. v ∈ initial A =⇒ run A (w ||| r) v =⇒ fins (accepting A) r \)
      using assms(1) by auto
    obtain v where 4: v ∈ V using assms(2) by force
    obtain x where 5: x ∈ greachable A w V v gaccepting A x using 2 4 by blast
    obtain y where 50: grun A w V x inf s (\( λ x. x ∈ V \land gaccepting A x \)) r using (rule graph.recurring-condition)
      show x ∈ V \land gaccepting A x using greachable-subset 4 5 by blast
    next
    fix v
    assume 1: v ∈ V \land gaccepting A v
    obtain v’ where 20: v’ ∈ gsuccessors A w V v using assms(4) I by (meson IntE equals0I)
      have 21: v’ ∈ V using 20 by auto
      have 22: \( \exists u ∈ greachable A w V v’. u ∈ V \land gaccepting A u \)
        using greachable-subset 2 21 by blast
      obtain r where 30: gpath A w V r v’ gtarget r v’ ∈ V \land gaccepting A (gtarget r v’)
        using 22 by blast
      show 3. r. r \neq [] \land gpath A w V r v \land gtarget r v ∈ V \land gaccepting A (gtarget r v)
        proof (intro exI conjI)
          show v’ \# r \neq [] by simp
          show gpath A w V (v’ \# r) v using 20 30 by auto
          show gtarget (v’ \# r) v ∈ V using 30 by simp
          show gaccepting A (gtarget (v’ \# r) v) using 30 by simp
        qed
qed auto

obtain u where 100: u ∈ ginitial A v ∈ gureachable A w u using 4 assms(3)
by blast

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

qed

lemma remove-run:
assumes finite (nodes A) w \notin language A
assumes V ⊆ gunodes A w clean A w V ≠ {}
obtain v r
where
  grun A w V r v
  sset (gtrace r v) ⊆ clean A w V
  sset (gtrace r v) ⊆ - prune A w (clean A w V)
proof –
obtain v where 1: u ∈ clean A w V ∀ x ∈ greachable A w (clean A w V) u.
¬ gaccepting A x
proof (rule find-safe)
  show w \notin language A using assms(2) by this
  show clean A w V ≠ {} using assms(4) by this
  show clean A w V ⊆ gunodes A w using assms(3) by (meson clean-decreasing
  subset-iff)
  next
  fix v
  assume 1: v ∈ clean A w V
  have 2: v ∈ V using 1 clean-decreasing by blast
  have 3: infinite (greachable A w V v) using 1 clean-def by auto
  have gsuccessors A w V v ⊆ gsuccessors A w v by auto
  also have finite ... using 2 assms(1, 3) finite-nodes-gsuccessors by blast
  finally have 4: finite (gsuccessors A w V v) by this
  have 5: infinite (insert v ((greachable A w V) ′ (gsuccessors A w V v)))
    using graph.reachable-step 3 by metis
  obtain u where 6: u ∈ gsuccessors A w V v infinite (greachable A w V u)
  using 4 5 by auto
have 7: \( u \in \text{clean } A w V \) using 6 unfolding clean-def by auto

show \( \text{gsuccessors } A w (\text{clean } A w V) v \neq \{\} \) using 6(1) 7 by auto

qed auto

have 2: \( u \in V \) using 1(1) unfolding clean-def by auto

have 3: \( \text{infinite } (\text{greachable } A w V u) \) using 1(1) unfolding clean-def by simp

have 4: \( \text{finite } (\text{gsuccessors } A w V v) \) if \( v \in \text{greachable } A w V u \) for \( v \)

proof –

have 1: \( v \in V \) using that greachable-subset 2 by blast

have \( \text{gsuccessors } A w V v \subseteq \text{gsuccessors } A w v \) by auto

also have \( \text{finite } \ldots \) using 1 assms (1, 3) finite-nodes-gsuccessors by blast

finally show \( \text{thesis} \) by this

qed

obtain \( r \) where 5: \( \text{grun } A w V r u \) using graph.koenig[OF 3 4] by this

have 6: \( \text{greachable } A w V u \subseteq V \) using 2 greachable-subset by blast

have 7: \( \text{sset } (\text{gtrace } r u) \subseteq V \) unfolding graph.reachable-trace[OF graph.reachable.reflexive 5(1)] 6 by blast

have 8: \( \text{sset } (\text{gtrace } r u) \subseteq \text{clean } A w V \)

unfolding clean-def using 7 infinite-greachable-gtrace[OF 5(1)] by auto

have 9: \( \text{sset } (\text{gtrace } r u) \subseteq \text{greachable } A w (\text{clean } A w V) u \)

using 5 8 by (metis graph.reachable.reflexive graph.reachable-trace grun-subset)

show \( \text{thesis} \)

proof

show \( \text{grun } A w V r u \) using 5(1) by this

show \( \text{sset } (\text{gtrace } r u) \subseteq \text{clean } A w V \) using 8 by this

show \( \text{sset } (\text{gtrace } r u) \subseteq \text{prune } A w (\text{clean } A w V) \)

proof (intro subsetI ComplI)

fix \( p \)

assume 10: \( p \in \text{sset } (\text{gtrace } r u) \) \( p \in \text{prune } A w (\text{clean } A w V) \)

have 20: \( \exists x \in \text{greachable } A w (\text{clean } A w V) \) \( p.\text{gaccepting } A x \)

using 10(2) unfolding prune-def by auto

have 30: \( \text{greachable } A w (\text{clean } A w V) p \subseteq \text{greachable } A w (\text{clean } A w V) \)

using 10(1) 9 by blast

show \( \text{False} \) using 1(2) 20 30 by force

qed

qed

qed

lemma level-bounded:

assumes \( \text{finite } (\text{nodes } A) \) \( w \notin \text{language } A \)

obtains \( n \)

where \( \land l. l \geq n \implies \text{card } (\text{level } A w (2 * k) l) \leq \text{card } (\text{nodes } A) - k \)

proof (induct \( k \) arbitrary: thesis)

|case 0|

show \( ?case \)

proof (rule 0)

fix \( l :: \text{nat} \)

have \( \text{finite } \{l\} \times \text{nodes } A \) using assms(1) by simp

also have \( \text{finite } \text{level } A w 0 l \subseteq \{l\} \times \text{nodes } A \) using gnodes-nodes by force

14
also \( \text{(card-mono)} \) have \( \text{card} \ldots = \text{card} (\text{nodes} \ A) \) using \text{assms(1)} \ by \ simp
finally show \( \text{card} (\text{level} \ A \ w (2 ^* 0) \ l) \leq \text{card} (\text{nodes} \ A) - 0 \) \ by \ simp
qed
next
case \( \text{Suc} \ k \)
show ?case
proof (cases graph \( A \ w (\text{Suc} \ (2 ^* k)) = \{\} )
case True
have \( 3: \text{graph} \ A \ w (2 ^* \text{Suc} \ k) = \{\} \) \ using \ True \ prune-decreasing \ by \ simp
blast
show ?thesis using \text{Suc(2)} \ 3 \ by \ simp
next
case False
obtain \( v \ r \) \ where \( \text{I}: \)
\( \text{grun} \ A \ w (\text{graph} \ A \ w (2 ^* \text{Suc} \ k)) \ r \ v \)
\( \text{sset} (\text{gtrace} \ r \ v) \subseteq \text{graph} \ A \ w (\text{Suc} \ (2 ^* k)) \)
\( \text{sset} (\text{gtrace} \ r \ v) \subseteq - \text{graph} \ A \ w (\text{Suc} \ (\text{Suc} \ (2 ^* k))) \)
proof (rule remove-run)
show finite \( (\text{nodes} \ A) \) \ w \( \notin \text{language} \ A \) \ using \text{assms by this}
show clean \( A \ w (\text{graph} \ A \ w (2 ^* k)) \neq \{\} \) \ using \ False \ by \ simp
show graph \( A \ w (2 ^* k) \subseteq \text{gunodes} \ A \ w \) \ using \ graph-nodes \ by \ this
qed auto
obtain \( l \ q \) \ where \( 2: v = (l, q) \) \ by \ force
obtain \( n \) \ where \( 90: \bigwedge l. \ n \leq l \implies \text{card} (\text{level} \ A \ w (2 ^* k) \ l) \leq \text{card} (\text{nodes} \ A) - k \)
using \text{Suc(1)} \ by \ blast
show ?thesis
proof (rule Suc(2))
fix \( j \)
assume \( 100: \ n + \text{Suc} \ l \leq j \)
have \( 6: \text{graph} \ A \ w (\text{Suc} \ (\text{Suc} \ (2 ^* k))) \subseteq \text{graph} \ A \ w (\text{Suc} \ (2 ^* k)) \)
using \ graph-antimono \ antimono-iff-le-Suc \ by \ blast
have \( 101: \text{gtrace} \ r \ v \ (j - \text{Suc} \ l) \in \text{graph} \ A \ w (\text{Suc} \ (2 ^* k)) \) \ using \text{I(2)}
snth-sset \ by \ auto
have \( 102: \text{gtrace} \ r \ v \ (j - \text{Suc} \ l) \notin \text{graph} \ A \ w (\text{Suc} \ (\text{Suc} \ (2 ^* k))) \) \ using \text{I(3)}
snth-sset \ by \ blast
have \( 103: \text{gtrace} \ r \ v \ (j - \text{Suc} \ l) \in \text{level} \ A \ w (\text{Suc} \ (2 ^* k)) \)
using \text{I(1)} \ 100 \ 101 \ 2 \ by \ (auto \ elim: \text{grun-elim})
have \( 104: \text{gtrace} \ r \ v \ (j - \text{Suc} \ l) \notin \text{level} \ A \ w (\text{Suc} \ (\text{Suc} \ (2 ^* k))) \) \ j \ using \text{I(4)}
100 \ 102 \ by \ simp
have level \( A \ w (2 ^* \text{Suc} \ k) \ j = \text{level} \ A \ w (\text{Suc} \ (2 ^* k)) \)
also have \( \ldots \subseteq \text{level} \ A \ w (\text{Suc} \ (2 ^* k)) \) \ j \ by \ simp
also have \( \ldots \subseteq \text{level} \ A \ w (2 ^* k) \) \ j \ by \ simp add: \text{Collect-mono} \ clean-def
finally have \( 105: \text{level} \ A \ w (2 ^* \text{Suc} \ k) \ j \subset \text{level} \ A \ w (2 ^* k) \) \ j \ by \ this
have \( \text{card} (\text{level} \ A \ w (2 ^* \text{Suc} \ k) \ j) < \text{card} (\text{level} \ A \ w (2 ^* k) \ j) \)
using \assms(1) \ 105 \ by \ simp \ add: \graph-level-finite \ psubset-card-mono
also have \( \ldots \leq \text{card} (\text{nodes} \ A) - k \) \ using \text{90} \ 100 \ by \ simp
finally show \( \text{card} (\text{level} \ A \ w (2 ^* \text{Suc} \ k) \ j) \leq \text{card} (\text{nodes} \ A) - \text{Suc} \ k \) \ by \ simp
qed

lemma graph-empty:
  assumes finite (nodes A) w /∈ language A
  shows graph A w (Suc (2 * card (nodes A))) = {}
proof -
  obtain n where 1: \( l \geq n \Rightarrow \text{card}(\text{level } A w (2 * \text{card} (\text{nodes } A)) l) = 0 \)
  using level-bounded[OF assms(1, 2), of card (nodes A)] by auto
  have (\( \bigcup l \in \{..< n\}. \text{level } A w (2 * \text{card} (\text{nodes } A)) l \)) ∪
    (\( \bigcup l \in \{n ..\}. \text{level } A w (2 * \text{card} (\text{nodes } A)) l \))
  by auto
also have (\( \bigcup l \in \{n ..\}. \text{level } A w (2 * \text{card} (\text{nodes } A)) l \)) ∪ \{}
  using graph-level-finite assms(1) by fastforce
finally have 100: finite (graph A w (2 * card (nodes A))) by this
have 101: finite (reachable A w (graph A w (2 * card (nodes A)) v)) for v
  using 100 reachable-subset[of A w graph A w (2 * card (nodes A)) v]
  using finite-insert infinite-super by auto
show \(?thesis using 101 by (simp add: clean-def)\)
qed

lemma graph-le:
  assumes finite (nodes A) w /∈ language A
  assumes v ∈ graph A w k
  shows k ≤ 2 * card (nodes A)
proof (metis Suc-leI empty-iff monotone-def not-le-imp-less rev-subsetD)

3.4 Node Ranks

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

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

lemma rank-removed:
  assumes finite (nodes A) w /∈ language A
  shows v /∈ graph A w (Suc (rank A w v))
proof
  assume v ∈ graph A w (Suc (rank A w v))
then have 2: Suc (rank A w v) ≤ rank A w v
  unfolding rank-def using Greatest-le-nat graph-le assms by metis
then show False by auto
qed

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

lemma language-ranking:
  assumes finite (nodes A) w \notin language A
  shows ranking A w (rank A w)
unfolding ranking-def
proof (intro conjI ballI allI impI)
  fix v
  assume 1: v ∈ gunodes A w
  have 2: v ∈ graph A w (rank A w v) using rank-member assms 1 by this
  show rank A w u ≤ 2 * card (nodes A) using graph-le assms 2 by this
next
  fix v u
  assume 1: v ∈ gunodes A w u ∈ gusuccessors A w v
  show rank A w u ≤ rank A w v using rank-le assms 1 by this
next
  fix v
  assume 1: v ∈ gunodes A w gaccepting A v
  have 2: v ∈ graph A w (rank A w v) using rank-member assms 1(1) by this
  have 3: v \notin graph A w (Suc (rank A w v)) using rank-removed assms by this
  have 4: v ∈ prune A w (graph A w (rank A w v)) using 2 1(2) unfolding prune-def by auto
  have 5: graph A w (Suc (rank A w v)) \neq prune A w (graph A w (rank A w v)) using 3 4 by blast
  show even (rank A w v) using 5 by auto
next
  fix v r k
\textbf{Assume 1:} \(v \in \text{gunodes } A \ w \ r \ v \ \text{smap} \ (\text{rank } A \ w) \ (\text{gtrace } r \ v) = \text{sconst } k\)

\textbf{Have 1:} \(\text{sset} \ (\text{gtrace } r \ v) \subseteq \text{gunode } A \ w\)

\textbf{Using 1(2) by} \text{(metis graph.reachable.reflexive graph.reachable-trace)}

\textbf{Then have 6:} \(\text{sset} \ (\text{gtrace } r \ v) \subseteq \text{gunodes } A \ w\) \text{ using 1(1) by blast}

\textbf{Have 60:} \(\text{rank } A \ w \ \text{ sset} \ (\text{gtrace } r \ v) \subseteq \{k\}\)

\textbf{Using 1(3) by} \text{(metis equalityD1 sset-sconst stream.set-map)}

\textbf{Then have 6:} \(\text{sset} \ (\text{gtrace } r \ v) \subseteq \text{graph } A \ w\)

\textbf{Using rank-member[OF assms\] subsetD[OF 6] 60 unfolding image-subset-iff by auto}

\textbf{Have 70:} \(\text{grun } A \ w \ (\text{graph } A \ w \ k) \ r \ v\) \text{ using grun-subset 1(2) 50 by this}

\textbf{Unfolding clean-def using 50 infinite-greachable-gtrace[OF 70] by auto}

\textbf{Have 8:} \(\text{sset} \ (\text{gtrace } r \ v) \cap \text{graph } A \ w \ (\text{Suc } k) = \{\}\) \text{ using rank-removed[OF assms\] 60 by blast}

\textbf{Have 9:} \(\text{sset} \ (\text{gtrace } r \ v) \neq \{\}\) \text{ using stream.set.sel(1) by auto}

\textbf{Have 10:} \(\text{graph } A \ w \ (\text{Suc } k) \neq \text{clean } A \ w \ (\text{graph } A \ w \ k)\) \text{ using 7 8 9 by blast}

\textbf{Show odd }k \text{ using 10 unfolding graph-Suc by auto}

\textbf{Qed}

3.5 Correctness Theorem

\textbf{Theorem language-ranking-iff:}

\textbf{Assumes} \(\text{finite (nodes } A\)}

\textbf{Shows} \(w \notin \text{language } A \iff (\exists f. \text{ranking } A \ w \ f)\)

\textbf{Using} \text{ranking-language language-ranking assms by blast}

end

4 Complementation

\textbf{Theory} Complementation

\textbf{Imports}

\textit{Transition-Systems-and-Automata.Maps}

\textit{Ranking}

\textbf{Begin}

4.1 Level Rankings and Complementation States

\textbf{Type-Synonym} \(\text{’state } lr = \text{’state } \rightarrow \text{nat}\)

\textbf{Definition} \(\text{lr-succ } :: (\text{’label } \text{, ’state}) \text{ nba } \Rightarrow \text{’state } lr \Rightarrow \text{’state } lr \text{ set where}

\text{lr-succ } A \ a \ f = \{g.

\text{dom } g = \bigcup (\text{transition } A \ a \ \text{’dom } f) \land

(\forall p \in \text{dom } f. \forall q \in \text{transition } A \ a \ p. \text{ the } (g \ q) \leq \text{ the } (f \ p)) \land

(\forall q \in \text{dom } g. \text{ accepting } A \ q \Rightarrow \text{ even } \text{ the } (g \ q)))\}

\textbf{Type-Synonym} \(\text{’state } st = \text{’state set}\)
definition \textit{st-succ} :: (\textit{label}, \textit{state}) \textit{nba} \Rightarrow (\textit{state}) \textit{lr} \Rightarrow (\textit{state}) \textit{st} \Rightarrow (\textit{state}) \textit{st} where
\textit{st-succ} A a P \equiv \{ q \in \text{dom} g \mid \text{if } P = \emptyset \text{ then dom } g \text{ else } \bigcup (\text{transition} A a \; \cdot \; P) \}.
\text{even} (\text{the} (g \; q))

\text{type-synonym} \; (\textit{state}) \textit{cs} = (\textit{state}) \textit{lr} \times (\textit{state}) \textit{st}

\text{definition} \textit{complement-succ} :: (\textit{label}, \textit{state}) \textit{nba} \Rightarrow (\textit{state}) \textit{cs} \Rightarrow (\textit{state}) \textit{cs} \textit{set} where
\textit{complement-succ} A a \equiv \lambda \; (f, \; P). \{ (g, \; \text{st-succ} A a g P) \mid g \in \text{lr-succ} A a \}

\text{definition} \textit{complement} :: (\textit{label}, \textit{state}) \textit{nba} \Rightarrow (\textit{label}, \textit{state}) \textit{cs} \textit{nba} where
\textit{complement} A \equiv \textit{nba} (\text{alphabet} A)
\{ (\text{const} (\text{Some} (2 \ast \text{card} (\text{nodes} A))) \; \cdot \; ' \text{ initial} A) \times \{ \emptyset \}) \}
\text{complement-def} \text{ complement-succ-def} \text{ lr-succ-def} \text{ by (induct) (auto, blast)}

\text{lemma} \textit{dom-nodes}: \text{assumes} f P \in \text{nodes} \; (\text{complement} A)
\text{shows} \; \text{dom} (\text{fst} f P) \subseteq \text{nodes} A
\text{using assms unfolding complement-def complement-succ-def lr-succ-def by (induct) (auto, blast)}

\text{lemma} \textit{ran-nodes}: \text{assumes} f P \in \text{nodes} \; (\text{complement} A)
\text{shows} \; \text{ran} (\text{fst} f P) \subseteq \{ 0 .. 2 \ast \text{card} (\text{nodes} A) \}
\text{using assms}
\text{proof induct}
\text{case (initial} f P)
\text{show ?case using initial unfolding complement-def by (auto) (metis eq-refl option.inject ran-restrictD)}
\text{next}
\text{case (execute} f P a g Q)
\text{obtain} f P \text{ where 1: } f P = (f, \; P) \text{ by force}
\text{have 2: } \text{ran } f \subseteq \{ 0 .. 2 \ast \text{card} (\text{nodes} A) \} \text{ using execute(2) unfolding I by auto}
\text{obtain a g Q where 3: } a g Q = (a, \; (g, \; Q)) \text{ using prod-cases3 by this}
\text{have 4: } p \in \text{dom } f \Longrightarrow q \in \text{transition} A \; a \; p \Longrightarrow (g \; q) \leq (f \; p) \text{ for } p \; q
\text{using execute(3)}
\text{unfolding 1 3 complement-def nba.simps complement-succ-def lr-succ-def by simp}
\text{have 8: } \text{dom } g = \bigcup ((\text{transition} A \; a) \; \cdot \; (\text{dom} f))
\text{using execute(3)}
\text{unfolding 1 3 complement-def nba.simps complement-succ-def lr-succ-def by simp}
\text{show ?case unfolding 1 3 ran-def}
\text{proof safe}
fix q k
assume 5: fst (snd (a, (g, Q))) q = Some k
have 6: q ∈ dom g using 5 by auto
obtain p where 7: p ∈ dom f q ∈ transition A a p using 6 unfolding 8 by auto
have k = the (g q) using 5 by auto
also have ... ≤ the (f p) using 4 7 by this
also have ... ≤ 2 * card (nodes A) using 2 7(1) by (simp add: domD ranI)
finally show k ∈ {0 .. 2 * card (nodes A)} by auto
qed

lemma states-nodes:
assumes fP ∈ nodes (complement A)
shows snd fP ⊆ nodes A
using assms
proof induct
  case (initial fP)
  show ?case using initial unfolding complement-def by auto
next
  case (execute fP agQ)
  obtain f P where 1: fP = (f, P) by force
  have 2: P ⊆ nodes A using execute(2) unfolding 1 by auto
  obtain a g Q where 3: agQ = (a, (g, Q)) using prod-cases3 by this
  have 11: a ∈ alphabet A using execute(3) unfolding 3 complement-def by auto
  have 10: (g, Q) ∈ nodes (complement A) using execute(1, 3) unfolding 1 3 by auto
  have 4: dom g ⊆ nodes A using dom-nodes[OF 10] by simp
  have 5: ∪ (transition A a ° P) ⊆ nodes A using 2 11 by auto
  have 6: Q ⊆ nodes A
  using execute(3)
  unfolding 1 3 complement-def nba.simps complement-succ-def st-succ-def
  using 4 5
  by (auto split: if-splits)
  show ?case unfolding 3 by auto
qed

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

lemma complement-trace-snth:
  assumes run (complement A) (w || r) p
  defines m ≡ p ## trace (w || r) p
  obtains
    \( \text{fst} (m !! \text{Suc } k) \in \text{lr-succ } A (w !! k) (\text{fst} (m !! k)) \)
    \( \text{snd} (m !! \text{Suc } k) = \text{st-succ } A (w !! k) (\text{fst} (m !! \text{Suc } k)) (\text{snd} (m !! k)) \)

proof
  have 1: r !! k ∈ transition (complement A) (w !! k) (m !! k) using nba.run-snth
  using assms by force
  show \( \text{fst} (m !! \text{Suc } k) \in \text{lr-succ } A (w !! k) (\text{fst} (m !! k)) \)
    using assms (2) 1 unfolding complement-def complement-succ-def nba.trace-alt-def
    by auto
  show \( \text{snd} (m !! \text{Suc } k) = \text{st-succ } A (w !! k) (\text{fst} (m !! \text{Suc } k)) (\text{snd} (m !! k)) \)
    using assms (2) 1 unfolding complement-def complement-succ-def nba.trace-alt-def
    by auto
  qed

4.2 Word in Complement Language Implies Ranking

lemma complement-ranking:
  assumes w ∈ language (complement A)
  obtains f
  where ranking A w f

proof
  obtain r p where 1: run (complement A) (w || r) p
  p ∈ initial (complement A)
  \( \text{infs} (\text{accepting} (\text{complement } A)) (p ## r) \)
  using assms by rule
  let \(?m = p ## r\)
  obtain 100:
    \( \text{fst} (?m !! \text{Suc } k) \in \text{lr-succ } A (w !! k) (\text{fst} (?m !! k)) \)
    \( \text{snd} (?m !! \text{Suc } k) = \text{st-succ } A (w !! k) (\text{fst} (?m !! \text{Suc } k)) (\text{snd} (?m !! k)) \)
  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 (\( \text{fst} (?m !! k) q \))
  define P where \( P k \equiv \text{snd} (?m !! k) \) for k
  have 2: \( \text{snd} v \in \text{dom} (\text{fst} (?m !! fst v)) \) if \( v \in \text{gunodes } A w \) 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 \( \text{snd} u \in \bigcup (\text{transition } A (w !! fst v) \cdot \text{dom} (\text{fst} (?m !! fst v))) \)
    using execute(2, 3) by auto
also have \dots = \text{dom} (\text{fst} (\text{Suc} (\text{fst} v)))

using 100 unfolding \text{l-r-succ-def} by simp
also have Suc (\text{fst} v) = \text{fst} u using execute(3) by auto
finally show \text{case by this}
qed

have 3: \text{f u} \leq \text{f v} if 10: v \in \text{gunodes A w} and 11: u \in \text{gusuccessors A w v for u v}

proof –
  have 15: snd u \in \text{transition A (w !! \text{fst} v)} (snd v) using 11 by auto
  have 16: snd v \in \text{dom} (\text{fst} (\text{Suc} (\text{fst} v))) using \geq 10 by this
  have f u = the (\text{fst} (\text{Suc} (\text{fst} u)) (snd u)) unfolding \text{f-def} by (simp add: case-prod-beta)
also have \text{fst u} = \text{Suc} (\text{fst} v) using 11 by auto
also have \text{the} (\text{fst} (\text{Suc} (\text{fst} u)) (snd u)) \leq \text{the} (\text{fst} (\text{Suc} (\text{fst} v)) (snd v))

using 100 15 16 unfolding \text{l-r-succ-def} by auto
also have \ldots = f v unfolding \text{f-def} by (simp add: case-prod-beta)
finally show \text{f u} \leq f v by this
qed

have 4: \exists l \geq k. P l = {} for k

proof –
  have 15: infs (\lambda (k, P). P = {}) using 1(3) unfolding \text{complement-def} by auto
obtain l where 17: l \geq k snd (?m !! l) = {} using 15 unfolding infs-snth by force
  have 19: P l = {} unfolding \text{P-def} using 17 by auto
show ?thesis using 19 17(1) by auto
qed

show ?thesis
proof (rule that, unfold \text{ranking-def}, intro conjI ballI impI allI)
  fix v
  assume v \in \text{gunodes A w}
  then show f v \leq 2 \ast \text{card (nodes A)}
  proof induct
    case (initial v)
    then show ?case using 1(2) unfolding \text{complement-def f-def} by auto
  next
case (execute v u)
  have f u \leq f v using 3[OF execute(1)] execute(3) by simp
also have \ldots \leq 2 \ast \text{card (nodes A)} using execute(2) by this
finally show ?case by this
qed

next
  fix v u
  assume 10: v \in \text{gunodes A w}
  assume 11: u \in \text{gusuccessors A w v}
  show f u \leq f v using 3 10 11 by this
next
  fix v
  assume 10: v \in \text{gunodes A w}
assume 11: \text{gaccepting} A v
show even \((f v)\)
using 10
proof cases
  case (initial)
  then show \(?\text{thesis} by 1\(2\)\) unfolding \(\text{complement-def}\) \(\text{f-def}\) by auto
next
case (execute u)
  have 12: \(\text{snd} v \in \text{dom} (\text{fst} (\text{Suc} (f u)))\) using \(\text{execute}\) \(\text{graph}\).\(\text{nodes}\).\(\text{execute}\)
2 by blast
  have 12: \(\text{snd} v \in \text{dom} (\text{fst} (\text{Suc} (f u)))\) using 12 \((\text{execute}\(2\))\) by auto
  have 13: accepting \((\text{snd} v)\) using \(11\) by auto
  have \(f v = \text{the} (\text{fst} (\text{Suc} f) (\text{snd} v))\) unfolding \(\text{f-def}\) by \((\text{simp add: case-prod-beta})\)
  also have \(\text{fst} v = \text{Suc} (\text{fst} u)\) using \(\text{execute}(2)\) by auto
  also have even \((\text{the} (\text{fst} (\text{Suc} f) (\text{snd} v)))\)
    using 100 12 13 unfolding \(\text{br-succ-def}\) by simp
finally show \(?\text{thesis} by this\)
qed
next
fix \(v\) \(s\) \(k\)
assume 10: \(v \in \text{gnodes} A w\)
assume 11: \(\text{grun} A w s v\)
assume 12: \(\text{smap} f (\text{gtrace} s v) = \text{sconst} k\)
show odd \(k\)
proof
  assume 13: even \(k\)
obtain \(t\) \(u\) where 14: \(u \in \text{ginitial}\) \(A\) \(\text{gapath} A w t u v = \text{gtarget} t u\) using
10 by auto
obtain \(l\) where 15: \(l \geq \text{length} t P l = \{\}\) using 4 by auto
have 30: \(\text{grun} A w (t \text{ at } s)\) \(u\) using 11 14\) by auto
have 21: \(\text{fst} (\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u) = \text{Suc} l\) for \(l\)
unfolding \text{sscan-snth}\).\text{symmetric}\) using 30\(14(1)\)\) by \((\text{auto elim:} \text{frac}\).\text{elim}\)\)
have 17: \(\text{snd} (\text{gtarget} (\text{stake} (\text{Suc} l + i) (t \text{ at } s)) u) \in P (\text{Suc} l + i)\) for \(i\)
proof (induct \(i\))
  case \(0\)
  have 20: \(\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u \in \text{gnodes} A w\)
using 14 11 by \((\text{force simp add:} \text{15(1)}\) \(\text{le-Suc}\).\text{graph}.\text{run-stake} \text{stake-shift})
  have \(\text{snd} (\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u) \in \text{dom} (\text{fst} (\text{Suc} f) (\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u))\))
using 2\(\text{OF}\) 20\) by \(\text{this}\)
  also have \(\text{fst} (\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u) = \text{Suc} l\) using 21\) by \(\text{this}\)
finally have 22: \(\text{snd} (\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u) \in \text{dom} (\text{fst} (\text{Suc} l))\) by \(\text{this}\)
  have \(\text{gtarget} (\text{stake} (\text{Suc} l) (t \text{ at } s)) u = \text{gtrace} (t \text{ at } s)\) \(u\) \(!! l\) unfolding \(\text{sscan-snth}\) \(\text{by rule}\)
also have \(\ldots = \text{gtrace} s v\) \(!! (l \text{ at } \text{length} t)\) using \(\text{15(1)}\) by simp
also have \(f \ldots = \text{smap} f (\text{gtrace} s v) \ldots \text{at} (l \text{ at } \text{length} t)\) by simp

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

finally show ?case by this

qed

obtain \( l' \) where \( 16: \ l' \geq \text{Suc } l \) \( P \ l' = \{\} \) using 4 by auto

show False using \( 16 \ 17 \) using {nat-le-iff-add} by auto

qed

4.3 Ranking Implies Word in Complement Language

definition reach where
reach A w i \equiv \{ target r p | r. p. path A r p \land p \in \text{initial } A \land \text{map fst } r = \text{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 \text{initial } A \text{ map fst } r = \text{stake } (\text{Suc } n) \ w 
  using / unfolding \{reach-def\} by blast

  have 3: path A (\text{take } n \ r \ @ \text{drop } n \ r) \ p \ using \{2(2)\} \ by simp
  have 4: map \text{fst } r = \text{stake } n \ w \ @ \{w !! n\} \ using \{2(4)\} \ \text{stake-Suc} \ by auto

  have 5: map \text{snd } r = \text{take } n \ (\text{map snd } r) \ @ \{q\} \ using \{2(1), 4\} \ \text{stake-Suc}

  have 6: \text{drop } n \ r = \{(w !! n, q)\} \ using \{4, 5\}

  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 (\text{transition } A (w !! n) \ i \ 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 \text{initial } A \text{ map fst } r = \text{stake } (\text{Suc } n) \ w 
  using / unfolding \{reach-def\} by blast

  have 3: path A (\text{take } n \ r \ @ \text{drop } n \ r) \ p \ using \{2(2)\} \ by simp
  have 4: map \text{fst } r = \text{stake } n \ w \ @ \{w !! n\} \ using \{2(4)\} \ \text{stake-Suc} \ by auto
have 5: map snd r = take n (map snd r) @ [q] using 2(1, 4) 4
by (metis One-nat-def Suc-inject Suc-neq-Zero Suc-pred append.right-neutral append-eq-conv-conj drop-map id-take-nth-drop last-ConsR last-conv-nth length-0-conv
length-map length-stake lessI nba.target-alt-def nba.states-alt-def zero-less-Suc)

have 6: drop n r = [(w !! n, q)] using 4 5
by (metis append-eq-conv-conj append-is-Nil-conv append-take-drop-id drop-map length-greater-0-conv length-stake stake-cycle-le stake-invert-Nil take-map zip-Cons-Cons zip-map-fst-snd)

show q ∈ ∪((transition A (w !! n) · (reach A w n))) unfolding reach-def
proof (intro UN-I CollectI exI conjI)
show target (take n r) p = target (take n r) p by rule
show path A (take n r) p using 3 by blast
show p ∈ initial A using 2 by (metis length-stake lessI nat.distinct(1)
stake-cycle-le stake-invert-Nil take-map take-stake)
show q ∈ transition A (w !! n) (target (take n r) p) using 3 unfolding 6
by auto
qed

next

fix p q
assume 1: p ∈ reach A w n q ∈ transition A (w !! n) p
obtain r x where 2: p = target r x path A r x x ∈ initial A map fst r = stake n w
using 1(1) unfolding reach-def by blast
show q ∈ reach A w (Suc n)
unfolding reach-def
proof (intro CollectI exI conjI)
show q = target (r @ [(w !! n, q)]) x using 1 2 by auto
show path A (r @ [(w !! n, q)]) x using assms 1(2) 2(1, 2) by auto
show x ∈ initial A using 2(3) by this
show map fst (r @ [(w !! n, q)]) = stake (Suc n) w using 1 2
by (metis eq-fst-iff list.simps(8) list.simps(9) map-append stake-Suc)
qed

lemma reach-Suc[simp]: reach A w (Suc n) = (if w !! n ∈ alphabet A
then ∪ (transition A (w !! n) · (reach A w n)) else {})
using reach-Suc-empty reach-Suc-succ by metis

lemma reach-nodes: reach A w i ⊆ nodes A by (induct i) (auto)
lemma reach-gunodes: {i} × reach A w i ⊆ gunodes A w
by (induct i) (auto intro: graph.nodes.execute)

lemma ranking-complement:
assumes finite (nodes A) w ∈ streams (alphabet A) ranking A w f
shows w ∈ language (complement A)
proof –
define f’ where f’ ≡ λ (k, p). if k = 0 then 2 * card (nodes A) else f (k, p)
have 0: ranking A w f'
unfolding ranking-def
proof (intro conjI ballI impI allI)
  show \( \forall v. v \in \text{ganynodes } A \implies f' \leq 2 \ast \text{card}(\text{nodes } A) \)
    using assms(3) unfolding ranking-def f'-def by auto
  show \( \forall u, v \in \text{ganynodes } A \implies u \in \text{gsuccessors } A \implies v \implies f' u \leq f' v \)
    using assms(3) unfolding ranking-def f'-def by fastforce
  show \( \forall v. v \in \text{ganynodes } A \implies \text{acceping } A \implies \text{even } (f' v) \)
    using assms(3) unfolding ranking-def f'-def by auto
next
have 1: \( v \in \text{ganynodes } A \implies \text{gunrun } A \implies \text{smap } f (\text{gtrace } r v) = \text{sconst} k \)
  k \implies \text{odd } k
  for v r k using assms(3) unfolding ranking-def by meson
fix v r k
assume 2: \( v \in \text{ganynodes } A \implies \text{guruun } A \implies \text{smap } f' (\text{gtrace } r v) = \text{sconst } k \)
have 20: \( \text{shd } r \in \text{gureachable } A \implies \text{v } \)
  using 2
by (auto) (metis graph.reachable.reflexive.graph.reachable-trace gtrace-alt-def subsetD shd-sset)
obtain 3:
  \( \text{shd } r \in \text{ganynodes } A \)
  \( \text{gunrun } A \implies \text{guruun } A \implies \text{smap } f' (\text{gtrace } (\text{shd } r)) = \text{sconst } k \)
using 2 by (metis graph.nodes-trans graph.run-scons-elim siterate.simps(2) sscon.simps(2) stream.map-sel(2))
have 4: \( k \neq 0 \) if \( (k, p) \in \text{ssset } r \) for \( k p \)
proof --
  obtain ru ka pa where 1: \( r = \text{fromN } (\text{Suc } ka) \)
    using grun-elim[OF 2(2)] by this
  have 2: \( k \in \text{ssset } (\text{Suc } ka) \) using 1(1) that
    by (metis image-eql prod.sel(1) szip-smap-fst stream.set-map)
  show ?thesis using 2 by simp
qed
have 5: \( \text{smap } f' (\text{gtrace } (\text{shd } r)) = \text{smap } f (\text{gtrace } (\text{shd } r)) \)
proof (rule stream.map-cong)
  show gtrace (shd r) = gtrace (shd r) by rule
next
fix z
assume 1: \( z \in \text{ssset } (\text{gtrace } (\text{shd } r)) \)
have 2: \( \text{fst } z \neq 0 \) using 4 1 by (metis gtrace-alt-def prod.collapse stl-sset)
  show f' z = f z using 2 unfolding f'-def by (auto simp: case-prod-beta)
qed
show odd k using 1 3 5 by simp
qed

define g where g i p \equiv \text{if } p \in \text{reach } A \implies \text{Some } (f'(i, p)) \) else None for
i p
have g-dom[simp]: dom (g i) = reach A \ w i for i
unfolding g-def by (auto) (metis option.simps(3))
have g-0[simp]: g 0 = \text{const } (\text{Some } (2 \ast \text{card}(\text{nodes } A)) | \text{\ initial } A \}
27
unfolding g-def f'-def by auto
have g-Suc[simp]: g (Suc n) ∈ lr-succ A (w !! n) (g n) for n
unfolding lr-succ-def
proof (intro CollectI conjI ballI impI)
  show dom (g (Suc n)) = ∪ (transition A (w !! n) ‘ dom (g n)) using snth-in assms(2) by auto
next
  fix p q
  assume 100: p ∈ dom (g n) q ∈ transition A (w !! n) p
  have 101: q ∈ reach A w (Suc n) using snth-assms(2) 100 by auto
  have 102: (n, p) ∈ gunodes A w using 100(1) reach-gunodes g-dom by blast
  have 103: (Suc n, q) ∈ gusuccessors A w (n, p) using snth-assms(2) 102
  100(2) by auto
  have 104: p ∈ reach A w n using 100(1) by simp
  have g (Suc n) q = Some (f' (Suc n, q)) using 101 unfolding g-def by simp
  also have the . . . = f' (Suc n, q) by simp
  also have . . . ≤ f' (n, p) using 0 unfolding ranking-def using 102 103 by simp
  also have . . . = the (Some (f' (n, p))) by simp
  also have Some (f' (n, p)) = g n p using 104 unfolding g-def by simp
  finally show the (g (Suc n) q) ≤ the (g n p) by this
next
  fix p
  assume 100: p ∈ dom (g (Suc n)) accepting A p
  have 101: p ∈ reach A w (Suc n) using 100(1) by simp
  have 102: (Suc n, p) ∈ gunodes A w using 101 reach-gunodes by blast
  have 103: gaccepting A (Suc n, p) using 100(2) by simp
  have the (g (Suc n) p) = f' (Suc n, p) using 101 unfolding g-def by simp
  also have even . . . using 0 unfolding ranking-def using 102 103 by auto
  finally show even (the (g (Suc n) p)) by this
qed

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

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

show ?thesis
proof
  show run (complement A) (w ||| stl s) (shd s)
proof (intro nba.snth-run conjI, simp-all del: stake.simps stake-szip)
  fix k
  show w !! k ∈ alphabet (complement A) using snth-in assms(2) unfolding
  complement-def by auto
    have stl s !! k = s !! Suc k by simp
  also have ... ∈ complement-succ A (w !! k) (s !! k)
    unfolding complement-succ-def s-def using P-Suc by simp
  also have: ... = complement-succ A (w !! k) (target (stake k (w ||| stl s))
    unfolding sscan-scons-snth P-Suc by simp
  finally show stl s !! k ∈
    transition (complement A) (w !! k) (target (stake k (w ||| stl s)) (shd s))
  by this
  show shd s ∈ initial (complement A) unfolding complement-def s-def using
  P-0 by simp
  show infs (accepting (complement A)) (shd s ## stl s)
  proof (rule ccontr)
    assume 20: ∀ n. ∃ k ≥ n. P k = {}
    proof (rule ccontr)
      assume 22: P (k + n) ≠ {} for n using 20 using le-add1 by blast
    define m where m n S ≡ \{ p ∈ ∪ (transition A (w !! n) · S). even (the (g (Suc n) p))\} for n S
    define R where R i n S ≡ rec-nat S (λ i. m (n + i)) i for i n S
    have R-0[simp]: R 0 n = id for n unfolding R-def by auto
    have R-Suc[simp]: R (Suc i) n = m (n + i) o R i n for i n unfolding
    R-def by auto
    have R-Suc': R (Suc i) n = R i (Suc n) o m n for i n unfolding R-Suc
    by (induct i) (auto)
    have R-reach: R i n S ⊆ reach A w (n + i) if S ⊆ reach A w n for i n S
      using snth-in assms(2) that m-def by (induct i) (auto)
    have P-R: P (k + i) = R i k (P k) for i
      using 22 by (induct i) (auto simp add: case-prod-beta m-def st-succ-def)
    have 50: R i n S = (\{ p ∈ S. R i n (p)\}) for i n S
      by (induct i) (auto simp add: m-def prod.case-eq-if)
    have 51: R (i + j) n S = {} if R i n S = {} for i j n S
      using that by (induct j) (auto simp add: m-def prod.case-eq-if)
    have 52: R j n S = {} if i ≤ j R i n S = {} for i j n S
      using 51 by (metis le-add-diff-inverse that(1) that(2))
    have 1: ∃ p ∈ S. ∀ i. R i n {p} ≠ {} if assms: finite S \ i. R i n S ≠ {} for n S
  proof (rule ccontr)
assume 1: ¬ (∃ p ∈ S. ∀ i. R i n {p} ≠ {}) 
obtain f where 3: ∃ p. p ∈ S ⊃ R (f p) n {p} = {} using 1 by metis 
  have 4: R (Sup (f' S)) n {p} = {} if p ∈ S for p 
  proof (rule 52) 
  show f p ≤ Sup (f' S) using assms(1) that by (auto intro: le-cSup-finite) 
    show R (f p) n {p} = {} using 3 that by this 
    qed 
  have R (Sup (f' S)) n S = (∪ p ∈ S. R (Sup (f' S)) n {p}) using 50 
  by this 
  also have ... = {} using 4 by simp 
  finally have 5: R (Sup (f' S)) n S = {} by this 
  show False using that(2) 5 by auto 
  qed 
  have 2: ∃ i. R i (k + 0) (P k) ≠ {} using 22 P-R by simp 
  obtain p where 3: p ∈ P k ∃ i. R i k {p} ≠ {} using 1[OF P-finite 2] 
  by auto 
  define Q where Q n p ≡ (∃ i. R i (k + n) {p} ≠ {}) ∧ p ∈ P (k + n) 
  for n p 
  have 5: ∃ q ∈ transition A (w !! (k + n)) p. Q (Suc n) q if Q n p for n p 
  proof 
  have 11: p ∈ P (k + n) ∃ i. R i (k + n) {p} ≠ {} using that unfolding 
  Q-def by auto 
  have 12: R (Suc i) (k + n) {p} ≠ {} for i using 11(2) by this 
  have 13: R i (k + Suc n) (m (k + n) {p}) ≠ {} for i using 12 unfolding 
  R-Suc' by simp 
  have {p} ⊆ P (k + n) using 11(1) by auto 
  also have ... ⊆ reach A w (k + n) using P-reach by this 
  finally have R 1 (k + n) {p} ⊆ reach A w (k + n + 1) using R-reach 
  by blast 
  also have ... ⊆ 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 ∈ m (k + n) {p} ∃ i. R i (k + Suc n) {q} ≠ {} 
  using 1[OF 14 13] by auto 
  show ?thesis unfolding Q-def prod.case 
  proof (intro bexI conjI allI) 
  show ∃ i. R i (k + Suc n) {q} ≠ {} using 14(2) by this 
  show q ∈ P (k + Suc n) 
  using 14(1) 11(1) 22 unfolding m-def by (auto simp add: st-succ-def) 
  show q ∈ transition A (w !! (k + n)) p using 14(1) unfolding m-def 
  by simp 
  qed 
  qed 
  obtain r where 23: 
  run A r p ∃ i. Q i ((p # t trace r p) !! i) ∃ i. fst (r !! i) = w !! (k + i) 
  proof (rule nba.invariant-run-index[of Q 0 p A ∧ n p a. fst a = w !! (k + n)])
show Q 0 p unfolding Q-def using 3 by auto
show ∃ a. (fst a ∈ alphabet A ∧ snd a ∈ transition A (fst a) p) ∧
Q (Suc n) (snd a) ∧ fst a = w !! (k + n) if Q n p for n p
using snth-in assms(2) 5 that by fastforce
qed auto
have 20: map fst r = stdrop k w using 23(3) by (intro eqI-snth) (simp add: case-prod-beta)
  have 21: (p ≠# map snd r) !! i ∈ P (k + i) for i
  using 23(2) unfolding Q-def unfolding nba.trace-alt-def by simp
  obtain r where 23: run A (stdrop k w || stl r) (shd r) ∧ i. r !! i ∈ P (k + i)
  using 20 21 23(1) by (metis stream.sel(1) stream.sel(2) stream-map)
  let ?v = (k, shd r)
  let ?r = fromN (Suc k) || stl r
  have shd r = r !! 0 by simp
  also have ... ∈ P k using 23(2)[of 0] by simp
  also have ... ⊆ reach A w k using P-reach by this
  finally have 24: ?v ∈ gunodes A w using reach-gunodes by blast
  have 25: graph A w ?r ?v using run-grun 23(1) by this
  obtain l where 26: Ball (map snd (map gtrace (stdrop l ?r) (gtarget (stake l ?r) ?e))) odd
  using ranking-stuck-odd 0 24 25 by this
  have 27: f (Suc (k + l), r !! Suc l) =
    shd (map f (map gtrace (map stl l ?r) (gtarget (stake l ?r) ?e))) by (simp add: algebra-simps)
    also have ... ∈ sset (map f (map gtrace (map stdrop l ?r) (gtarget (stake l ?r) ?e)))
      using shd-sset by this
    finally have 28: odd (f (Suc (k + l), r !! Suc l)) using 26 by auto
    have r !! Suc l ∈ P (Suc (k + l)) using 23(2) by (metis add-Suc-right)
    also have ... ∈ {p ∈ P (Suc (k + l))} unfolding 23(2) by (auto simp: st-succ-def)
    also have ... ⊆ {p. even (the (g (Suc (k + l)) p))} by auto
    finally have 29: even (the (g (Suc (k + l)) (r !! Suc l))) by auto
    have 30: r !! Suc l ∈ reach A w (Suc (k + l))
      using 23(2) P-reach by (metis add-Suc-right subsetCE)
    have 31: even (f (Suc (k + l), r !! Suc l)) using 29 30 unfolding g-def
      by simp
    show False using 28 31 by simp
  qed
have 11: (λ k. P k = []) nats using 10 unfolding infs-snth by simp
have infs (λ S. S = []) (map snd (map g nats || map P nats))
  using 11 by (simp add: comp-def)
then have infs (λ x. snd x = []) (map g nats || map P nats)
  by (simp add: comp-def del: stream-map)
then have infs (λ (f, P). P = []) (map g nats || map P nats)
  by (simp add: case-prod-beta)
then have infs (λ (f, P). P = []) (stl (map g nats || map P nats)) by blast
then have $\inf \ (\lambda (f, P). \ P = \{\}) \ (\map s m a p (w \ ||| \ s t l \ (\map s m a p g \ n a t s \ ||| \ \map s m a p P \ n a t s)))$ by simp
then have $\inf \ (\lambda (f, P). \ P = \{\}) \ (s t l \ s)$ unfolding $s$-def by simp
then show $?thesis$ unfolding complement-def by auto
qed
qed
qed

4.4 Correctness Theorem

**Theorem** complement-language:
**Assumes** finite (nodes $A$)
**Shows** language (complement $A$) = streams (alphabet $A$) − language $A$
**Proof** (safe del: notI)
have 1: alphabet (complement $A$) = alphabet $A$ unfolding complement-def
nba.sel by rule
show $w \in$ streams (alphabet $A$) if $w \in$ language (complement $A$) for $w$
using nba.language-alphabet that 1 by force
show $w \notin$ language $A$ if $w \in$ language (complement $A$) for $w$
using complement-ranking ranking-language that by metis
show $w \in$ language (complement $A$) if $w \in$ streams (alphabet $A$) $w \notin$ language $A$ for $w$
using language-ranking ranking-complement assms that by blast
qed
end

5 Complementation Implementation

**Theory** Complementation-Implement
**Imports**
  Transition-Systems-and-Automata.NBA-Implement
  Complementation
**Begin**

unbundle lattice-syntax

type-synonym item = nat × bool
type-synonym 'state items = 'state → item

type-synonym state = (nat × item) list
abbreviation item-rel ≡ nat-rel × bool-rel
abbreviation state-rel ≡ (nat-rel, item-rel) list-map-rel

abbreviation pred $A$ $a$ $q$ ≡ \{ $p$, $q$ ∈ transition $A$ $a$ $p$\}

5.1 Phase 1

definition cs-lr :: 'state items ⇒ 'state lr where
\[ cs-lr f \equiv \text{map-option} \circ \text{fst} \]

**Definition**

\[ cs-st f \equiv f - '\text{Some } \text{snd} - ' \begin{Cases} \text{True} \end{Cases} \]

**Abbreviation**

\[ cs-abs f \equiv (cs-lr f, cs-st f) \]

**Definition**

\[ cs-rep \equiv \lambda (g, P). \text{map-option} \lambda (k, (k, p \in P)) (g p) \]

**Lemma**

\[ cs-abs-rep \begin{Cases} \text{simp} \end{Cases} : cs-rep (cs-abs f) = f \]

**Proof**

\[ \text{show } cs-rep (cs-abs f) x = f x \text{ for } x \]

\[ \text{unfolding } cs-lr-def \text{ cs-st-def cs-rep-def by (cases } f x \text{) (force+)} \]

**Qed**

**Lemma**

\[ cs-rep-lr \begin{Cases} \text{simp} \end{Cases} : cs-lr (cs-rep (g, P)) = g \]

**Proof**

\[ \text{show } cs-lr (cs-rep (g, P)) x = g x \text{ for } x \]

\[ \text{unfolding } cs-rep-def \text{ cs-lr-def by (cases } g x \text{) (auto)} \]

**Qed**

**Lemma**

\[ cs-rep-st \begin{Cases} \text{simp} \end{Cases} : cs-st (cs-rep (g, P)) = P \cap \text{dom } g \]

**Proof**

\[ \text{unfolding } cs-rep-def \text{ cs-st-def by force} \]

**Lemma**

\[ cs-lr-dom \begin{Cases} \text{simp} \end{Cases} : \text{dom } (cs-lr f) = \text{dom } f \text{ unfolding } cs-lr-def \text{ by simp} \]

**Lemma**

\[ cs-lr-apply \begin{Cases} \text{simp} \end{Cases} : \text{assumes } p \in \text{dom } f \]

\[ \text{shows } \text{the } (cs-lr f p) = \text{fst } (\text{the } (f p)) \]

**Using**

\[ \text{assms unfolding } cs-lr-def \text{ by auto} \]

**Lemma**

\[ cs-rep-dom \begin{Cases} \text{simp} \end{Cases} : \text{dom } (cs-rep (g, P)) = \text{dom } g \text{ unfolding } cs-rep-def \text{ by auto} \]

**Lemma**

\[ cs-rep-apply \begin{Cases} \text{simp} \end{Cases} : \text{assumes } p \in \text{dom } f \]

\[ \text{shows } \text{fst } (\text{the } (cs-rep (f, P) p)) = \text{the } (f p) \]

**Using**

\[ \text{assms unfolding } cs-rep-def \text{ by auto} \]

**Abbreviation**

\[ cs-rel \equiv (\text{state items} \times \text{state } cs) \text{ set where} \]

\[ cs-rel \equiv \text{br } cs-abs \text{ top} \]

**Lemma**

\[ cs-rel-inv-single-valued \text{: single-valued } (cs-rel^{-1}) \]

**Using**

\[ \text{(auto intro: inj-onI) (metis cs-abs-rep)} \]

**Definition**

\[ refresh-1 \equiv '\text{state items} \Rightarrow '\text{state items where} \]

\[ \text{refresh-1 } f \equiv \text{if } \text{True } \in \text{snd } \text{ran } f \text{ then } f \text{ else map-option } (\text{apsnd } \text{top}) \circ f \]

**Definition**

\[ ranks-1 \equiv \text{('label, 'state) nba } \Rightarrow '\text{label} \Rightarrow '\text{state items} \Rightarrow '\text{state items set where} \]

\[ \text{ranks-1 } A a f \equiv \{ g. \text{dom } g = \bigcup (\text{transition } A a \ ' (\text{dom } f)) \land \text{(}\forall p \in \text{dom } f. \forall q \in \text{transition } A a p. \text{fst } (\text{the } (g q)) \leq \text{fst } (\text{the } (f p))) \land \text{(}\forall q \in \text{dom } g. \text{accepting } A q \rightarrow \text{even } (\text{fst } (\text{the } (g q)))) \land \text{cs-st } g = \{ q \in \bigcup (\text{transition } A a \ ' (\text{cs-st } f)). \text{even } (\text{fst } (\text{the } (g q)))\} \} \]
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) nba \Rightarrow (label, state items) nba where
complement-1 A \equiv nba
(alphabet A)
(const (Some (2 * card (nodes A), False)) | initial A))
(complement-succ-1 A)
lambda f. cs-st f = {}

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

lemma complement-succ-1-abs:
assumes g \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) \setminus (dom f))
  \forall p \in dom f. \forall q \in transition A a p. fst (the (g q)) \leq fst (the (f p))
  \forall p \in dom g. accepting A p \rightarrow\ even (fst (the (g p)))
  using assms unfolding complement-succ-1-def ranks-1-def by simp-all
  show cs-lr g \in lr-succ A a (cs-lr f)
  unfolding lr-succ-def
proof (intro CollectI conjI ballI impI)
  show dom (cs-lr g) = \bigcup ((transition A a) \setminus (dom (cs-lr f))) using 1
  by simp
next
  fix p q
  assume 2: p \in dom (cs-lr f) q \in transition A a p
  have 3: q \in dom (cs-lr g) using 1 2 by auto
  show the (cs-lr g q) \leq the (cs-lr f p) using 1 2 3 by simp
next
  fix p
  assume 2: p \in dom (cs-lr g) accepting A p
  show even (the (cs-lr g p)) using 1 2 by auto
  qed
  have 2: cs-st g = \{ q \in \bigcup ((transition A a) \setminus cs-st (refresh-1 f)). even (fst (the (g q)))\}
  using assms unfolding complement-succ-1-def ranks-1-def by simp
  show cs-st g = st-succ A a (cs-lr g) (cs-st f)
  proof (cases cs-st f = {})
    case True
    have 3: the (cs-lr g q) = fst (the (g q)) if q \in \bigcup ((transition A a) \setminus (dom f))
for q
  using that 1(1) by simp
  show ?thesis using 2 3 unfolding st-succ-def refresh-1-cs-st True cs-lr-dom
  1(1) by force
  next
    case False
    have 3: \( q \in \bigcup ((\text{transition } A a) \cdot (\text{cs-st } f)) \)
    for q
      using that 1(1) by
      (auto intro!; cs-lr-apply)
      (metis IntE UN-iff cs-abs-rep cs-lr-dom cs-rep-st domD prod.collapse)
      have cs-st g = \( \{ q \in \bigcup (\text{transition } A a \cdot \text{cs-st } (\text{refresh-1 } f)), \text{ even } (\text{fst } (\text{the } (g q))) \} \)
      using 2 by
      also have \( \text{cs-st } (\text{refresh-1 } f) = \text{cs-st } f \) using False by simp
      also have \( \{ q \in \bigcup ((\text{transition } A a) \cdot (\text{cs-st } f)), \text{ even } (\text{fst } (\text{the } (g q))) \} = \)
      \( \{ q \in \bigcup ((\text{transition } A a) \cdot (\text{cs-st } f)), \text{ even } (\text{the } (\text{cs-lr } g q)) \} \) using 3 by
      metis
      also have \( \ldots = \text{st-succ } A a \cdot \text{(cs-lr } g \cdot \text{(cs-st } f) \) unfolding st-succ-def using
      False by simp
    finally show ?thesis by this
    qed
  qed

lemma complement-succ-1-rep:
  assumes \( P \subseteq \text{dom } f (g, Q) \in \text{complement-succ } A \cdot (f, P) \)
  shows \( \text{cs-rep } (g, Q) \in \text{complement-succ-1 } A \cdot (\text{cs-rep } (f, P)) \)
unfolding complement-succ-1-def ranks-1-def comp-apply
proof (intro Collect1 conjI ballI impI)
  have 1:
    \( \text{dom } g = \bigcup ((\text{transition } A a) \cdot (\text{dom } f)) \)
    \( \forall p \in \text{dom } f. \forall q \in \text{transition } A a \cdot p. \text{ the } (g q) \leq \text{ the } (f p) \)
    \( \forall p \in \text{dom } g. \text{ accepting } A a \cdot p \rightarrow \text{ even } (\text{the } (g p)) \)
    using assms(2) unfolding complement-succ-def cs-lr-def by simp-all
  have 2: \( Q = \{ q \in \text{if } P = \{ \} \text{ then dom } g \text{ else } \bigcup ((\text{transition } A a) \cdot P), \text{ even } \)
  \( (\text{the } (g q)) \)
    using assms(2) unfolding complement-succ-def st-succ-def by simp
  have 3: \( Q \subseteq \text{dom } g \text{ unfolding } 2 1(1) \) using assms(1) by auto
  show \( \text{dom } (\text{cs-rep } (g, Q)) = \bigcup (\text{transition } A a \cdot \text{dom } (\text{refresh-1 } (\text{cs-rep } (f, P)))) \) using 1 by simp
    show \( \bigwedge p q, p \in \text{dom } (\text{refresh-1 } (\text{cs-rep } (f, P))) \implies q \in \text{transition } A a \cdot p \implies \)
    \( \text{fst } (\text{the } (\text{cs-rep } (g, Q))) \leq \text{fst } (\text{the } (\text{refresh-1 } (\text{cs-rep } (f, P))) p) \)
    using 1(1, 2) by (auto (metis UN-I cs-rep-apply domI option.sel)
    show \( \bigwedge p q, p \in \text{dom } (\text{cs-rep } (g, Q)) \implies \text{accepting } A a \implies \text{ even } (\text{fst } (\text{the } (\text{cs-rep } (g, Q)) p)) \)
      using 1(1, 3) by auto
    show \( \text{cs-st } (\text{cs-rep } (g, Q)) = \{ q \in \bigcup (\text{transition } A a \cdot \text{cs-st } (\text{refresh-1 } (\text{cs-rep } (f, P)))) \} \)
      even (\( \text{fst } (\text{the } (\text{cs-rep } (g, Q))) \)))
  proof (cases \( P = \{ \} \))
case True
have cs-st (cs-rep (g, Q)) = Q using 3 by auto
also have ... = {q ∈ dom g. even (the (g q))} unfolding 2 using True by auto
also have ... = {q ∈ dom g. even (the (cs-rep (g, Q) q))} using cs-rep-apply by metis
next
case False
have 4: fst (the (cs-rep (g, Q) q)) = the (g q) if q ∈ ∪{(transition A a) · (dom f)} unfolding 2 using False by auto
also have ... = {q ∈ ∪{(transition A a) · P}, even (the (g q))} using 4 by force
also have P = (cs-st (refresh-1 (cs-rep (f, P)))) using assms(1) False by auto
finally show ?thesis by simp
qed
qed

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

36
show cs-abs g ∈ complement-succ A a (cs-abs f) using complement-succ-1-abs
1 by this
qed

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

5.2 Phase 2

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

lemma ranks-2-refine: ranks-2 = ranks-1
proof (intro ext)
fix A :: (′a, ′b) nba and a f
show ranks-2 A a f = ranks-1 A a f
proof safe
fix g
assume 1: \( g \in \text{ranks-2} \ A \ a \ f \)

have 2: \( \text{dom} \ g = \bigcup \{(\text{transition} \ A \ a) \cdot (\text{dom} \ f)\} \) using 1 unfolding ranks-2-def by auto

have 3: \( g \ q = \text{Some} \ (l, d) \implies l \leq \bigcap (\text{fst} \ ' \text{Some} - ^{'} \ f ^{'} \ \text{pred} \ A \ a \ q) \) for \( q \ l \ d \)

using 1 unfolding ranks-2-def by auto

have 4: \( g \ q = \text{Some} \ (l, d) \implies l \equiv \bigcup (\text{snd} \ ' \text{Some} - ^{'} \ f ^{'} \ \text{pred} \ A \ a \ q) \)

even \( l \) for \( q \ l \ d \)

using 1 unfolding ranks-2-def by auto

have 5: \( g \ q = \text{Some} \ (l, d) \implies \text{accepting} \ A \ q \implies \text{even} \ l \)

for \( q \ l \ d \)

using 1 unfolding ranks-2-def by auto

show \( g \in \text{ranks-1} \ A \ a \ f \)

unfolding ranks-1-def

proof (intro CollectI conjI ballI impI)

show dom g = \( \bigcup \{(\text{transition} \ A \ a) \cdot (\text{dom} \ f)\} \) using 2 by this

next

fix \( p \ q \)

assume 10: \( p \in \text{dom} \ f \ q \in \text{transition} \ A \ a \ p \)

obtain \( k \ c \) where 11: \( f \ p = \text{Some} \ (k, c) \)

using 10(1) by auto

have 12: \( q \in \text{dom} \ g \)

using 10 2 by auto

obtain \( l \ d \) where 13: \( g \ q = \text{Some} \ (l, d) \)

using 12 by auto

have \( \text{fst} \ (\text{the} \ (g \ q)) = l \)

unfolding 13 by simp

also have \( \ldots \leq \bigcap (\text{fst} \ ' \text{Some} - ^{'} \ f ^{'} \ \text{pred} \ A \ a \ q) \)

using 3 13 by this

also have \( \ldots \leq k \)

proof (rule cInf-lower)

show \( k \in \text{fst} \ ' \text{Some} - ^{'} \ f ^{'} \ \text{pred} \ A \ a \ q \)

using 11 10(2) by force

show \( \text{bdd-below} \ (\text{fst} \ ' \text{Some} - ^{'} \ f ^{'} \ \text{pred} \ A \ a \ q) \)

by simp

qed

also have \( \ldots = \text{fst} \ (\text{the} \ (f \ p)) \)

unfolding 11 by simp

finally show \( \text{fst} \ (\text{the} \ (g \ q)) \leq \text{fst} \ (\text{the} \ (f \ p)) \)

by this

next

fix \( q \)

assume 10: \( q \in \text{dom} \ g \text{ accepting} \ A \ q \)

show even \( (\text{fst} \ (\text{the} \ (g \ q))) \)

using 10 5 by auto

next

show cs-st \( g \ = \{q \in \bigcup \{(\text{transition} \ A \ a) \cdot (\text{cs-st} \ f)\}. \) even \( (\text{fst} \ (\text{the} \ (g \ q)))\}

proof

show cs-st \( g \subseteq \{q \in \bigcup \{(\text{transition} \ A \ a) \cdot (\text{cs-st} \ f)\}. \) even \( (\text{fst} \ (\text{the} \ (g \ q)))\}

using 4 unfolding cs-st-def image-def vimage-def by auto metis

show \( \{q \in \bigcup \{(\text{transition} \ A \ a) \cdot (\text{cs-st} \ f)\}. \) even \( (\text{fst} \ (\text{the} \ (g \ q)))\} \subseteq \text{cs-st} \ g \)

proof safe

fix \( p \ q \)

assume 10: \( \text{even} \ (\text{fst} \ (\text{the} \ (g \ q))) \) \( p \in \text{cs-st} \ f \ q \in \text{transition} \ A \ a \ p \)

have 12: \( q \in \text{dom} \ g \)

using 10 2 unfolding cs-st-def by auto

show \( q \in \text{cs-st} \ g \)

using 10 4 12 unfolding cs-st-def image-def by force

qed

qed

next

fix \( g \)

38
assume 1: \( g \in \text{ranks-1} \ A \ a \ f \)

have 2: \( \text{dom} \ g = \bigcup \{(\text{transition} \ A \ a \ p) \ | \ p \in \text{dom} \ f \} \) using 1 unfolding ranks-1-def

by auto

have 3: \( \forall \ p \ q. \ p \in \text{dom} \ f \implies q \in \text{transition} \ A \ a \ p \implies \text{fst} \ (\text{the} \ (g \ q)) \leq \text{fst} \ (\text{the} \ (f \ p)) \)

using 1 unfolding ranks-1-def by auto

have 4: \( \forall \ q. \ q \in \text{dom} \ g \implies \text{accepting} \ A \ q \implies \text{even} \ (\text{fst} \ (\text{the} \ (g \ q))) \)

using 1 unfolding ranks-1-def by auto

have 5: \( \text{cs-st} \ g = \{q \in \bigcup \{(\text{transition} \ A \ a \ p) \ | \ p \in \text{dom} \ f \} \ | \ \text{even} \ (\text{fst} \ (\text{the} \ (g \ q))) \} \)

using 1 unfolding ranks-1-def by auto

show \( g \in \text{ranks-2} \ A \ a \ f \)

unfolding ranks-2-def

proof (intro CollectI conjI allI impI)

show \( \text{dom} \ g = \bigcup \{(\text{transition} \ A \ a \ p) \ | \ p \in \text{dom} \ f \} \) using 2 by this

next

fix \( q \ l \ d \)

assume 10: \( g \ q = \text{Some} \ (l, d) \)

have 11: \( q \in \text{dom} \ g \) using 10 by auto

show \( l \leq \bigcap \{(\text{fst} \ (\text{Some} \ - \ f \ \text{pred} \ A \ a \ q)) \) using 3 10 by (auto (metis domI fst-conv option.sel))

qed

show \( d \longleftrightarrow q \in \text{cs-st} \ g \) unfolding cs-st-def by (force simp: 10)

also have \( \text{cs-st} \ g = \{q \in \bigcup \{(\text{transition} \ A \ a \ p) \ | \ p \in \text{dom} \ f \} \ | \ \text{even} \ (\text{fst} \ (\text{the} \ (g \ q))) \} \)

using 5 by this

also have \( q \in \ldots \longleftrightarrow (\exists \ x. \ x \in \text{cs-st} \ f. \ q \in \text{transition} \ A \ a \ x) \wedge \text{even} \ l \)

unfolding mem-Collect-eq 10 by simp

also have \( \ldots \longleftrightarrow \bigcap \{(\text{snd} \ (\text{Some} \ - \ f \ \text{pred} \ A \ a \ q)) \wedge \text{even} \ l \}

unfolding cs-st-def image-def vimage-def by auto metis

finally show \( d \longleftrightarrow \bigcap \{(\text{snd} \ (\text{Some} \ - \ f \ \text{pred} \ A \ a \ q)) \wedge \text{even} \ l \) by this

show \( \text{accepting} \ A \ q \implies \text{even} \ l \) using 4 10 11 by force

qed

qed

qed

lemma complement-2-refine: \( (\text{complement-2}, \ \text{complement-1}) \in (\text{Id}, \ \text{Id}) \ \text{nba-rel} \)

\( \rightarrow (\text{Id}, \ \text{Id}) \ \text{nba-rel} \)

unfolding complement-2-def complement-1-def complement-succ-2-def complement-succ-1-def

unfolding ranks-2-refine cs-st-def image-def vimage-def run-def by auto

5.3 Phase 3

definition bounds-3 :: ('label, 'state) nba \( \Rightarrow \) 'label \( \Rightarrow \) 'state items \( \Rightarrow \) 'state items

where

\( \text{bounds-3} \ A \ a \ f \equiv \lambda \ q. \ \text{let} \ S = \text{Some} \ - \ f \ \text{pred} \ A \ a \ q \ \text{in} \)
if $S = \{\} \text{ then None else Some } \bigcup \{\text{map-option } (\text{items-3 A p}) \ (f \ p)\}

definition items-3 :: ('label, 'state) nba ⇒ 'state ⇒ item ⇒ item set where
items-3 A p ≡ λ (k, c). \{(l, c ∧ even l) \mid l, l ≤ k ∧ \text{ (accepting } A p \rightarrow \text{ even } l)\}
definition get-3 :: ('label, 'state) nba ⇒ 'state items ⇒ ('state → item set) where
get-3 A f ≡ λ p. \text{ map-option } (\text{items-3 A p}) \ (f \ p)
definition complement-succ-3 ::
('label, 'state) nba ⇒ 'label ⇒ 'state items set where
complement-succ-3 A a ≡ expand-map ◦ get-3 A ◦ bounds-3 A a ◦ refresh-1
definition complement-3 :: ('label, 'state) nba ⇒ ('label, 'state items) nba where
complement-3 A ≡ nba
(\text{ alphabet } A)
\{(\text{Some } \circ (\text{const } (2 * \text{ card } (\text{nodes } A), \text{False})) \mid \text{ initial } A\}
(\text{ complement-succ-3 } A)
(λ f. ∀ (p, k, c) ∈ \text{ map-to-set } f. ¬ c)

lemma bounds-3-dom[simp]: dom (\text{bounds-3 } A a f) = \bigcup (\text{transition } A a \ ' (\text{dom } f))
unfolding bounds-3-def Let-def dom-def by (force split: if-splits)
lemma items-3-nonempty[introl, simp]: items-3 A p s ≠ \{\} unfolding items-3-def by auto
lemma items-3-finite[introl, simp]: finite (items-3 A p s)
unfolding items-3-def by (auto split: prod.splits)
lemma get-3-dom[simp]: dom (get-3 A f) = dom f unfolding get-3-def by (auto split: bind-splits)
lemma get-3-finite[intro, simp]: S ∈ ran (get-3 A f) ⇒ finite S
unfolding get-3-def ran-def by auto
lemma get-3-update[simp]: get-3 A (f (p ↦ s)) = (get-3 A f) (p ↦ items-3 A p s)
unfolding get-3-def by auto
lemma expand-map-get-bounds-3: \text{ expand-map } ◦ \text{get-3 } A ◦ \text{bounds-3 } A a = \text{ runs-2 } A a
proof (intro ext set-eqI, unfold comp-apply)
fix f g
have 1: (\forall x S y. \text{get-3 } A (\text{bounds-3 } A a f) x = \text{Some } S \rightarrow g x = \text{Some } y \rightarrow
y ∈ S) \iff
(\forall q S l d. \text{get-3 } A (\text{bounds-3 } A a f) q = \text{Some } S \rightarrow g q = \text{Some } (l, d) \rightarrow
(l, d) ∈ S) \iff
by auto
have 2: (\forall S. \text{get-3 } A (\text{bounds-3 } A a f) q = \text{Some } S \rightarrow g q = \text{Some } (l, d) \rightarrow
(l, d) ∈ S) \iff
\ (g q = \text{Some } (l, d) \rightarrow l ≤ \bigcap (\text{map-option } (\text{items-3 } A a f) \ (f \ p) \ (\text{pred } A a q)) \land
(d \iff \bigcup (\text{map-option } (\text{items-3 } A a f) \ (f \ p) \ (\text{pred } A a q)) \land \text{even } l) \land (\text{accepting } A q \rightarrow \text{even } l))
if 3: dom g = \bigcup (\text{transition } A a \ ' (\text{dom } f)) \text{ for } q l d
proof –
have \( q \notin \text{dom } g \) if \( \text{Some} - f \vdash \text{pred } A a q = \{\} \) unfolding 3 using that

by force

show \( \text{thesis} \) unfolding get-3-def items-3-def bounds-3-def Let-def using 4

by auto

qed

show \( g \in \text{expand-map } (\text{get-3 } A (\text{bounds-3 } A a f)) \leftrightarrow g \in \text{ranks-2 } A a f \)

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: \( \text{complement-succ-3 } = \text{complement-succ-2} \)

unfolding complement-succ-3-def complement-succ-2-def expand-map-get-bounds-3

by rule

lemma complement-initial-3-refine: \( \{\text{const } (\text{Some } (2 \ast \text{card } (\text{nodes } A), \text{False}))\} \mid \text{initial } A \} = \{(\text{Some } \circ (\text{const } (2 \ast \text{card } (\text{nodes } A), \text{False}))) \mid \text{initial } A \}

unfolding comp-apply by rule

lemma complement-accepting-3-refine: \( \text{True } \notin \text{snd } ' \text{ran } f \leftrightarrow (\forall (p, k, c) \in \text{map-to-set } f. \neg c) \)

unfolding map-to-set-def ran-def by auto

lemma complement-3-refine: \( (\text{complement-3}, \text{complement-2}) \in \langle \text{Id}, \text{Id} \rangle \text{nba-rel} \)

\( \to \langle \text{Id}, \text{Id} \rangle \text{nba-rel} \)

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 = \( \lambda (k, c). \{(l, c \land \text{even } l) \mid l. k \leq \text{Suc } l \land l \leq k \land (\text{accepting } A p \to \text{even } l)\}\)

definition get-4 :: ("label", "state") nba \( \Rightarrow \) "state items \( \Rightarrow \) ("state \to \) item set)

where

get-4 A f = \( \lambda p. \text{map-option } (\text{items-4 } A p) (f p) \)

definition complement-succ-4 ::

(\"label", \"state\") nba \( \Rightarrow \) \"label \Rightarrow \) \"state items \( \Rightarrow \) \"state items set where

complement-succ-4 A a = \( \equiv \text{expand-map } \circ \text{get-4 } A \circ \text{bounds-3 } A a \circ \text{refresh-1} \)

definition complement-4 :: (\"label", \"state\") nba \( \Rightarrow \) (\"label", \"state items\") nba where

complement-4 A = \( \equiv \text{nba\)}

(alphabet A)

\( \{(\text{Some } \circ (\text{const } (2 \ast \text{card } (\text{nodes } A), \text{False}))) \mid \text{initial } A \}\}

(complement-succ-4 A)

(\(\lambda f. \forall (p, k, c) \in \text{map-to-set } f. \neg c\))

lemma get-4-dom[simp]: dom (get-4 A f) = dom f unfolding get-4-def by (auto split: bind-splits)

definition R :: \"state items \to \) rel where
\[ R \equiv \{(f, g) \} . \]
\[ \text{dom } f = \text{dom } g \land \]
\[ (\forall p \in \text{dom } f. \ \text{fst } (\text{the } (f p)) \leq \text{fst } (\text{the } (g p))) \land \]
\[ (\forall p \in \text{dom } f. \ \text{snd } (\text{the } (f p)) \leftrightarrow \text{snd } (\text{the } (g p))) \}

**Lemma bounds-R:**
\[ \text{assumes } (f, g) \in R \]
\[ \text{assumes } \text{bounds-3 } A \ a \ (\text{refresh-1 } f) \ p = \text{Some } (n, e) \]
\[ \text{assumes } \text{bounds-3 } A \ a \ (\text{refresh-1 } g) \ p = \text{Some } (k, c) \]
\[ \text{shows } n \leq k \ e \longleftrightarrow c \]

**Proof**

- **have 1:**
  \[ \text{dom } f = \text{dom } g \]
  \[ \forall p \in \text{dom } f. \ \text{fst } (\text{the } (f p)) \leq \text{fst } (\text{the } (g p)) \]
  \[ \forall p \in \text{dom } f. \ \text{snd } (\text{the } (f p)) \leftrightarrow \text{snd } (\text{the } (g p)) \]
  \[ \text{using } \text{assms}(1) \ \text{unfolding } \text{R-def} \ \text{by auto} \]
- **have 2:**
  \[ n = \bigcap \{ \text{fst } \ (\text{Some } -' \ \text{refresh-1 } f \ ' \ \text{pred } A \ a \ p) \} \]
  \[ \text{using } \text{assms}(2) \ \text{unfolding } \text{bounds-3-def} \ \text{by } (\text{auto simp: Let-def split: if-splits}) \]
- **also have**
  \[ \text{fst } \ (\text{Some } -' \ \text{refresh-1 } f \ ' \ \text{pred } A \ a \ p = \text{fst } \ (\text{Some } -' \ f ' \ \text{pred } A \ a \ p) \]

**Proof**

- **show**
  \[ \text{fst } \ (\text{Some } -' \ \text{refresh-1 } f ' \ \text{pred } A \ a \ p \subseteq \text{fst } \ (\text{Some } -' \ f ' \ \text{pred } A \ a \ p) \]
  \[ \text{unfolding } \text{refresh-1-def image-def} \]
  \[ \text{by } (\text{auto simp: map-option-case split: option.split}) \ (\text{force}) \]
- **show**
  \[ \text{fst } \ (\text{Some } -' \ f ' \ \text{pred } A \ a \ p \subseteq \text{fst } \ (\text{Some } -' \ \text{refresh-1 } f ' \ \text{pred } A \ a \ p) \]
  \[ \text{unfolding } \text{refresh-1-def image-def} \]
  \[ \text{by } (\text{auto simp: map-option-case split: option.split}) \ (\text{metis fst-conv option.sel}) \]

**Qed**

- **also have**
  \[ \ldots \ \text{fst } \ (\text{Some } -' \ f ' \ \text{pred } A \ a \ p \cap \text{dom } f) \]
  \[ \text{unfolding } \text{dome-def image-def Int-def} \ \text{by auto metis} \]
- **also have**
  \[ \ldots \ \text{fst } \ (\text{Some } -' \ f ' \ \text{pred } A \ a \ p \cap \text{dom } f) \]
  \[ \text{unfolding } \text{dome-def by force} \]
- **also have**
  \[ \ldots = (\text{fst } \ (\text{the } \ o) \ ' \ (\text{pred } A \ a \ p \cap \text{dom } f)) \ \text{by force} \]
- **also have**
  \[ \ldots = \bigcap \{ (\text{fst } \ (\text{the } \ o) \ ' \ (\text{pred } A \ a \ p \cap \text{dom } f)) \} \ \text{by force} \]
  \[ \bigcap \{ (\text{fst } \ (\text{the } \ o) \ ' \ (\text{pred } A \ a \ p \cap \text{dom } g)) \} \]
  \[ \text{proof } (\text{rule cINF-mono}) \]
- **show**
  \[ \text{pred } A \ a \ p \cap \text{dom } g \neq \{ \} \]
  \[ \text{using } \text{assms}(2) \ 1(1) \ \text{unfolding } \text{bounds-3-def refresh-1-def} \]
  \[ \text{by } (\text{auto simp: Let-def split: if-splits}) \ (\text{force+}) \]
  \[ \text{show} \ \exists \ n \in \text{pred } A \ a \ p \cap \text{dom } f. \ (\text{fst } \ (\text{the } \ o) \ ' \ (\text{pred } A \ a \ p \cap \text{dom } f)) \ \text{by rule} \]
- **show**
  \[ \text{if } m \in \text{pred } A \ a \ p \cap \text{dom } g \ \text{for } m \ \text{using } 1 \ \text{that by auto} \]

**Qed**

- **also have**
  \[ \ldots = \text{fst } \ (\text{Some } -' \ g ' \ \text{pred } A \ a \ p \cap \text{dom } g) \]
  \[ \text{unfolding } \text{dome-def by force} \]
- **also have**
  \[ \ldots = \text{fst } \ (\text{Some } -' \ g ' \ \text{pred } A \ a \ p) \]
  \[ \text{unfolding } \text{dome-def Int-def by auto metis} \]
- **also have**
  \[ \ldots = \text{fst } \ (\text{Some } -' \ \text{refresh-1 } g ' \ \text{pred } A \ a \ p) \]

42
proof

  show \( \text{fst} ' \text{Some} \cap \text{pred} A a p \subseteq \text{fst} ' \text{Some} \cap \text{pred} A a p \),

  unfolding refresh-1-def image-def
  by (auto simp: map-option-case split: option.split) (metis fst-conv option.sel)

  show \( \text{fst} ' \text{Some} \cap \text{refresh-1} g \cap \text{pred} A a p \subseteq \text{fst} ' \text{Some} \cap \text{refresh-1} g \cap \text{pred} A a p \),

  unfolding refresh-1-def image-def
  by (auto simp: map-option-case split: option.split) (force)

  qed

also have \( \bigcap (\text{fst} ' (\text{Some} \cap \text{refresh-1} g ' \cap \text{pred} A a p)) = k \),

using assms(3) unfolding bounds-3-def by (auto simp: Let-def list-conv)

finally show \( n \leq k \) by this

  have \( c \leftrightarrow \bigcup (\text{snd} ' (\text{Some} \cap \text{refresh-1} f ' \cap \text{pred} A a p)) \),

  unfolding refresh-1-def dom-def by auto

also have \( \text{snd} ' \text{Some} \cap \text{refresh-1} f ' \cap \text{pred} A a p = \text{snd} ' \text{Some} \cap \text{refresh-1} f ' \),

by force

also have \( \ldots = (\text{snd} \circ \text{the} ' \text{refresh-1} f ' \cap \text{pred} A a p \cap \text{dom} (\text{refresh-1} f)) \),

proof (rule image-cong)

  show \( \text{pred} A a p \cap \text{dom} (\text{refresh-1} f) = \text{pred} A a p \cap \text{dom} (\text{refresh-1} g) \),

  unfolding refresh-1-dom 1(1) by rule

  show \( (\text{snd} \circ \text{the} ' \text{refresh-1} f ' q \leftrightarrow (\text{snd} \circ \text{the} ' \text{refresh-1} g ' q) \),

  if \( \exists q \in \text{pred} A a p \cap \text{dom} (\text{refresh-1} g) \) for \( q \),

  proof

    have \( \exists k. \forall x \in \text{ran} f. ~ \neg \text{snd} x \implies (n, \text{True}) \in \text{ran} q \implies q = \text{Some} (k, c) \implies c \) for \( n k c \),

    using 1(1, 3) unfolding dom-def ran-def
    by (auto dest!: Collect-inj) (metis option.sel snd-conv)

    have \( \exists k. \forall x \in \text{ran} g. ~ \neg \text{snd} x \implies (k, \text{True}) \in \text{ran} f \) \( \implies \text{False} \) for \( k \),

    using 1(1, 3) unfolding dom-def ran-def
    by (auto dest!: Collect-inj) (metis option.sel snd-conv)

    show \( (\text{snd} \circ \text{the} ' \text{refresh-1} f ' q \leftrightarrow (\text{snd} \circ \text{the} ' \text{refresh-1} g ' q) \),

    using 1(1, 3) 2 3 unfolding refresh-1-def by (rule split: if-splits)

    show \( (\text{snd} \circ \text{the} ' \text{refresh-1} f ' q \leftrightarrow (\text{snd} \circ \text{the} ' \text{refresh-1} g ' q) \),

    using 1(1, 3) 2 4 5 unfolding refresh-1-def
    by (auto simp: map-option-case split: option.splits if-splits) (force+)

  qed

  qed

also have \( \ldots = (\text{snd} \circ \text{the} ' \text{refresh-1} g ' \cap \text{pred} A a p \cap \text{dom} (\text{refresh-1} g)) \) by force

also have \( \ldots = (\text{snd} ' \text{Some} \cap \text{refresh-1} g ' \cap \text{pred} A a p \cap \text{dom} (\text{refresh-1} g)) \),

unfolding dom-def by force

also have \( \ldots = (\text{snd} ' \text{Some} \cap \text{refresh-1} g ' \cap \text{pred} A a p \cap \text{dom} (\text{refresh-1} g)) \),

unfolding dom-def image-def Int-def by auto metis

43
also have 1 (snd (Some - refresh-1 g pred A a p)) \mapsto c
using assms(3) unfolding bounds-3-def by (auto simp: Let-def split: if-splits)
finally show e \mapsto c 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
    unfolding complement-3-def complement-4-def R-def by simp
proof
  have 1: f' \in expand-map (get-3 A (bounds-3 A a (refresh-1 f)))
    unfolding that(1) unfolding complement-3-def complement-4-def succ-3-def by auto
  have 2:
    dom f' = dom g
    \forall p \in dom f. fst (the (f p)) \leq fst (the (g p))
    \forall p \in dom f. snd (the (f p)) \mapsto snd (the (g p))
    using that(2) unfolding R-def by auto
    have dom f' = dom (get-3 A (bounds-3 A a (refresh-1 f))) using expand-map-dom 1 by this
  also have \ldots = dom (bounds-3 A a (refresh-1 f)) by simp
  finally have 3: dom f' = dom (bounds-3 A a (refresh-1 f)) by this
  define g' where g' p \equiv do
    \{ (k, c) \leftarrow bounds-3 A a (refresh-1 g) p;
        (l, d) \leftarrow f' p;
        Some (if even k = even l then k else k - 1, d) \}
  for p
  have 4: g' p = do
    \{ kc \leftarrow bounds-3 A a (refresh-1 g) p;
        ld \leftarrow f' p;
        Some (if even (fst kc) = even (fst ld) then fst kc else fst kc - 1, snd ld) \}
  for p unfolding g'-def case-prod-beta by rule
  have dom g' = dom (bounds-3 A a (refresh-1 g)) \cap dom f' using 4 bind-eq-Some-cone
by fastforce
  also have \ldots = dom f' using 2 3 by simp
  finally have 5: dom g' = dom f' by this
  have 6: (l, d) \in items-3 A p (k, c)
    if bounds-3 A a (refresh-1 f) p = Some (k, c) f' p = Some (l, d) for p k c l
  d
  using 1 that unfolding expand-map-alt-def get-3-def by blast
  show \??thesis
unfolding complement-4-def nba.sel complement-succ-4-def comp-apply
proof
  show \((f', g') \in R\)
  unfolding R-def mem-Collect-eq prod.case
  proof (intro conjI ballI)
    show dom \(f' = dom g'\) using 5 by rule
  next
  fix \(p\)
  assume 10: \(p \in dom f'\)
  have 11: \(p \in dom (\text{bounds-3} A a (\text{refresh-1} g))\)
    using 2(1) 3 10 by simp
  obtain \(k c\) where 12: \(\text{bounds-3} A a (\text{refresh-1} g) p = \text{Some} (k, c)\)
    using 11
  11 by fast
  obtain \(l d\) where 13: \(f' p = \text{Some} (l, d)\)
    using 10 by auto
  obtain \(n e\) where 14: \(\text{bounds-3} A a (\text{refresh-1} f) p = \text{Some} (n, e)\)
    using 10
  10 3 by fast
  have 15: \((l, d) \in \text{items-3} A p (n, e)\)
    using 6 14 13 by this
  have 16: \(n \leq k\) using bounds-R(1) that(2) 14 12 by this
  have 17: \(l \leq k\)
    unfolding items-3-def by simp
  have 18: even \(k\) \(\iff\) odd \(l\) \(\implies\) \(l \leq k\) \(\iff\) \(l \leq k - 1\)
    by presburger
  have 19: even \(c\)
    unfolding bounds-R(2) that(2) 14 12 by this
  show \(\text{fst} (\text{the} (f' p)) \leq \text{fst} (\text{the} (g' p))\)
    using 17 18 unfolding 4 12 13
  by simp
  show \(\text{snd} (\text{the} (f' p)) \leq \text{snd} (\text{the} (g' p))\)
    using 19 unfolding 4 12 13
  by simp
qed
show \(g' \in \text{expand-map} (\text{get-4} A (\text{bounds-3} A a (\text{refresh-1} g)))\)
unfolding expand-map-alt-def mem-Collect-eq
proof (intro conjI allI impI)
  show dom \(g' = dom (\text{get-4} A (\text{bounds-3} A a (\text{refresh-1} g)))\)
    using 2(1) 3
  5 by simp
  fix \(p S xy\)
  assume 10: \(\text{get-4} A (\text{bounds-3} A a (\text{refresh-1} g)) p = \text{Some} S\)
  assume 11: \(g' p = \text{Some} xy\)
  obtain \(k c\) where 12: \(\text{bounds-3} A a (\text{refresh-1} g) p = \text{Some} (k, c) S = \text{items-4} A p (k, c)\)
    using 10 unfolding get-4-def by auto
  obtain \(l d\) where 13: \(f' p = \text{Some} (l, d) xy = (\text{if even} k \iff \text{even} l\) then \(k\) else \(k - 1, d)\)
    using 11 12 unfolding g'-def by (auto split: bind-splits)
  obtain \(n e\) where 14: \(\text{bounds-3} A a (\text{refresh-1} f) p = \text{Some} (n, e)\)
    using 13(1) 3 by fast
  have 15: \((l, d) \in \text{items-3} A p (n, e)\)
    using 6 14 13(1) by this
  have 16: \(n \leq k\)
    using bounds-R(1) that(2) 14 12(1) by this
  have 17: \(e \iff c\)
    using bounds-R(2) that(2) 14 12(1) by this
  show \(xy \in S\)
    using 15 16 17 unfolding 12(2) 13(2) items-3-def items-4-def
  by auto
qed
qed
show $\bigwedge p \ p_1 (p, q) \in R \implies \text{accepting} (\text{complement-3 A}) p \implies \text{accepting} (\text{complement-4 A}) q$

  unfolding complement-3-def complement-4-def R-def map-to-set-def
  by (auto) (metis domIff eq-snd-iff option.exhaust-sel option.sel)
qed

lemma complement-4-less: complement-4 A $\leq$ complement-3 A
unfolding less-eq-nba-def
unfolding complement-4-def complement-3-def nba.sel
unfolding complement-succ-4-def complement-succ-3-def
proof (safe intro: le-funI, unfold comp-apply)
  fix $a \ f \ g$
  assume $g \in \text{expand-map} (\text{get-4 A} (\text{bounds-3 A} a (\text{refresh-1 f})))$
  then show $g \in \text{expand-map} (\text{get-3 A} (\text{bounds-3 A} a (\text{refresh-1 f})))$
  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: $\text{language} (\text{complement-4 A}) \subseteq \text{language} (\text{complement-3 A})$
using language-mono complement-4-less by (auto dest: monoD)

lemma complement-4-language: $\text{language} (\text{complement-3 A}) = \text{language} (\text{complement-4 A})$
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 $(\text{nodes} (\text{complement-3 A}), \text{nodes} (\text{complement-2 A})) \in \langle \text{Id} \rangle \ set-rel$
    using complement-3-refine by parametricity auto
  also have $(\text{nodes} (\text{complement-2 A}), \text{nodes} (\text{complement-1 A})) \in \langle \text{Id} \rangle \ set-rel$
    using complement-2-refine by parametricity auto
  also have $(\text{nodes} (\text{complement-1 A}), \text{nodes} (\text{complement A})) \in \langle \text{cs-rel} \rangle \ set-rel$
    using complement-1-refine by parametricity auto
  finally have 1: $(\text{nodes} (\text{complement-3 A}), \text{nodes} (\text{complement A})) \in \langle \text{cs-rel} \rangle \ set-rel$
    by simp
  have 2: finite (nodes (complement A)) using complement-finite assms(1) by this
  have 3: finite (nodes (complement-3 A))
    using finite-set-rel-transfer-back 1 cs-rel-inv-single-valued 2 by this
  have 4: nodes (complement-4 A) $\subseteq$ nodes (complement-3 A)
    using nodes-mono complement-4-less by (auto dest: monoD)
  show finite (nodes (complement-4 A)) using finite-subset 4 3 by this
qed

lemma complement-4-correct:
assumes finite (nodes A)
s 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 \((\text{language~(complement-3~A)}, \text{language~(complement-2~A)}) \in \langle\langle \text{Id} \rangle \rangle \text{stream-rel} \rangle \text{set-rel})\)
using complement-3-refine by parametricity auto
also have \((\text{language~(complement-2~A)}, \text{language~(complement-1~A)}) \in \langle\langle \text{Id} \rangle \rangle \text{stream-rel} \rangle \text{set-rel})\)
using complement-2-refine by parametricity auto
also have \((\text{language~(complement-1~A)}, \text{language~(complement~A)}) \in \langle\langle \text{Id} \rangle \rangle \text{stream-rel} \rangle \text{set-rel})\)
using complement-1-refine by parametricity auto
also have \((\text{language~(complement~A)}) = \text{streams~(alphabet~A)} - \text{language~A})\)
using complement-language assms (1) by this
finally show \((\text{language~(complement-4~A)} = \text{streams~(alphabet~A)} - \text{language~A})\)
by simp
qed

5.5 Phase 5
definition refresh-5 :: 'state items ⇒ 'state items nres where
  refresh-5 f ≡ if \(\exists (p, k, c) \in \text{map-to-set } f\). c
  then RETURN f
  else do
    \{ 
      \ASSUME (finite (dom f));
      FOREACH (map-to-set f) (λ (p, k, c) m. do
        \{ 
          \ASSERT (p ∉ dom m);
          RETURN (m (p↦→ (k, True)))
        }
      )
    Map.empty
  
definition merge-5 :: item ⇒ item option ⇒ item where
  merge-5 ≡ λ (k, c). λ None ⇒ (k, c) | Some (l, d) ⇒ (k ⊓ l, c ⊔ d)
definition bounds-5 :: ('label, 'state) nba ⇒ 'label ⇒ 'state items ⇒ 'state items nres where
  bounds-5 A a f ≡ do
    \{ 
      \ASSUME (finite (dom f));
      \ASSUME (∀ p. finite (transition A a p));
      FOREACH (map-to-set f) (λ (p, s) m.
        FOREACH (transition A a p) (λ q f.
          RETURN (f (q↦merge-5 s (f q))))
        )
      Map.empty
    
definition items-5 :: ('label, 'state) nba ⇒ 'state ⇒ item ⇒ item set where
  items-5 A p ≡ λ (k, c). do
    \{ 
      let values = if accepting A p then Set.filter even {k − 1 .. k} else {k − 1 .. k};
    
  qed
let item = λ l. (l, c ∧ even l);
item ' values
}
definition get-5 :: ('label, 'state) nba ⇒ 'state items ⇒ ('state → item set)
where
get-5 A f ≡ λ p. map-option (items-5 A p) (f p)
definition expand-5 :: ('a → 'b set) ⇒ ('a → 'b) set nres where
expand-5 f ≡ FOREACH (map-to-set f) (λ (x, S) X. do {
    ASSERT (∀ g ∈ X. x /∈ dom g);
    ASSERT (∀ a ∈ S. ∀ b ∈ S. a /≠ b → (λ y. (λ g. g (x → y)) ' X) a ∩ (λ y. (λ g. g (x → y)) ' X) b = {});
    RETURN (∪ y ∈ S. (λ g. g (x → y)) ' X)
}) {Map.empty}
definition complement-succ-5 :: ('label, 'state) nba ⇒ 'label ⇒ 'state items set nres where
complement-succ-5 A a f ≡ do
{ f ← refresh-5 f;
  f ← bounds-5 A a f;
  ASSERT (finite (dom f));
  expand-5 (get-5 A f)
}

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

48
next
  case False
  then have pred A a q ∩ {x. x ≠ p} = pred A a q
  by auto
  with False show ?thesis by (simp add: bounds-3-def)
qed

lemma refresh-5-refine: (refresh-5, f. RETURN (refresh-1 f)) ∈ Id → ⟨Id⟩ nres-rel
proof safe
  fix f :: 'a ⇒ item option
  have 1: (∃ (p, k, c) ∈ map-to-set f. c) ←→ True ∈ snd ′ ran f
  unfolding image-def map-to-set-def ran-def by force
  show (refresh-5 f, RETURN (refresh-1 f)) ∈ ⟨Id⟩ nres-rel
  unfolding refresh-5-def refresh-1-def 1
  by (refine-vcg FOREACH-rule-map-eq[where X = λ m. map-option (apsnd T) o m]) (auto)
qed

lemma bounds-5-refine: (bounds-5 A a, f. RETURN (bounds-3 A a f)) ∈ Id → ⟨Id⟩ nres-rel
  unfolding bounds-5-def by
  (refine-vcg FOREACH-rule-map-eq[where X = bounds-3 A a] FOREACH-rule-insert-eq)
  (auto simp: override-on-insert bounds-3-empty bounds-3-update)
lemma items-5-refine: items-5 = items-4
  unfolding items-5-def items-4-def by (intro ext) (auto split: if-splits)
lemma get-5-refine: get-5 = get-4
  unfolding get-5-def get-4-def items-5-refine by rule
lemma expand-5-refine: (expand-5 f, ASSERT (finite (dom f)) ⇒ RETURN
  (expand-map f)) ∈ ⟨Id⟩ nres-rel
  unfolding expand-5-def
  by (refine-vcg FOREACH-rule-map-eq[where X = expand-map]) (auto dest!: expand-map-dom map-upd-eqD1)

lemma complement-succ-5-refine: (complement-succ-5, RETURN ‚œœœ complement-succ-4) ∈
  Id → Id → Id → ⟨Id⟩ nres-rel
  unfolding complement-succ-5-def complement-succ-4-def get-5-refine comp-apply
  by (refine-vcg vcg1[OF refresh-5-refine] vcg1[OF bounds-5-refine] vcg0[OF expand-5-refine]) (auto)

5.6 Phase 6

definition expand-map-get-6 :: ('label, 'state) nba ⇒ 'state items ⇒ 'state items
  set nres where
  expand-map-get-6 A f ≡ FOREACH (map-to-set f) (λ (k, v) X. do {
      ASSERT (∀ g ∈ X. k ∉ dom g);
    ASSERT (∀ a ∈ (items-5 A k v). ∀ b ∈ (items-5 A k v). a ≠ b → (λ y. (λ g. g (k ↦ y)) ′ X) a ∩ (λ y. (λ g. g (k ↦ y)) ′ X) b = {})});
\[
\text{RETURN } (\bigcup y \in \text{items-5} \ A \ k \ v. (\lambda g. g (k \mapsto y)) \ X) \\
} \{\text{Map.empty}\}
\]

**lemma** expand-map-get-6-refine: \((\text{expand-map-get-6}, \text{expand-5} \circ \text{get-5}) \in \text{Id} \to \text{Id} \to \text{Id} \to \text{Id} \to \text{nres-rel})

**unfolding** expand-map-get-6-def expand-5-def get-5-def by (auto intro: FORE-ACH-rule-map-map[\text{param-fol}])

**definition** complement-succ-6 ::
\((\text{\'label, \'state}) \text{nba} \Rightarrow \text{\'label} \Rightarrow \text{\'state items} \Rightarrow \text{\'state items set nres where where}
\text{complement-succ-6} \ A \ a \ f \equiv \text{do}
\begin{align*}
& f \leftarrow \text{refresh-5} \ f; \\
& f \leftarrow \text{bounds-5} \ A \ a \ f; \\
& \text{ASSUME} \ (\text{finite} \ (\text{dom} \ f)); \\
& \text{expand-map-get-6} \ A \ f
\end{align*}
\}

**lemma** complement-succ-6-refine:
\((\text{complement-succ-6, complement-succ-5}) \in \text{Id} \to \text{Id} \to \text{Id} \to \text{Id} \to \text{nres-rel})

**unfolding** complement-succ-6-def complement-succ-5-def

by (refine-vcg vcg2[OF expand-map-get-6-refine]) (auto intro: refine-IdI)

5.7 Phase 7

**interpretation** autoref-syn by this

**context**
fixes \(fi \ f\)
assumes \(fi[\text{autoref-rules}]: (fi, f) \in \text{state-rel}\)
begin

**private lemma** [simp]: \(\text{finite} \ (\text{dom} \ f)\)

using list-map-rel-finite \(fi\) unfolding finite-map-rel-def by force

**schematic-goal** refresh-7: \((?f :: \text{?a}, \text{refresh-5} \ f) \in \text{\?R}\)

unfolding refresh-5-def by (autoref-monadic (plain))

end

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

**lemma** refresh-7-refine: \((\lambda f. \text{RETURN} \ (\text{refresh-7} \ f), \text{refresh-5} \in \text{state-rel} \to \text{state-rel}) \to \text{nres-rel}\)

using refresh-7.refine by fast

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

50
assumes \texttt{succi[autoref-rules]}: (succi, transition A a) ∈ nat-rel → (nat-rel)
\texttt{list-set-rel}
assumes \texttt{fi[autoref-rules]}: (fi, f) ∈ state-rel

begin

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

private lemma \texttt{[autoref-op-pat]}: transition A a ≡ OP (transition A a) by simp

private lemma \texttt{[autoref-rules]}: (\lambda k l. upt k (Suc l), atLeastAtMost) ∈ nat-rel → nat-rel → (nat-rel) list-set-rel
by (auto simp: list-set-rel-def in-br-conv)
schematic-goal items-7: (?f :: ?'a, items-5 A) ∈ ?R
unfolding \texttt{items-5-def Let-def Set.filter-def} by autoref

end

concrete-definition items-7 uses items-7

51
context
  fixes A :: ('label, nat) nba
  fixes ai
  fixes fi f
assumes ai: (ai, accepting A) ∈ nat-rel → bool-rel
assumes fi[autoref-rules]: (fi, f) ∈ ⟨nat-rel, item-rel⟩ 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 \( \bigwedge m. m \in S \implies x \notin dom m \)
  shows inj-on (\( \lambda m. m (x \mapsto y) \)) S
  using assms unfolding dom-def inj-on-def by (auto) (metis fun-upd-triv
fun-upd-upd)
private lemmas [simp] = op-map-update-def[abs-def]
private lemma [autoref-op-pat]: items-5 A ≡ OP (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) ∈ \langle \langle 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) ∈ nat-rel → bool-rel
  shows (\( \lambda f. RETURN (expand-map-get-7 ai f) \),
    \( \lambda f. ASSUME (finite (dom f)) \gg expand-map-get-6 A f \) ∈ 
    \langle \langle nat-rel, item-rel \rangle dflt-ahm-rel \rangle \langle \langle state-rel \rangle list-set-rel \rangle 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 Ai
  fixes ai
  fixes pi
assumes Ai: (Ai, A) ∈ ⟨Id, Id⟩ nbai-nba-rel
assumes ai: (ai, a) ∈ Id
assumes pi[autoref-rules]: (pi, p) ∈ 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)) \Rightarrow expand-map-get-6 A g) \equiv
OP (\lambda g. ASSUME (finite (dom g)) \Rightarrow expand-map-get-6 A g) \text{ by simp }

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

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

schematic-goal complement-succ-7: (?f :: ?'a, complement-succ-6 A a p) \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 \circ \circ \circ complement-succ-7, complement-succ-6) \in
\langle (Id, Id) nbai-nba-rel \rightarrow Id \rightarrow state-rel \rightarrow
\langle (state-rel) list-set-rel \rangle nres-rel
using complement-succ-7.refine unfolding comp-apply by parametricity

context
  fixes A :: ('label, nat) nba
  fixes Ai
  fixes n ni :: nat
  assumes Ai: (Ai, A) \in \langle Id, Id \rangle nbai-nba-rel
  assumes ni[autoref-rules]: (ni, n) \in Id

begin

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

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

schematic-goal complement-initial-7: 
(?f, \{(\text{Some} \circ (\text{const} (2 + n, \text{False})) | \text{ initial A})\} \in \langle state-rel \rangle list-set-rel
by autoref

end

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

53
schematic-goal complement-accepting-7: (∀f, λf. ∀(p, k, c) ∈ map-to f. ¬c) ∈ state-rel → bool-rel
by autoref

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

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

lemma complement-7-refine[autoref-rules]:
assumes (Ai, A) ∈ ⟨Id, Id⟩ nbai-nba-rel
assumes (ni,
(OP card :: (Id) abs-rel bhc → nat-rel) $
((OP nodes :: (Id, Id) nbai-nba-rel → (Id) abs-rel bhc) $ A)) ∈ nat-rel
shows (complement-7 Ai ni, (OP complement-4 ::
(Id, Id) nbai-nba-rel → (Id, state-rel) nbai-nba-rel) $ A) ∈ ⟨Id, state-rel⟩ nbai-nba-rel

proof —
note complement-succ-7-refine
also note complement-succ-6-refine
also note complement-succ-5-refine
finally have 1: (complement-succ-7, complement-succ-4) ∈
⟨Id, Id⟩ nbai-nba-rel → Id → state-rel → ⟨state-rel⟩ list-set-rel
unfolding nres-rel-comp unfolding nres-rel-def unfolding fun-rel-def by auto
show ?thesis
  unfolding complement-7-def complement-4-def
  using 1 complement-initial-7.refine complement-accepting-7.refine assms
  unfolding autoref-tag-defs
  by parametricity
qed

end

6 Boolean Formulae

theory Formula
imports Main
begin

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

54
Negation 'a formula |
Conjunction 'a formula 'a formula |
Disjunction 'a formula 'a formula |

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

end

7 Final Instantiation of Algorithms Related to Complementation

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

begin

7.1 Syntax

  syntax -do-let :: [pttrn, 'a] ⇒ do-bind ((2let - =/ -) 13)

7.2 Hashcodes on Complement States

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

lemma list-hash-eq:
  assumes distinct xs distinct ys set xs = set ys
  shows list-hash xs = list-hash ys
proof –
  have mset (remdups xs) = mset (remdups ys) using assms(3)
    using set-eq-iff-mset-remdups-eq by blast
  then have mset xs = mset ys using assms(1, 2) by (simp add: distinct-remdups-id)
  have fold (xor ◦ hc) xs = fold (xor ◦ hc) ys
    apply (rule fold-multiset-equiv)
    apply (simp-all add: fun-eq-iff ac-simps)
    using mset xs = mset ys .
  then show ?thesis unfolding list-hash-def by simp
qed

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

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

7.3 Complementation

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

theorem complement-impl-correct:
assumes finite (NBA.nodes A)
\textbf{assumes} \((Ai, A) \in \langle \text{Id, nat-rel} \rangle \text{ nbai-nba-rel}\\
\text{shows} NBA.\text{language} \left(\text{nbae-nba} \left(\text{nbae-i-nbae} \left(\complement-impl Ai\right)\right)\right) =\\
\text{streams} \left(\text{nba.alphabet} A\right) - NBA.\text{language} A\\
\text{using} \text{op-translate-language}[\text{OF} \complement-impl.\text{refine}[\text{OF} \text{assms}]]\\
\text{using} \text{complement-4-correct}[\text{OF} \text{assms(1)}]\\
\text{by} \ simp\\

\textbf{7.4 Language Subset}\\
\textbf{definition} [\text{simp}]: \text{op-language-subset} A B \equiv NBA.\text{language} A \subseteq NBA.\text{language} B\\
\textbf{lemmas} [\text{autoref-op-pat}] = \text{op-language-subset-def}\text{[symmetric]}\\
\textbf{schematic-goal} language-subset-impl:
\textbf{assumes} [\text{simp}]: \text{finite} (NBA.\text{nodes} B)\\
\text{assumes} [\text{autoref-rules}]: (Ai, A) \in \langle \text{Id, nat-rel} \rangle \text{ nbai-nba-rel}\\
\text{assumes} [\text{autoref-rules}]: (Bi, B) \in \langle \text{Id, nat-rel} \rangle \text{ nbai-nba-rel}\\
\text{shows} (\exists f :: \text{?c}, \text{do} \{\\
\text{let} AB' = \text{intersect}' A (\complement-4 B);\\
\text{ASSERT} (\text{finite} (NGBA.\text{nodes} AB'));\\
\text{RETURN} (NGBA.\text{language} AB' = \{\})\\
\}) \in \text{?R}\\
\text{by} (\text{autoref-monadic}\text{(plain)})\\
\textbf{concrete-definition} language-subset-impl \text{uses language-subset-impl}\\
\textbf{lemma} language-subset-impl-refine[\text{autoref-rules}]:
\textbf{assumes} \text{SIDE-PRECOND} (\text{finite} (NBA.\text{nodes} A))\\
\text{assumes} \text{SIDE-PRECOND} (\text{finite} (NBA.\text{nodes} B))\\
\text{assumes} \text{SIDE-PRECOND} (\text{nba.alphabet} A \subseteq \text{nba.alphabet} B)\\
\text{assumes} (Ai, A) \in \langle \text{Id, nat-rel} \rangle \text{ nbai-nba-rel}\\
\text{assumes} (Bi, B) \in \langle \text{Id, nat-rel} \rangle \text{ nbai-nba-rel}\\
\text{shows} (\text{language-subset-impl} Ai Bi, (\text{OP} \text{op-language-subset} ::: (\text{Id, nat-rel} \text{ nbai-nba-rel} \rightarrow (\text{Id, nat-rel} \text{ nbai-nba-rel} \rightarrow \text{bool-rel}) \$ A \$ B)) \in \text{bool-rel}\\
\text{proof} –\\
\text{have} (\text{RETURN} (\text{language-subset-impl} Ai Bi), \text{do} \{\\
\text{let} AB' = \text{intersect}' A (\complement-4 B);\\
\text{ASSERT} (\text{finite} (NGBA.\text{nodes} AB'));\\
\text{RETURN} (NGBA.\text{language} AB' = \{\})\\
\}) \in (\text{bool-rel}) \text{nres-rel}\\
\text{using} \text{language-subset-impl.refine} \text{assms}(2, 4, 5) \text{ unfolding} \text{autoref-tag-defs}\\
\text{by this}\\
\text{also have} (\text{do} \{\\
\text{let} AB' = \text{intersect}' A (\complement-4 B);\\
\text{ASSERT} (\text{finite} (NGBA.\text{nodes} AB'));\\
\text{RETURN} (NGBA.\text{language} AB' = \{\})\\
\}), \text{RETURN} (NBA.\text{language} A \subseteq NBA.\text{language} B) \in (\text{bool-rel}) \text{nres-rel}\\
\text{proof} \text{refine-vcg}\\
\text{show} \text{finite} (\text{NGBA.\nodes (intersect'} A (\complement-4 B))) \text{ using} \text{assms}(1,\text{...})
2) by auto
have 1: NBA.language A \subseteq \text{streams}(\text{nba.alphabet B})
using nba.language-alphabet streams-mono2 assms(3) unfolding autoref-tag-defs
by blast
have 2: NBA.language (\text{complement-4 } B) = \text{streams}(\text{nba.alphabet B}) - NBA.language B
using complement-4-correct assms(2) by auto
show (NGBA.language (\text{intersect}' A (\text{complement-4 } B))) = \{\},
NBA.language A \subseteq NBA.language B \in \text{bool-rel} using 1 2 by auto
qed
finally show \text{thesis} using \text{RETURN-nres-relD} unfolding nres-rel-comp by force
qed

7.5 Language Equality

definition [simp]: \text{op-language-equal } A B \equiv NBA.language A = NBA.language B

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

schematic-goal language-equal-impl:
assumes [simp]: finite (NBA.nodes A)
assumes [simp]: finite (NBA.nodes B)
assumes [simp]: nba.alphabet A = nba.alphabet B
assumes [autoref-rules]: (Ai, A) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
assumes [autoref-rules]: (Bi, B) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
shows (\text{if } :: \text{'c}, NBA.language A \subseteq NBA.language B \land NBA.language B \subseteq NBA.language A) \in \text{?R}
by autoref
concrete-definition language-equal-impl uses language-equal-impl
lemma language-equal-impl-refine[autoref-rules]:
assumes SIDE-PRECOND (finite (NBA.nodes A))
assumes SIDE-PRECOND (finite (NBA.nodes B))
assumes SIDE-PRECOND (nba.alphabet A = nba.alphabet B)
assumes (Ai, A) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
assumes (Bi, B) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
shows (\text{language-equal-impl Ai Bi, (OP op-language-equal :::}
\langle Id, \text{nat-rel} \rangle nbai-nba-rel \rightarrow \langle Id, \text{nat-rel} \rangle nbai-nba-rel \rightarrow \text{bool-rel}) \in \text{bool-rel}
using language-equal-impl.refine[\text{OF assms[unfolded autoref-tag-defs]}] by auto

schematic-goal product-impl:
assumes [simp]: finite (NBA.nodes B)
assumes [autoref-rules]: (Ai, A) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
assumes [autoref-rules]: (Bi, B) \in \langle Id, \text{nat-rel} \rangle nbai-nba-rel
shows (\text{if } :: \text{'c}, \text{do}
\text{let } AB' = \text{intersect } A (\text{complement-4 } B);
\text{ASSERT (finite (NBA.nodes AB'))};

58
op-translate \( AB' \)
\( ) \in \mathbb{R} \)
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
\( nbaei \) alphabeti initialei transitioni acceptingi
\( nbae-nba-impl \) complement-impl language-equal-impl product-impl
in SML module-name Complementation file-prefix Complementation

end

8 Build and test exported program with MLton

theory Complementation-Build
imports Complementation-Final
begin

external-file (code/Autool.mlb)

external-file (code/Prelude.sml)

external-file (code/Autool.sml)

compile-generated-files
(code/Complementation.ML) (in Complementation-Final)

external-files
(code/Autool.mlb)
(code/Prelude.sml)
(code/Autool.sml)

export-files (code/Complementation.sml) and (code/Autool) (exe)
where fn dir =>
let
val exec = Generated-Files.execute (dir + Path.basic code);
val - = exec (Prepare) me Complementation.ML Complementation.sml;
val - = exec (Compilation) (verbatim "$ISABELLE-MLTON$ISABELLE-MLTON-OPTIONS

- profile time - default - type intinf Autool.mlb);
val - = exec (Test) ./Autool help;
in () end

end
References