# Büchi Complementation

Julian Brunner

February 23, 2021

**Abstract**

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

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

## Contents

# 1 Alternating Function Iteration

**theory** *Alternate*
**imports** *Main*
**begin**

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

**lemma** *alternate-Suc*[*simp*]: *alternate f g* (*Suc k*) = (*if even k then f else g*) ∘ *alternate f g k*
⟨*proof*⟩

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

**lemma** *alternate-antimono*:
**assumes** ⋀ *x*. *f x* ≤ *x* ⋀ *x*. *g x* ≤ *x*
**shows** *antimono* (*alternate f g*)
⟨*proof*⟩

**end**

# 2 Run Graphs

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

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

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

**global-interpretation** *graph*: *transition-system-initial*
*const*
λ *u* (*k*, *p*). *w* !! *k* ∈ *alphabet A* ∧ *u* ∈ {*Suc k*} × *transition A* (*w* !! *k*) *p* ∩ *V*
λ *v*. *v* ∈ *ginitial A* ∩ *V*
**for** *A w V*
**defines**
*gpath* = *graph.path* **and** *grun* = *graph.run* **and**
*greachable* = *graph.reachable* **and** *gnodes* = *graph.nodes*
⟨*proof*⟩

We disable rules that are degenerate due to *execute* = (λ*x* -. *x*).

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

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

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

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

**lemma** *gtarget-alt-def*: *gtarget r v* = *last* (*v* # *r*) ⟨*proof*⟩
**lemma** *gstates-alt-def*: *gstates r v* = *r* ⟨*proof*⟩
**lemma** *gtrace-alt-def*: *gtrace r v* = *r* ⟨*proof*⟩

**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*⟩

**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*⟩

**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*⟩

**lemma** *run-grun*:
**assumes** *run A* (*sdrop k w* ||| *r*) *p*
**shows** *gurun A w* (*fromN* (*Suc k*) ||| *r*) (*k*, *p*)
⟨*proof*⟩

**lemma** *grun-run*:
**assumes** *grun A w V* (*fromN* (*Suc k*) ||| *r*) (*k*, *p*)
**shows** *run A* (*sdrop k w* ||| *r*) *p*
⟨*proof*⟩

**lemma** *greachable-reachable*:
**fixes** *l q k p*
**defines** $u \equiv (l,\ q)$
**defines** $v \equiv (k,\ p)$
**assumes** $u \in greachable\ A\ w\ V\ v$
**shows** $q \in reachable\ A\ p$
⟨*proof*⟩

**lemma** *gnodes-nodes*: $gnodes\ A\ w\ V \subseteq UNIV \times nodes\ A$
⟨*proof*⟩

**lemma** *gpath-subset*:
**assumes** *gpath A w V r v*
**assumes** $set\ (gstates\ r\ v) \subseteq U$
**shows** *gpath A w U r v*
⟨*proof*⟩
**lemma** *grun-subset*:
**assumes** *grun A w V r v*
**assumes** $sset\ (gtrace\ r\ v) \subseteq U$
**shows** *grun A w U r v*
⟨*proof*⟩

**lemma** *greachable-subset*: $greachable\ A\ w\ V\ v \subseteq insert\ v\ V$
⟨*proof*⟩

**lemma** *gtrace-infinite*:
**assumes** *grun A w V r v*
**shows** *infinite* (*sset* (*gtrace r v*))
⟨*proof*⟩

**lemma** *infinite-greachable-gtrace*:
**assumes** *grun A w V r v*
**assumes** $u \in sset\ (gtrace\ r\ v)$
**shows** *infinite* (*greachable A w V u*)

⟨*proof*⟩

**lemma** *finite-nodes-gsuccessors*:
**assumes** *finite* (*nodes A*)
**assumes** $v \in$ *gunodes A w*
**shows** *finite* (*gusuccessors A w v*)
⟨*proof*⟩

**end**

# 3 Rankings

**theory** *Ranking*
**imports**
*Alternate*
*Graph*
**begin**

## 3.1 Rankings

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

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

## 3.2 Ranking Implies Word not in Language

**lemma** *ranking-stuck*:
**assumes** *ranking A w f*
**assumes** *v* ∈ *gunodes A w gurun A w r v*
**obtains** *n k*
**where** *smap f* (*gtrace* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)) = *sconst k*
⟨*proof*⟩

**lemma** *ranking-stuck-odd*:
**assumes** *ranking A w f*
**assumes** *v* ∈ *gunodes A w gurun A w r v*
**obtains** *n*
**where** *Ball* (*sset* (*smap f* (*gtrace* (*sdrop n r*) (*gtarget* (*stake n r*) *v*)))) *odd*
⟨*proof*⟩

**lemma** *ranking-language*:
**assumes** *ranking A w f*

**shows** $w \notin$ *language* $A$
⟨*proof*⟩

## 3.3 Word not in Language Implies Ranking

### 3.3.1 Removal of Endangered Nodes

**definition** *clean* :: ('*label*, '*state*) *nba* ⇒ '*label stream* ⇒ '*state node set* ⇒ '*state node set* **where**
*clean A w V* ≡ {*v* ∈ *V*. *infinite* (*greachable A w V v*)}

**lemma** *clean-decreasing*: *clean A w V* ⊆ *V* ⟨*proof*⟩
**lemma** *clean-successors*:
**assumes** *v* ∈ *V u* ∈ *gusuccessors A w v*
**shows** *u* ∈ *clean A w V* ⟹ *v* ∈ *clean A w V*
⟨*proof*⟩

### 3.3.2 Removal of Safe Nodes

**definition** *prune* :: ('*label*, '*state*) *nba* ⇒ '*label stream* ⇒ '*state node set* ⇒ '*state node set* **where**
*prune A w V* ≡ {*v* ∈ *V*. ∃ *u* ∈ *greachable A w V v*. *gaccepting A u*}

**lemma** *prune-decreasing*: *prune A w V* ⊆ *V* ⟨*proof*⟩
**lemma** *prune-successors*:
**assumes** *v* ∈ *V u* ∈ *gusuccessors A w v*
**shows** *u* ∈ *prune A w V* ⟹ *v* ∈ *prune A w V*
⟨*proof*⟩

### 3.3.3 Run Graph Interation

**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* ⟨*proof*⟩
**lemma** *graph-Suc*[*simp*]: *graph A w* (*Suc k*) = (*if even k then clean A w else prune A w*) (*graph A w k*)
⟨*proof*⟩

**lemma** *graph-antimono*: *antimono* (*graph A w*)
⟨*proof*⟩
**lemma** *graph-nodes*: *graph A w k* ⊆ *gunodes A w* ⟨*proof*⟩
**lemma** *graph-successors*:
**assumes** *v* ∈ *gunodes A w u* ∈ *gusuccessors A w v*
**shows** *u* ∈ *graph A w k* ⟹ *v* ∈ *graph A w k*
⟨*proof*⟩

**lemma** *graph-level-finite*:
**assumes** *finite* (*nodes A*)
**shows** *finite* (*level A w k l*)
⟨*proof*⟩

**lemma** *find-safe*:
**assumes** $w \notin$ *language A*
**assumes** $V \neq \{\}$ $V \subseteq$ *gunodes A w*
**assumes** $\bigwedge v.\ v \in V \Longrightarrow$ *gsuccessors A w V v* $\neq \{\}$
**obtains** *v*
**where** $v \in V\ \forall\ u \in$ *greachable A w V v*. $\neg$ *gaccepting A u*
⟨*proof*⟩

**lemma** *remove-run*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A*
**assumes** $V \subseteq$ *gunodes A w clean A w V* $\neq \{\}$
**obtains** *v r*
**where**
*grun A w V r v*
*sset* (*gtrace r v*) $\subseteq$ *clean A w V*
*sset* (*gtrace r v*) $\subseteq -$ *prune A w* (*clean A w V*)
⟨*proof*⟩

**lemma** *level-bounded*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A*
**obtains** *n*
**where** $\bigwedge l.\ l \geq n \Longrightarrow$ *card* (*level A w* (*2* $*$ *k*) *l*) $\leq$ *card* (*nodes A*) $-$ *k*
⟨*proof*⟩
**lemma** *graph-empty*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A*
**shows** *graph A w* (*Suc* (*2* $*$ *card* (*nodes A*))) $= \{\}$
⟨*proof*⟩
**lemma** *graph-le*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A*
**assumes** $v \in$ *graph A w k*
**shows** $k \leq$ *2* $*$ *card* (*nodes A*)
⟨*proof*⟩

## 3.4 Node Ranks

**definition** *rank* :: (*'label*, *'state*) *nba* $\Rightarrow$ *'label stream* $\Rightarrow$ *'state node* $\Rightarrow$ *nat* **where**
*rank A w v* $\equiv$ *GREATEST k*. $v \in$ *graph A w k*

**lemma** *rank-member*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A* $v \in$ *gunodes A w*
**shows** $v \in$ *graph A w* (*rank A w v*)
⟨*proof*⟩
**lemma** *rank-removed*:
**assumes** *finite* (*nodes A*) $w \notin$ *language A*

**shows** $v \notin \mathit{graph}\ A\ w\ (\mathit{Suc}\ (\mathit{rank}\ A\ w\ v))$
⟨*proof*⟩
**lemma** *rank-le*:
**assumes** *finite* (*nodes* $A$) $w \notin \mathit{language}\ A$
**assumes** $v \in \mathit{gunodes}\ A\ w\ u \in \mathit{gusuccessors}\ A\ w\ v$
**shows** $\mathit{rank}\ A\ w\ u \leq \mathit{rank}\ A\ w\ v$
⟨*proof*⟩

**lemma** *language-ranking*:
**assumes** *finite* (*nodes* $A$) $w \notin \mathit{language}\ A$
**shows** $\mathit{ranking}\ A\ w\ (\mathit{rank}\ A\ w)$
⟨*proof*⟩

### 3.5 Correctness Theorem

**theorem** *language-ranking-iff*:
**assumes** *finite* (*nodes* $A$)
**shows** $w \notin \mathit{language}\ A \longleftrightarrow (\exists\ f.\ \mathit{ranking}\ A\ w\ f)$
⟨*proof*⟩

**end**

## 4 Complementation

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

### 4.1 Level Rankings and Complementation States

**type-synonym** $'\mathit{state}\ \mathit{lr} = '\mathit{state} \rightharpoonup \mathit{nat}$

**definition** *lr-succ* :: ($'\mathit{label}$, $'\mathit{state}$) *nba* ⇒ $'\mathit{label}$ ⇒ $'\mathit{state}\ \mathit{lr}$ ⇒ $'\mathit{state}\ \mathit{lr}\ \mathit{set}$ **where**
*lr-succ* $A\ a\ f \equiv \{g.$
$\mathit{dom}\ g = \bigcup\ (\mathit{transition}\ A\ a\ `\ \mathit{dom}\ f) \wedge$
$(\forall\ p \in \mathit{dom}\ f.\ \forall\ q \in \mathit{transition}\ A\ a\ p.\ \mathit{the}\ (g\ q) \leq \mathit{the}\ (f\ p)) \wedge$
$(\forall\ q \in \mathit{dom}\ g.\ \mathit{accepting}\ A\ q \longrightarrow \mathit{even}\ (\mathit{the}\ (g\ q)))\}$

**type-synonym** $'\mathit{state}\ \mathit{st} = '\mathit{state}\ \mathit{set}$

**definition** *st-succ* :: ($'\mathit{label}$, $'\mathit{state}$) *nba* ⇒ $'\mathit{label}$ ⇒ $'\mathit{state}\ \mathit{lr}$ ⇒ $'\mathit{state}\ \mathit{st}$ ⇒ $'\mathit{state}\ \mathit{st}$ **where**
*st-succ* $A\ a\ g\ P \equiv \{q \in \mathit{if}\ P = \{\}\ \mathit{then}\ \mathit{dom}\ g\ \mathit{else}\ \bigcup\ (\mathit{transition}\ A\ a\ `\ P).\ \mathit{even}\ (\mathit{the}\ (g\ q))\}$

**type-synonym** $'\mathit{state}\ \mathit{cs} = '\mathit{state}\ \mathit{lr} \times '\mathit{state}\ \mathit{st}$

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

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

**lemma** *dom-nodes*:
**assumes** *fP* ∈ *nodes* (*complement A*)
**shows** *dom* (*fst fP*) ⊆ *nodes A*
⟨*proof*⟩
**lemma** *ran-nodes*:
**assumes** *fP* ∈ *nodes* (*complement A*)
**shows** *ran* (*fst fP*) ⊆ {*0* .. *2* ∗ *card* (*nodes A*)}
⟨*proof*⟩
**lemma** *states-nodes*:
**assumes** *fP* ∈ *nodes* (*complement A*)
**shows** *snd fP* ⊆ *nodes A*
⟨*proof*⟩

**theorem** *complement-finite*:
**assumes** *finite* (*nodes A*)
**shows** *finite* (*nodes* (*complement A*))
⟨*proof*⟩

**lemma** *complement-trace-snth*:
**assumes** *run* (*complement A*) (*w* ||| *r*) *p*
**defines** *m* ≡ *p* ## *trace* (*w* ||| *r*) *p*
**obtains**
*fst* (*m* !! *Suc k*) ∈ *lr-succ A* (*w* !! *k*) (*fst* (*m* !! *k*))
*snd* (*m* !! *Suc k*) = *st-succ A* (*w* !! *k*) (*fst* (*m* !! *Suc k*)) (*snd* (*m* !! *k*))
⟨*proof*⟩

## 4.2 Word in Complement Language Implies Ranking

**lemma** *complement-ranking*:
**assumes** *w* ∈ *language* (*complement A*)
**obtains** *f*
**where** *ranking A w f*
⟨*proof*⟩

## 4.3 Ranking Implies Word in Complement Language

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

**lemma** *reach-0*[*simp*]: *reach A w 0 = initial A* ⟨*proof*⟩
**lemma** *reach-Suc-empty*:
**assumes** *w !! n ∉ alphabet A*
**shows** *reach A w* (*Suc n*) = {}
⟨*proof*⟩
**lemma** *reach-Suc-succ*:
**assumes** *w !! n ∈ alphabet A*
**shows** *reach A w* (*Suc n*) = ⋃ (*transition A* (*w !! n*) ‘ *reach A w n*)
⟨*proof*⟩
**lemma** *reach-Suc*[*simp*]: *reach A w* (*Suc n*) = (*if w !! n ∈ alphabet A*
*then* ⋃ (*transition A* (*w !! n*) ‘ *reach A w n*) *else* {})
⟨*proof*⟩
**lemma** *reach-nodes*: *reach A w i* ⊆ *nodes A* ⟨*proof*⟩
**lemma** *reach-gunodes*: {*i*} × *reach A w i* ⊆ *gunodes A w*
⟨*proof*⟩

**lemma** *ranking-complement*:
**assumes** *finite* (*nodes A*) *w* ∈ *streams* (*alphabet A*) *ranking A w f*
**shows** *w* ∈ *language* (*complement A*)
⟨*proof*⟩

### 4.4 Correctness Theorem

**theorem** *complement-language*:
**assumes** *finite* (*nodes A*)
**shows** *language* (*complement A*) = *streams* (*alphabet A*) − *language A*
⟨*proof*⟩

**end**

## 5 Complementation Implementation

**theory** *Complementation-Implement*
**imports**
*HOL−Library.Lattice-Syntax*
*Transition-Systems-and-Automata.NBA-Implement*
*Complementation*
**begin**

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

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

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

## 5.1 Phase 1

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

**lemma** *cs-abs-rep*[*simp*]: *cs-rep* (*cs-abs f*) = *f*
⟨*proof*⟩
**lemma** *cs-rep-lr*[*simp*]: *cs-lr* (*cs-rep* (*g*, *P*)) = *g*
⟨*proof*⟩
**lemma** *cs-rep-st*[*simp*]: *cs-st* (*cs-rep* (*g*, *P*)) = *P* ∩ *dom g*
⟨*proof*⟩

**lemma** *cs-lr-dom*[*simp*]: *dom* (*cs-lr f*) = *dom f* ⟨*proof*⟩
**lemma** *cs-lr-apply*[*simp*]:
**assumes** *p* ∈ *dom f*
**shows** *the* (*cs-lr f p*) = *fst* (*the* (*f p*))
⟨*proof*⟩

**lemma** *cs-rep-dom*[*simp*]: *dom* (*cs-rep* (*g*, *P*)) = *dom g* ⟨*proof*⟩
**lemma** *cs-rep-apply*[*simp*]:
**assumes** *p* ∈ *dom f*
**shows** *fst* (*the* (*cs-rep* (*f*, *P*) *p*)) = *the* (*f p*)
⟨*proof*⟩

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

**lemma** *cs-rel-inv-single-valued*: *single-valued* (*cs-rel*$^{-1}$)
⟨*proof*⟩

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

(*alphabet A*)
({*const* (*Some* (2 ∗ *card* (*nodes A*), *False*)) |‘ *initial A*})
(*complement-succ-1 A*)
(λ *f*. *cs-st f* = {})

**lemma** *refresh-1-dom*[*simp*]: *dom* (*refresh-1 f*) = *dom f* ⟨*proof*⟩
**lemma** *refresh-1-apply*[*simp*]: *fst* (*the* (*refresh-1 f p*)) = *fst* (*the* (*f p*))
⟨*proof*⟩
**lemma** *refresh-1-cs-st*[*simp*]: *cs-st* (*refresh-1 f*) = (*if cs-st f* = {} *then dom f else cs-st f*)
⟨*proof*⟩

**lemma** *complement-succ-1-abs*:
**assumes** *g* ∈ *complement-succ-1 A a f*
**shows** *cs-abs g* ∈ *complement-succ A a* (*cs-abs f*)
⟨*proof*⟩
**lemma** *complement-succ-1-rep*:
**assumes** *P* ⊆ *dom f* (*g*, *Q*) ∈ *complement-succ A a* (*f*, *P*)
**shows** *cs-rep* (*g*, *Q*) ∈ *complement-succ-1 A a* (*cs-rep* (*f*, *P*))
⟨*proof*⟩

**lemma** *complement-succ-1-refine*: (*complement-succ-1*, *complement-succ*) ∈
*Id* → *Id* → *cs-rel* → ⟨*cs-rel*⟩ *set-rel*
⟨*proof*⟩
**lemma** *complement-1-refine*: (*complement-1*, *complement*) ∈ ⟨*Id*, *Id*⟩ *nba-rel* → ⟨*Id*, *cs-rel*⟩ *nba-rel*
⟨*proof*⟩

## 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*⟩

**lemma** *complement-2-refine*: (*complement-2*, *complement-1*) ∈ ⟨*Id*, *Id*⟩ *nba-rel* → ⟨*Id*, *Id*⟩ *nba-rel*
⟨*proof*⟩

## 5.3 Phase 3

**definition** *bounds-3* :: ('*label*, '*state*) *nba* ⇒ '*label* ⇒ '*state items* ⇒ '*state items* **where**
*bounds-3 A a f* ≡ λ *q*. *let S* = *Some* − ‘ *f* ‘ *pred A a q in*
*if S* = {} *then None else Some* (⋂ (*fst* ‘ *S*), ⨆ (*snd* ‘ *S*))
**definition** *items-3* :: ('*label*, '*state*) *nba* ⇒ '*state* ⇒ *item* ⇒ *item set* **where**
*items-3 A p* ≡ λ (*k*, *c*). {(*l*, *c* ∧ *even l*) |*l*. *l* ≤ *k* ∧ (*accepting A p* ⟶ *even l*)}
**definition** *get-3* :: ('*label*, '*state*) *nba* ⇒ '*state items* ⇒ ('*state* ⇀ *item set*) **where**
*get-3 A f* ≡ λ *p*. *map-option* (*items-3 A p*) (*f p*)
**definition** *complement-succ-3* ::
('*label*, '*state*) *nba* ⇒ '*label* ⇒ '*state items* ⇒ '*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*
(*alphabet A*)
({(*Some* ∘ (*const* (2 ∗ *card* (*nodes A*), *False*))) |‘ *initial A*})
(*complement-succ-3 A*)
(λ *f*. ∀ (*p*, *k*, *c*) ∈ *map-to-set f*. ¬ *c*)

**lemma** *bounds-3-dom*[*simp*]: *dom* (*bounds-3 A a f*) = ⋃ ((*transition A a*) ‘ (*dom f*))
⟨*proof*⟩

**lemma** *items-3-nonempty*[*intro*!, *simp*]: *items-3 A p s* ≠ {} ⟨*proof*⟩
**lemma** *items-3-finite*[*intro*!, *simp*]: *finite* (*items-3 A p s*)
⟨*proof*⟩

**lemma** *get-3-dom*[*simp*]: *dom* (*get-3 A f*) = *dom f* ⟨*proof*⟩
**lemma** *get-3-finite*[*intro*, *simp*]: *S* ∈ *ran* (*get-3 A f*) ⟹ *finite S*
⟨*proof*⟩
**lemma** *get-3-update*[*simp*]: *get-3 A* (*f* (*p* ↦ *s*)) = (*get-3 A f*) (*p* ↦ *items-3 A p s*)
⟨*proof*⟩

**lemma** *expand-map-get-bounds-3*: *expand-map* ∘ *get-3 A* ∘ *bounds-3 A a* = *ranks-2 A a*
⟨*proof*⟩

**lemma** *complement-succ-3-refine*: *complement-succ-3* = *complement-succ-2*
⟨*proof*⟩
**lemma** *complement-initial-3-refine*: {*const* (*Some* (2 ∗ *card* (*nodes A*), *False*))

$|`$ *initial A*$\}$ =
$\{$(*Some* $\circ$ (*const* (*2* $*$ *card* (*nodes A*), *False*))) $|`$ *initial A*$\}$
⟨*proof*⟩
**lemma** *complement-accepting-3-refine*: *True* $\notin$ *snd* ` *ran f* $\longleftrightarrow$ ($\forall$ (*p*, *k*, *c*) $\in$ *map-to-set f*. $\neg$ *c*)
⟨*proof*⟩

**lemma** *complement-3-refine*: (*complement-3*, *complement-2*) $\in$ ⟨*Id*, *Id*⟩ *nba-rel* $\rightarrow$ ⟨*Id*, *Id*⟩ *nba-rel*
⟨*proof*⟩

## 5.4 Phase 4

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

**lemma** *get-4-dom*[*simp*]: *dom* (*get-4 A f*) = *dom f* ⟨*proof*⟩

**definition** *R* :: '*state items rel* **where**
*R* $\equiv$ $\{$(*f*, *g*).
*dom f* = *dom g* $\wedge$
($\forall$ *p* $\in$ *dom f*. *fst* (*the* (*f p*)) $\leq$ *fst* (*the* (*g p*))) $\wedge$
($\forall$ *p* $\in$ *dom f*. *snd* (*the* (*f p*)) $\longleftrightarrow$ *snd* (*the* (*g p*)))$\}$

**lemma** *bounds-R*:
**assumes** (*f*, *g*) $\in$ *R*
**assumes** *bounds-3 A a* (*refresh-1 f*) *p* = *Some* (*n*, *e*)
**assumes** *bounds-3 A a* (*refresh-1 g*) *p* = *Some* (*k*, *c*)
**shows** *n* $\leq$ *k e* $\longleftrightarrow$ *c*
⟨*proof*⟩

**lemma** *complement-4-language-1*: *language* (*complement-3 A*) $\subseteq$ *language* (*complement-4 A*)
⟨*proof*⟩
**lemma** *complement-4-less*: *complement-4 A* $\leq$ *complement-3 A*
⟨*proof*⟩

**lemma** *complement-4-language-2*: *language* (*complement-4 A*) ⊆ *language* (*complement-3 A*)
⟨*proof*⟩
**lemma** *complement-4-language*: *language* (*complement-3 A*) = *language* (*complement-4 A*)
⟨*proof*⟩

**lemma** *complement-4-finite*[*simp*]:
**assumes** *finite* (*nodes A*)
**shows** *finite* (*nodes* (*complement-4 A*))
⟨*proof*⟩
**lemma** *complement-4-correct*:
**assumes** *finite* (*nodes A*)
**shows** *language* (*complement-4 A*) = *streams* (*alphabet A*) − *language A*
⟨*proof*⟩

## 5.5 Phase 5

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

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

  lemma bounds-3-empty: bounds-3 A a Map.empty = Map.empty
    ⟨proof⟩
  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⟩

  lemma refresh-5-refine: (refresh-5, λ f. RETURN (refresh-1 f)) ∈ Id → ⟨Id⟩
nres-rel
  ⟨proof⟩
  lemma bounds-5-refine: (bounds-5 A a, λ f. RETURN (bounds-3 A a f)) ∈ Id
→ ⟨Id⟩ nres-rel
    ⟨proof⟩
  lemma items-5-refine: items-5 = items-4
    ⟨proof⟩
  lemma get-5-refine: get-5 = get-4
    ⟨proof⟩
  lemma expand-5-refine: (expand-5 f, ASSERT (finite (dom f)) ≫ RETURN
(expand-map f)) ∈ ⟨Id⟩ nres-rel
    ⟨proof⟩

  lemma complement-succ-5-refine: (complement-succ-5, RETURN ∘∘∘ comple-
ment-succ-4) ∈
    Id → Id → Id → ⟨Id⟩ nres-rel
    ⟨proof⟩
```

## 5.6 Phase 6

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

**lemma** *expand-map-get-6-refine*: (*expand-map-get-6*, *expand-5* ∘∘ *get-5*) ∈ *Id* → *Id* → ⟨*Id*⟩ *nres-rel*
⟨*proof*⟩

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

**lemma** *complement-succ-6-refine*:
(*complement-succ-6*, *complement-succ-5*) ∈ *Id* → *Id* → *Id* → ⟨*Id*⟩ *nres-rel*
⟨*proof*⟩

## 5.7 Phase 7

**interpretation** *autoref-syn* ⟨*proof*⟩

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

**private lemma** [*simp*]: *finite* (*dom f*)
⟨*proof*⟩

**schematic-goal** *refresh-7*: (*?f* :: *?'a*, *refresh-5 f*) ∈ *?R*
⟨*proof*⟩

**end**

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

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

⟨*proof*⟩

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

**private lemma** [*simp*]: *finite* (*transition A a p*)
⟨*proof*⟩ **lemma** [*simp*]: *finite* (*dom f*) ⟨*proof*⟩ **lemma** [*autoref-op-pat*]: *transition A a* ≡ *OP* (*transition A a*) ⟨*proof*⟩ **lemma** [*autoref-rules*]: (*min*, *min*) ∈ *nat-rel* → *nat-rel* → *nat-rel* ⟨*proof*⟩

**schematic-goal** *bounds-7*:
**notes** *ty-REL*[**where** *R* = ⟨*nat-rel*, *item-rel*⟩ *dflt-ahm-rel*, *autoref-tyrel*]
**shows** (*?f* :: *?'a*, *bounds-5 A a f*) ∈ *?R*
⟨*proof*⟩

**end**

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

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

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

**private lemma** [*autoref-op-pat*]: *accepting A* ≡ *OP* (*accepting A*) ⟨*proof*⟩ **lemma** [*autoref-rules*]: ((*dvd*), (*dvd*)) ∈ *nat-rel* → *nat-rel* → *bool-rel* ⟨*proof*⟩ **lemma** [*autoref-rules*]: (λ *k l*. *upt k* (*Suc l*), *atLeastAtMost*) ∈
*nat-rel* → *nat-rel* → ⟨*nat-rel*⟩ *list-set-rel*
⟨*proof*⟩

**schematic-goal** *items-7*: (*?f* :: *?'a*, *items-5 A*) ∈ *?R*
⟨*proof*⟩

**end**

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

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

**private lemma** $[simp]$: $finite\ (dom\ f)$
$\langle proof\rangle$ **lemma** $[simp]$:
**assumes** $\bigwedge\ m.\ m \in S \Longrightarrow x \notin dom\ m$
**shows** $inj\text{-}on\ (\lambda\ m.\ m\ (x \mapsto y))\ S$
$\langle proof\rangle$ **lemmas** $[simp] = op\text{-}map\text{-}update\text{-}def[abs\text{-}def]$

**private lemma** $[autoref\text{-}op\text{-}pat]$: $items\text{-}5\ A \equiv OP\ (items\text{-}5\ A)\ \langle proof\rangle$ **lemmas** $[autoref\text{-}rules] = items\text{-}7.refine[OF\ ai]$

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

**end**

**concrete-definition** $expand\text{-}map\text{-}get\text{-}7$ **uses** $expand\text{-}map\text{-}get\text{-}7$

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

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

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

**private lemma** $[autoref\text{-}op\text{-}pat]$: $(\lambda\ g.\ ASSUME\ (finite\ (dom\ g)) \gg ex\text{-}$

*pand-map-get-6 A g*) ≡
OP (λ *g*. *ASSUME* (*finite* (*dom g*)) ≫ *expand-map-get-6 A g*) ⟨*proof*⟩
**lemma** [*autoref-op-pat*]: *bounds-5 A a* ≡ *OP* (*bounds-5 A a*) ⟨*proof*⟩ **lemmas** [*autoref-rules*] =
*refresh-7-refine*
*bounds-7-refine*[*OF succi*]
*expand-map-get-7-refine*[*OF acceptingi*]

**schematic-goal** *complement-succ-7*: (*?f* :: *?'a*, *complement-succ-6 A a p*) ∈ *?R*
⟨*proof*⟩

**end**

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

**lemma** *complement-succ-7-refine*:
(*RETURN* ∘∘∘ *complement-succ-7*, *complement-succ-6*) ∈
⟨*Id*, *Id*⟩ *nbai-nba-rel* → *Id* → *state-rel* →
⟨⟨*state-rel*⟩ *list-set-rel*⟩ *nres-rel*
⟨*proof*⟩

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

**private lemma** [*autoref-op-pat*]: *initial A* ≡ *OP* (*initial A*) ⟨*proof*⟩ **lemmas** [*autoref-rules*] = *nbai-nba-param*(*3*)[*THEN fun-relD*, *OF Ai*]

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

**end**

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

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

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

**definition** *complement-7* :: (*'label*, *nat*) *nbai* ⇒ *nat* ⇒ (*'label*, *state*) *nbai* **where**

*complement-7 Ai ni* ≡ *nbai*
(*alphabeti Ai*)
(*complement-initial-7 Ai ni*)
(*complement-succ-7 Ai*)
(*complement-accepting-7*)

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

**end**

# 6 Boolean Formulae

**theory** *Formula*
**imports** *Main*
**begin**

**datatype** *'a formula* =
*False* |
*True* |
*Variable 'a* |
*Negation 'a formula* |
*Conjunction 'a formula 'a formula* |
*Disjunction 'a formula 'a formula*

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

**end**

# 7 Final Instantiation of Algorithms Related to Complementation

**theory** *Complementation-Final*
**imports**
*Complementation-Implement*

*Formula*
*Transition-Systems-and-Automata.NBA-Translate*
*Transition-Systems-and-Automata.NGBA-Algorithms*
*HOL−Library.Permutation*
**begin**

## 7.1 Syntax

**no-syntax** *-do-let* :: [*pttrn*, *'a*] ⇒ *do-bind* ((*2let - =/ -*) [*1000*, *13*] *13*)
**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*⟩

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

**lemma** *state-hash-bounded-hashcode*[*autoref-ga-rules*]: *is-bounded-hashcode state-rel*
(*gen-equals* (*Gen-Map.gen-ball* (*foldli* ∘ *list-map-to-list*)) (*list-map-lookup* (=))
(*prod-eq* (=) (⟷)))) *state-hash*
⟨*proof*⟩

## 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*
⟨*proof*⟩
**concrete-definition** *complement-impl* **uses** *complement-impl*

**theorem** *complement-impl-correct*:
**assumes** *finite* (*NBA.nodes A*)
**assumes** (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**shows** *NBA.language* (*nbae-nba* (*nbaei-nbae* (*complement-impl Ai*))) =
*streams* (*nba.alphabet A*) − *NBA.language A*
⟨*proof*⟩

## 7.4 Language Subset

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

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

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

## 7.5 Language Equality

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

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

**schematic-goal** *language-equal-impl*:
**assumes** [*simp*]: *finite* (*NBA.nodes A*)
**assumes** [*simp*]: *finite* (*NBA.nodes B*)
**assumes** [*simp*]: *nba.alphabet A* = *nba.alphabet B*
**assumes** [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**assumes** [*autoref-rules*]: (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**shows** (*?f* :: *?'c*, *NBA.language A* ⊆ *NBA.language B* ∧ *NBA.language B* ⊆ *NBA.language A*) ∈ *?R*
⟨*proof*⟩
**concrete-definition** *language-equal-impl* **uses** *language-equal-impl*
**lemma** *language-equal-impl-refine*[*autoref-rules*]:
**assumes** *SIDE-PRECOND* (*finite* (*NBA.nodes A*))
**assumes** *SIDE-PRECOND* (*finite* (*NBA.nodes B*))
**assumes** *SIDE-PRECOND* (*nba.alphabet A* = *nba.alphabet B*)
**assumes** (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**assumes** (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**shows** (*language-equal-impl Ai Bi*, (*OP op-language-equal* :::

$\langle Id,\ nat\text{-}rel\rangle\ nbai\text{-}nba\text{-}rel \to \langle Id,\ nat\text{-}rel\rangle\ nbai\text{-}nba\text{-}rel \to bool\text{-}rel)\ \$\ A\ \$\ B) \in bool\text{-}rel$
⟨*proof*⟩

**schematic-goal** *product-impl*:
**assumes** [*simp*]: *finite* (*NBA.nodes B*)
**assumes** [*autoref-rules*]: (*Ai*, *A*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**assumes** [*autoref-rules*]: (*Bi*, *B*) ∈ ⟨*Id*, *nat-rel*⟩ *nbai-nba-rel*
**shows** (*?f* :: *?'c*, *do* {
*let AB' = intersect A* (*complement-4 B*);
*ASSERT* (*finite* (*NBA.nodes AB'*));
*op-translate AB'*
}) ∈ *?R*
⟨*proof*⟩
**concrete-definition** *product-impl* **uses** *product-impl*

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

**end**

# References


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