Development of a decision procedure for finite automata in Isabelle/HOL

Stephan Merz

INRIA Nancy & LORIA

December 15, 2011

Contents

1. VeriDis Team
2. Non-Deterministic Finite Automata
3. The interactive Proof Assistant Isabelle
4. A Verified Implementation of the Decision Procedure

VeriDis Team

- Joint team between INRIA Nancy and MPI-INF Saarbrücken
  - local team at INRIA Nancy since 2010
  - joint proposal approved in summer 2011
  - Automation of Logic group at MPI-INF (Christoph Weidenbach)
- Formal verification techniques
  - tools for automated deduction: SMT (veriT), FOL (Spass)
  - integration of automatic and interactive tools
  - model checking, counter-models, verification platform (TLAPS)
- Methodology for development of distributed algorithms
  - refinement concepts, domain-specific models
  - extensions to probabilistic algorithms

Contents

1 VeriDis Team

2 Non-Deterministic Finite Automata

3 The interactive Proof Assistant Isabelle

4 A Verified Implementation of the Decision Procedure

Non-Deterministic Finite Automata (NFA)

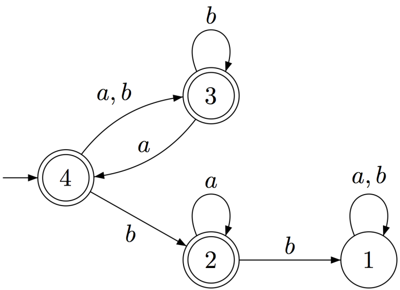
Definition (NFA)

An NFA $\mathcal{A} = (Q, q_0, \delta, F)$ over alphabet Σ is given by

- a finite set Q of states, an initial state $q_0 \in Q$,
- a transition relation $\delta \subseteq Q \times \Sigma \times Q$,
- a set $F \subseteq Q$ of accepting states.

A DFA is an NFA whose transition relation is functional.

Example:



Results and Applications of NFA

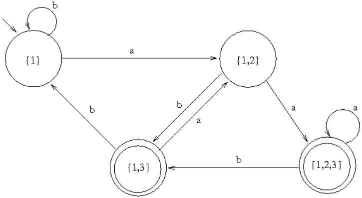
- NFA define regular languages
  - $w \in \Sigma^*$ accepted by $\mathcal{A}$ if $q_0 \stackrel{w}{\Longrightarrow} q_f$ for some $q_f \in F$
  - $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* : w \text{ accepted by } \mathcal{A}\}$
- Robust class of languages
  - closure properties: union, intersection, complement, projection, …
  - equivalence of NFA and DFA: subset construction
  - decision problems: emptiness, universality, inclusion, …
  - generalizations: ω-words, trees, …
- NFA are widely used in computer science
  - parsing: lexical analysis
  - transition systems, state diagrams, protocols
  - representation of properties

Deciding Emptiness of NFA

The Emptiness Problem

Given an NFA $\mathcal{A}$, determine if $\mathcal{L}(\mathcal{A}) = \emptyset$.

- Algorithm for deciding emptiness
  - determine if some state $q_f \in F$ can be reached from q_0
  - standard graph problem: enumerate reachable nodes
  - depth-first search from q_0, stop when hitting some $q_f \in F$
  - remember nodes already seen to avoid cycles
- Linear time complexity



Deciding Universality of NFA

The Universality Problem

Given an NFA $\mathcal{A}$, determine if $\mathcal{L}(\mathcal{A}) = \Sigma^*$.

- Algorithm for Deciding Universality
  - reduce to emptiness: $\mathcal{L}(\mathcal{A}) = \Sigma^*$ iff $\mathcal{L}(\overline{\mathcal{A}}) = \emptyset$

    where $\overline{\mathcal{A}}$ denotes the automaton for the complement
- How can we construct $\overline{\mathcal{A}}$?
  - easy to complement a DFA: exchange final and non-final states
  - first construct DFA, then complement and check for emptiness

    $\mathcal{A} \rightsquigarrow \mathcal{A}_d \rightsquigarrow \overline{\mathcal{A}_d} \rightsquigarrow$ DFS search

Deciding Universality of NFA

The Universality Problem

Given an NFA $\mathcal{A}$, determine if $\mathcal{L}(\mathcal{A}) = \Sigma^*$.

- Algorithm for Deciding Universality
  - reduce to emptiness: $\mathcal{L}(\mathcal{A}) = \Sigma^*$ iff $\mathcal{L}(\overline{\mathcal{A}}) = \emptyset$

    where $\overline{\mathcal{A}}$ denotes the automaton for the complement
- How can we construct $\overline{\mathcal{A}}$?
  - easy to complement a DFA: exchange final and non-final states
  - first construct DFA, then complement and check for emptiness

    $\mathcal{A} \rightsquigarrow \mathcal{A}_d \rightsquigarrow \overline{\mathcal{A}_d} \rightsquigarrow$ DFS search
- Exponential worst-case complexity: subset construction
  - unavoidable in general: PSPACE-complete problem
  - … but maybe one can do better in practice?

Improving the Decision Procedure for Universality

Two main insights

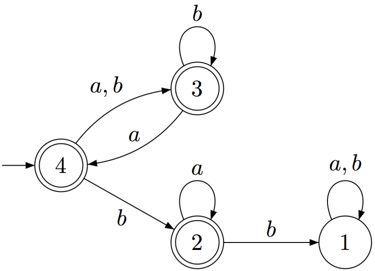
1. Keep only maximal sets in the subset construction
2. Interleave subset construction and search

1. Restrict to maximal sets
   - subset construction remembers which states may be reached
   - if one may reach S and $T \supseteq S$, no need to remember S
2. Interleave determinization, complementation, and search
   - no need to construct $\overline{\mathcal{A}_d}$: it's enough to prove non-emptiness

M. De Wulf, L. Doyen, T.A. Henzinger, J.F. Raskin: *Antichains – A New Algorithm for Checking Universality of Finite Automata.* CAV 2006, LNCS 4144, pp.17-30, 2006.

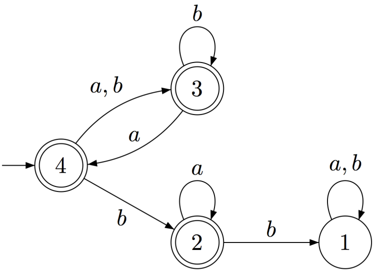
Backward Algorithm for Universality (1)

- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example



Backward Algorithm for Universality (1)

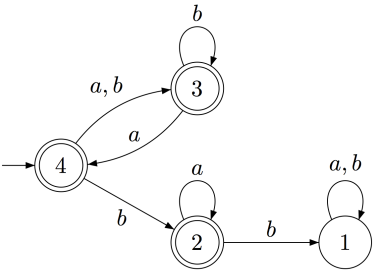
- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example



$$\mathcal{S}_0 = \{\{1\}\}$$

Backward Algorithm for Universality (1)

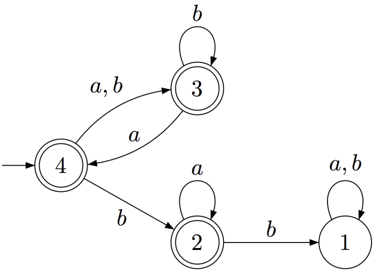
- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example



$$\mathcal{S}_0 = \{\{1\}\}$$
$$\mathcal{S}_1 = \{\{1\}, \{1, 2\}\}$$

Backward Algorithm for Universality (1)

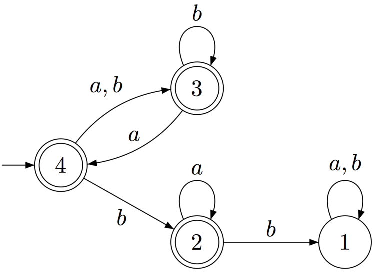
- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example



$$\mathcal{S}_0 = \{\{1\}\}$$
$$\mathcal{S}_1 = \{\{1\}, \{1, 2\}\}$$
$$\mathcal{S}_2 = \{\{1\}, \{2\}, \{1, 2\}\}$$

Backward Algorithm for Universality (1)

- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example

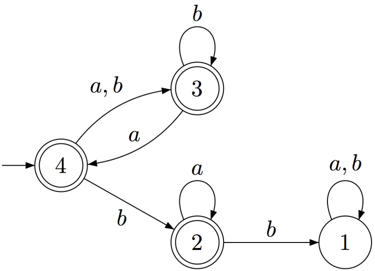


$$\mathcal{S}_0 = \{\{1\}\}$$
$$\mathcal{S}_1 = \{\{1\},\{1,2\}\}$$
$$\mathcal{S}_2 = \{\{1\},\{2\},\{1,2\}\}$$
$$\mathcal{S}_3 = \{\{1\},\{2\},\{1,2\}\}$$

- fixpoint reached
- no set contains initial state 4
- automaton is universal

Backward Algorithm for Universality (1)

- Intuitive Idea
  - track sets of states from which non-final states cannot be avoided
  - initially: $\mathcal{S} = \{Q \setminus F\}$ [non-final states]
  - update: for $S \in \mathcal{S}$, add sets of states all of whose successors are in S (for some $a \in \Sigma$)
  - stop: fixpoint reached or $q_0 \in S$ for some $S \in \mathcal{S}$
- Example



optimization

drop non-maximal sets

$$\mathcal{S}_0 = \{\{1\}\}$$
$$\mathcal{S}_1 = \{\{1\}, \{1,2\}\}$$
$$\mathcal{S}_2 = \{\{1\}, \{2\}, \{1,2\}\}$$
$$\mathcal{S}_3 = \{\{1\}, \{2\}, \{1,2\}\}$$

- fixpoint reached
- no set contains initial state 4
- automaton is universal

Backward Algorithm for Universality (2)

$$
\begin{array}{l}
\mathcal{S}_{old} := \emptyset \\
\mathcal{S}_{new} := \{Q \setminus F\} \\
\textbf{while } \mathcal{S}_{new} \neq \mathcal{S}_{old} \wedge q_0 \not\sqsubseteq \mathcal{S}_{new} \\
\textbf{do} \quad \mathcal{S}_{old} := \mathcal{S}_{new} \\
\qquad\;\; \mathcal{S}_{new} := \lceil \mathcal{S}_{new} \cup CPre(\mathcal{S}_{new}) \rceil \\
\textbf{end}
\end{array}
$$

where:

$$
\begin{array}{rcl}
q \sqsubseteq \mathcal{S} & \text{iff} & q \in S \text{ for some } S \in \mathcal{S} \\
cpre(S, a) & \triangleq & \{q \in Q : \delta(q, a) \subseteq S\} \\
CPre(\mathcal{S}) & \triangleq & \{cpre(S, a) : S \in \mathcal{S}, a \in \Sigma\} \\
\lceil \mathcal{S} \rceil & \triangleq & \{S \in \mathcal{S} : \neg\exists T \in \mathcal{S} : S \subset T\}
\end{array}
$$

Correctness Proof (Idea)

Soundness

At the end of algorithm, $q_0 \mathsf{E} \mathcal{S}_{new}$ holds iff $\mathcal{A}$ is not universal.

Proof (idea). Let $\mathcal{S}_i$ denote the value of $\mathcal{S}_{new}$ at the i-th iteration.

1. If $S \in \mathcal{S}_i$ then there is $w \in \Sigma^*$ such that for all $q \in S$, whenever $q \stackrel{w}{\Longrightarrow} q'$ then $q' \in Q \setminus F$.
2. If $S \subseteq Q$ and $w \in \Sigma^k$ such that for all $q \in S$, whenever $q \stackrel{w}{\Longrightarrow} q'$ then $q' \in Q \setminus F$, then $S \subseteq S'$ for some $S' \in \mathcal{S}_k$.

The theorem follows easily from these two lemmas. Q.E.D.

Moreover, the algorithm is certain to terminate.

- sets $\mathcal{S}_i$ increasing: for all $S \in \mathcal{S}_i$ exists $T \in \mathcal{S}_{i+1}$ s.t. $S \subseteq T$
- every $\mathcal{S}_i$ is bounded from above by (the maximal subsets of) 2^Q

Contents

1 VeriDis Team

2 Non-Deterministic Finite Automata

3 The interactive Proof Assistant Isabelle

4 A Verified Implementation of the Decision Procedure

Isabelle foundations

- Logical framework ("meta level")
  - Paulson & Nipkow, since 1986
  - goal: rapid prototyping of deductive systems
  - encode syntax and deduction rules of logics

- Minimal higher-order logic with equality, types à la ML

types	$prop$	propositions
	$\alpha \Rightarrow \beta$	function type
operators	$\Longrightarrow\ :: [prop, prop] \Rightarrow prop$	implication
	$\equiv\ :: [\alpha, \alpha] \Rightarrow prop$	equality
	$\bigwedge\ :: (\alpha \Rightarrow prop) \Rightarrow prop$	universal quantification

  - primitive deduction rules for these operators
  - small trusted kernel certifies reasoning

Isabelle/HOL: basis

Principal object logic, inspired by Gordon's HOL system

type *bool*

consts

$$
\begin{array}{lll}
\_ & :: & bool \Rightarrow prop \\
= & :: & [\alpha, \alpha] \Rightarrow bool \\
\to & :: & [bool, bool] \Rightarrow bool \\
\forall & :: & (\alpha \Rightarrow bool) \Rightarrow bool
\end{array}
$$

lifting to propositions

HOL connectives

axioms

$$
\begin{array}{ll}
\text{impI}: & (A \Longrightarrow B) \Longrightarrow A \to B \\
\text{mp}: & A \Longrightarrow (A \to B) \Longrightarrow B
\end{array}
$$

definitions

$$
\begin{array}{lll}
False & \equiv & \forall P.\, P \\
\neg A & \equiv & A \to False
\end{array}
$$

Isabelle/HOL: features

- Tools, library, automation
  - (co-)inductive definitions, algebraic data types, extensible records
  - module system (*locales*)
  - structured proof language (Isar)
  - large mathematical library: main workhorse for applications
  - built-in proof tools: logic, set theory, equality reasoning, ...
  - external reasoners: SAT, SMT, first-order provers
  - code generation from executable definitions
- Sample applications
  - cryptographic protocols (Paulson)
  - Java semantics, virtual machine, and compiler (Nipkow et al.)
  - SEL4: verified micro-kernel (Klein et al.)
  - Archive of Formal Proofs: `http://afp.sf.net/`

Formalizing NFA in Isabelle

- Representation by a record and a well-formedness predicate

record $(\rho, \alpha) nfa =$

$states :: \rho\ set$

$alpha :: \alpha\ set$

$init :: \rho\ set$

$final :: \rho\ set$

$trans :: [\rho, \alpha, \rho] \Rightarrow bool$

definition *wf_nfa* **where**

$$
\begin{aligned}
wf\_nfa\ auto\ \equiv\ & \\
& finite\ (states\ auto) \\
& \wedge\ init\ auto \subseteq states\ auto \\
& \wedge\ final\ auto \subseteq states\ auto \\
& \wedge\ \forall q \in states\ auto.\ \forall a \in alpha\ auto.\ trans\ auto\ q\ a \subseteq states\ auto
\end{aligned}
$$

Elementary Definitions on NFA

- Iterated transition relation

$$\textbf{fun}\ \mathit{transit} :: [(\rho,\alpha)\mathit{nfa}, \rho, \alpha\ \mathit{list}] \Rightarrow \rho\ \mathit{set}\ \textbf{where}$$
$$\mathit{transit}\ \mathit{auto}\ q\ [] = \{q\}$$
$$|\ \mathit{transit}\ \mathit{auto}\ q\ (a \# w) = \bigcup_{q' \in \mathit{trans}\ \mathit{auto}\ q\ a} \mathit{transit}\ \mathit{auto}\ q'\ w$$

- Language accepted by NFA

$$\textbf{definition}\ \mathit{words} :: \alpha\ \mathit{set} \Rightarrow \alpha\ \mathit{list}\ \mathit{set}\ \textbf{where}$$
$$\mathit{words}\ \mathit{alph} \equiv \{w.\ \mathit{set}\ w \subseteq \mathit{alph}\}$$
$$\textbf{definition}\ \mathit{language} :: (\rho,\alpha)\mathit{nfa} \Rightarrow' a\ \mathit{list}\ \mathit{set}\ \textbf{where}$$
$$\mathit{language}\ \mathit{auto} \equiv \{w \in \mathit{words}\ (\mathit{alpha}\ \mathit{auto}).\ \exists q \in \mathit{init}\ \mathit{auto}.$$
$$\mathit{transit}\ \mathit{auto}\ q\ w \cap \mathit{final}\ \mathit{auto} \neq \{\}\}$$

- Universality of NFA

$$\textbf{definition}\ \mathit{universal} :: (\rho,\alpha)\mathit{nfa} \Rightarrow \mathit{bool}\ \textbf{where}$$
$$\mathit{universal}\ \mathit{auto} \equiv \mathit{language}\ \mathit{auto} = \mathit{words}(\mathit{alpha}\ \mathit{auto})$$

Abstract Algorithm for Deciding Universality

- Auxiliary definitions

definition $maxi :: \rho\ set\ set \Rightarrow \rho\ set\ set$ **where**
$maxi\ s \equiv \{v \in s.\ \forall v' \in s.\ v \subseteq v' \to v' \subseteq v\}$
definition $cpre :: [(\rho, \alpha)nfa, \rho\ set, \alpha] \Rightarrow \rho\ set$ **where**
$cpre\ auto\ s\ a \equiv \{q \in states\ auto.\ trans\ auto\ q\ a \subseteq s\}$
definition $CPre :: [(\rho, \alpha)nfa, \rho\ set\ set] \Rightarrow \rho\ set\ set$ **where**
$CPre\ auto\ S \equiv \{cpre\ auto\ s\ a | s\ a.\ s \in S \wedge a \in alpha\ auto\}$

- Core of decision procedure: sequence of sets $\mathcal{S}_i$

fun $Bhat :: [(\rho, \alpha)nfa, nat] \Rightarrow \rho\ set\ set$ **where**
$Bhat\ auto\ 0 = \{states\ auto - final\ auto\}$
$|\ Bhat\ auto\ (Suc\ k) = maxi(Bhat\ auto\ k \cup CPre\ auto\ (Bhat\ auto\ k))$

Correctness Proof (1)

- Elementary lemmas
  - *maxi s* subset of s, contains only maximal sets: trivial
  - for any $v \in s$ there is some $v' \in maxi\ s$ such that $v \subseteq v'$: tedious
    [induction over finite sets, many case distinctions]
  - all elements of *Bhat auto k* are subsets of *states auto* : easy
  - sets *Bhat auto k* are "increasing": straightforward inductive proof

Correctness Proof (1)

- Elementary lemmas
  - *maxi s* subset of *s*, contains only maximal sets: trivial
  - for any $v \in s$ there is some $v' \in \mathit{maxi}\ s$ such that $v \subseteq v'$: tedious
    [induction over finite sets, many case distinctions]
  - all elements of *Bhat auto k* are subsets of *states auto* : easy
  - sets *Bhat auto k* are "increasing": straightforward inductive proof
- Main correctness lemmas

lemma *sound* :
  assumes *wf_nfa auto* **and** $E \in \mathit{Bhat}\ \mathit{auto}\ i$
  shows $\exists w \in \mathit{words}\ (\mathit{alpha}\ \mathit{auto}).\ \forall q \in E.$
    $\mathit{transit}\ \mathit{auto}\ q\ w \subseteq \mathit{states}\ \mathit{auto} - \mathit{final}\ \mathit{auto}$

lemma *complete* :
  assumes *wf_nfa auto* **and** $E \subseteq \mathit{states}\ \mathit{auto}$ **and** $w \in \mathit{words}\ (\mathit{alpha}\ \mathit{auto})$
  and $\forall q \in E.\ \mathit{transit}\ \mathit{auto}\ q\ w \subseteq \mathit{states}\ \mathit{auto} - \mathit{final}\ \mathit{auto}$
  shows $\exists E' \in \mathit{Bhat}\ \mathit{auto}\ (\mathit{length}\ w).\ E \subseteq E'$

Correctness Proof (2)

- Final theorem

theorem *univ* :
  assumes *wf_nfa auto*
  shows *universal auto* $= (\forall i.\ \forall E \in \mathit{Bhat\ auto\ i}.\ \neg(\mathit{init\ auto} \subseteq E))$

- Observations
  - formalization close to ordinary mathematical notation
  - proofs closely follow “paper proof”
  - reasonable proof size: about 40 lines per correctness lemma
  - existence of maximal sets surprisingly difficult to prove

Correctness Proof (2)

- Final theorem

theorem *univ* :
  assumes *wf_nfa auto*
  shows $universal\ auto = (\forall i.\ \forall E \in Bhat\ auto\ i.\ \neg(init\ auto \subseteq E))$

- Observations
  - formalization close to ordinary mathematical notation
  - proofs closely follow “paper proof”
  - reasonable proof size: about 40 lines per correctness lemma
  - existence of maximal sets surprisingly difficult to prove

- Have we gained anything in formalizing this construction?

Contents

1. VeriDis Team
2. Non-Deterministic Finite Automata
3. The interactive Proof Assistant Isabelle
4. A Verified Implementation of the Decision Procedure

Towards an Executable Program

- Lessons learnt so far
  - Isabelle/HOL: adequate for formalizing automata theory
  - formalization of proof may increase our understanding …
  - … but really we knew that it was correct
- Added value: verified implementation
  - generate executable code from the formalization
  - common basis for proof and for execution
  - safeguard against implementation errors
  - use by itself or to evaluate hand-written code
- Problems for code generation
  - high level of abstraction: use of set comprehensions etc.
  - definition of (infinite) sequence rather than actual algorithm

Executable Set Constructions

- Isabelle Collection Framework [Lammich 2009]
  - formalize standard data structures, including sets and maps
  - association lists, hash sets, red-black trees, finger trees, . . .
  - abstract interface provides basic set operations
  - implementations: abstraction mapping and correctness lemmas

- Separation of algorithm and data structure
  - express algorithm in terms of abstract interface
  - instantiate desired implementation of data structure

Executable Version of NFA

- Record definition and abstraction mapping

$$
\begin{array}{l}
\textbf{record } (\rho, \alpha)\mathit{exec\_nfa} = \\
\quad \mathit{states\_ls} :: \rho\ \mathit{exec\_set} \\
\quad \mathit{alpha\_ls} :: \alpha\ \mathit{exec\_set} \\
\quad \mathit{init\_ls} :: \rho\ \mathit{exec\_set} \\
\quad \mathit{final\_ls} :: \rho\ \mathit{exec\_set} \\
\quad \mathit{trans\_ls} :: [\rho, \alpha] \Rightarrow \rho\ \mathit{exec\_set} \\
\textbf{definition } \alpha_{auto} :: (\rho, \alpha)\mathit{exec\_nfa} \Rightarrow (\rho, \alpha)\mathit{nfa}\ \textbf{where} \\
\quad \alpha_{auto}\ \mathit{auto} \ \equiv \ (\!|\ \mathit{states} = \alpha_{set}\ (\mathit{states\_ls}\ \mathit{auto}), \\
\qquad\qquad\qquad\quad \mathit{alpha} = \alpha_{set}\ (\mathit{alpha\_ls}\ \mathit{auto}), \\
\qquad\qquad\qquad\quad \mathit{init} = \alpha_{set}\ (\mathit{init\_ls}\ \mathit{auto}), \\
\qquad\qquad\qquad\quad \mathit{final} = \alpha_{set}\ (\mathit{final\_ls}\ \mathit{auto}), \\
\qquad\qquad\qquad\quad \mathit{trans} = (\lambda q\ a.\ \alpha_{set}\ (\mathit{trans\_ls}\ \mathit{auto}\ q\ a))\ |\!)
\end{array}
$$

- Write algorithm for *exec_nfa*, state correctness in terms of α_{auto}

Basic Operations

- Compute predecessors: explicit iteration

$$\textbf{definition}\ \mathit{exec\_cpre} :: [(\rho,\alpha)\mathit{exec\_nfa}, \rho\ \mathit{exec\_set}, \alpha] \Rightarrow \rho\ \mathit{exec\_set}\ \textbf{where}$$
$$\mathit{exec\_cpre}\ \mathit{auto}\ S\ a \equiv \mathit{set\_iterate}\ (\lambda\ q\ \mathit{pre}.\ \mathit{if}\ (\mathit{exec\_subset}\ (\mathit{trans\_ls}\ \mathit{auto}\ q\ a)\ S)\ \mathit{then}\ \mathit{exec\_ins}\ q\ \mathit{pre}\ \mathit{else}\ \mathit{pre})\ (\mathit{states\_ls}\ \mathit{auto})\ \mathit{exec\_empty}$$
$$\textbf{lemma}\ \mathit{cpre\_corres} : \alpha_{set}\ (\mathit{exec\_cpre}\ \mathit{auto}\ S\ a) = \mathit{cpre}\ (\alpha_{auto}\ \mathit{auto})\ (\alpha_{set}\ S)\ a$$

- Similar definitions for *CPre* and *maxi*

Implementation of Decision Procedure

- Procedure operates on triples $(\mathcal{S}_{old}, \mathcal{S}_{new}, k)$

definition *Bhat_init* **where**
Bhat_init auto $\equiv$
  (*exec_empty*,
   exec_ins (*exec_diff* (*states_ls auto*) (*final_ls auto*)) *exec_empty*,
   0)
definition *Bhat_step* **where**
Bhat_step auto s $\equiv$
  (*fst* (*snd s*),
   exec_maxi (*exec_union* (*fst* (*snd s*)) (*exec_CPre auto* (*fst* (*snd s*)))),
   Suc (*snd* (*snd s*)))
definition *while_Bhat* **where**
while_Bhat auto $\equiv$
  fst (*while* (λ(*old*, *new*, *cnt*). ($\neg$*exec_equal_set old new*))
      (*Bhat_step auto*)
      (*Bhat_init auto*))

Correctness Proof

- Implementation mirrors abstract computation

lemma *Bhat_correctness* :
  assumes *wf_exec_nfa auto*
  shows $\forall k.\ \forall E \in Bhat\ (\alpha_{auto}\ auto)\ k.\ \exists E' \in \alpha_{set}\ (while\_Bhat\ auto).\ E \subseteq E'$
    $\wedge\ \exists k.\ \alpha_{set}\ (while\_Bhat\ auto) = Bhat\ (\alpha_{auto}\ auto)\ k$
definition *universal_impl* **where**
  $universal\_impl\ auto\ \equiv$
    $\neg(exec\_bex\ (while\_Bhat\ auto)\ (\lambda E.\ exec\_subset\ (init\_ls\ auto)\ E))$
theorem *impl_correct* :
  assumes *wf_exec_nfa auto*
  shows $universal\ (\alpha_{auto}\ auto) = universal\_impl\ auto$

- Elements of correctness proof ($\sim$ 200 lines, unfinished)
  - linking invariant connecting abstract and concrete algorithms
  - well-foundedness of step relation for proving termination

Conclusion

- Development of formally verified decision procedure
  - proof assistants mature enough: reasonable effort
  - correctness proof generic w.r.t. underlying data structure
  - verified reference implementation
- Proof and executable code from same source
  - eliminate coding errors in implementation
  - need to trust proof checker and code generator
- What remains to be done
  - finish proof (mostly straightforward)
  - substitute efficient data structure