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

Stephan Merz

INRIA Nancy & LORIA

December 15, 2011
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** $A = (Q, q_0, \delta, F)$ over alphabet $\Sigma$ 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:**

![Diagram of an NFA]
Results and Applications of NFA

- NFA define regular languages
  - $w \in \Sigma^*$ accepted by $A$ if $q_0 \xrightarrow{w} q_f$ for some $q_f \in F$
  - $L(A) = \{w \in \Sigma^* : w \text{ accepted by } A\}$

- Robust class of languages
  - closure properties: union, intersection, complement, projection, ...
  - equivalence of NFA and DFA: subset construction
  - decision problems: emptiness, universality, inclusion, ...
  - generalizations: $\omega$-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 $A$, determine if $L(A) = \emptyset$.

- **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 $A$, determine if $L(A) = \Sigma^*$.

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

- How can we construct $\overline{A}$?
  - easy to complement a DFA: exchange final and non-final states
  - first construct DFA, then complement and check for emptiness
    $A \leadsto A_d \leadsto \overline{A_d} \leadsto$ DFS search

Exponential worst-case complexity: subset construction
  - unavoidable in general: PSPACE-complete problem
  - ... but maybe one can do better in practice?
Deciding Universality of NFA

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

- Algorithm for Deciding Universality
  - reduce to emptiness: $L(\mathcal{A}) = \Sigma^*$ iff $L(\overline{\mathcal{A}}) = \emptyset$
    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
   - subset construction remembers which states may be reached
   - if one may reach \( S \) and \( T \supseteq S \), no need to remember \( S \)

2. Interleave subset construction and search
   - no need to construct \( \overline{A_d} \) : it’s enough to prove non-emptiness

Backward Algorithm for Universality (1)

- **Intuitive Idea**
  - track sets of states from which non-final states cannot be avoided
  - initially: $S = \{ Q \setminus F \}$ [non-final states]
  - update: for $S \in S$, 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 $S \in S$

- **Example**

![Diagram of a finite automaton](image-url)
Intuitive Idea

- track sets of states from which non-final states cannot be avoided
- initially: $S = \{ Q \setminus F \}$ [non-final states]
- update: for $S \in S$, 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 $S \in S$

Example

$S_0 = \{ \{1\} \}$
Intuitive Idea

- track sets of states from which non-final states cannot be avoided
- initially: \( S = \{Q \setminus F\} \) [non-final states]
- update: for \( S \in S \), 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 \( S \in S \)

Example

\[
\begin{align*}
S_0 &= \{\{1\}\} \\
S_1 &= \{\{1\}, \{1, 2\}\}
\end{align*}
\]
Backward Algorithm for Universality (1)

- **Intuitive Idea**
  - track sets of states from which non-final states cannot be avoided
  - initially: $S = \{ Q \setminus F \}$ [non-final states]
  - update: for $S \in S$, 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 $S \in S$

- **Example**

\[
S_0 = \{ \{1\} \} \\
S_1 = \{ \{1\}, \{1, 2\} \} \\
S_2 = \{ \{1\}, \{2\}, \{1, 2\} \}
\]
Intuitive Idea

- track sets of states from which non-final states cannot be avoided
- initially: $S = \{Q \setminus F\}$ [non-final states]
- update: for $S \in S$, 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 $S \in S$

Example

$S_0 = \{\{1\}\}$
$S_1 = \{\{1\}, \{1, 2\}\}$
$S_2 = \{\{1\}, \{2\}, \{1, 2\}\}$
$S_3 = \{\{1\}, \{2\}, \{1, 2\}\}$

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

- track sets of states from which non-final states cannot be avoided
- initially: $S = \{Q \setminus F\}$ [non-final states]
- update: for $S \in S$, 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 $S \in S$

Example

- $S_0 = \{\{1\}\}$
- $S_1 = \{\{1\}, \{1, 2\}\}$
- $S_2 = \{\{1\}, \{2\}, \{1, 2\}\}$
- $S_3 = \{\{1\}, \{2\}, \{1, 2\}\}$

- fixpoint reached
- no set contains initial state 4
- automaton is universal
Backward Algorithm for Universality (2)

\[ S_{\text{old}} := \emptyset \]
\[ S_{\text{new}} := \{ Q \setminus F \} \]

\textbf{while} \( S_{\text{new}} \neq S_{\text{old}} \land q_0 \not\in S_{\text{new}} \)
\textbf{do} \quad \begin{align*}
S_{\text{old}} &:= S_{\text{new}} \\
S_{\text{new}} &:= \left[ S_{\text{new}} \cup CPre(S_{\text{new}}) \right]
\end{align*}
\textbf{end}

where:
\[ q \in S \quad \text{iff} \quad q \in S \text{ for some } S \in S \]
\[ cpre(S, a) \triangleq \{ q \in Q : \delta(q, a) \subseteq S \} \]
\[ CPre(S) \triangleq \{ cpre(S, a) : S \in S, a \in \Sigma \} \]
\[ \lceil S \rceil \triangleq \{ S \in S : \neg \exists T \in S : S \subset T \} \]
Correctness Proof (Idea)

Soundness
At the end of algorithm, \( q_0 \in S_{new} \) holds iff \( A \) is not universal.

Proof (idea). Let \( S_i \) denote the value of \( S_{new} \) at the \( i \)-th iteration.

1. If \( S \in S_i \) then there is \( w \in \Sigma^* \) such that for all \( q \in S \), whenever \( q \xrightarrow{w} q' \) then \( q' \in Q \setminus F \).

2. If \( S \subseteq Q \) and \( w \in \Sigma^k \) such that for all \( q \in S \), whenever \( q \xrightarrow{w} q' \) then \( q' \in Q \setminus F \), then \( S \subseteq S' \) for some \( S' \in S_k \).

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

Moreover, the algorithm is certain to terminate.

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

<table>
<thead>
<tr>
<th>types</th>
<th>prop</th>
<th>propositions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha \Rightarrow \beta$</td>
<td>function type</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operators</th>
<th>$\implies ::= [prop, prop] \Rightarrow prop$</th>
<th>implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\equiv ::= [\alpha, \alpha] \Rightarrow prop$</td>
<td>equality</td>
<td></td>
</tr>
<tr>
<td>$\land ::= (\alpha \Rightarrow prop) \Rightarrow prop$</td>
<td>universal quantification</td>
<td></td>
</tr>
</tbody>
</table>

- 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**

- :: bool ⇒ prop  lifting to propositions
= :: [α, α] ⇒ bool  HOL connectives
→ :: [bool, bool] ⇒ bool
∀ :: (α ⇒ bool) ⇒ bool

**axioms**

impI : (A ⇒ B) ⇒ A → B
mp : A ⇒ (A → B) ⇒ B

**definitions**

False ≡ ∀P. P
¬A ≡ A → False
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.)
Formalizing NFA in Isabelle

- Representation by a record and a well-formedness predicate

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

\[
\begin{align*}
\text{states} &:: \rho \text{ set} \\
\text{alpha} &:: \alpha \text{ set} \\
\text{init} &:: \rho \text{ set} \\
\text{final} &:: \rho \text{ set} \\
\text{trans} &:: [\rho, \alpha, \rho] \Rightarrow \text{bool}
\end{align*}
\]

definition \(wf\ nfa\) where

\[
\begin{align*}
wf\ nfa\ auto &\equiv \\
&\text{finite (states auto)} \\
&\land \text{init auto }\subseteq \text{states auto} \\
&\land \text{final auto }\subseteq \text{states auto} \\
&\land \forall q \in \text{states auto}. \forall a \in \text{alpha auto}. \text{trans auto} \ q \ a \subseteq \text{states auto}
\end{align*}
\]
Elementary Definitions on NFA

- **Iterated transition relation**

  \[
  \text{fun transit :: } (\rho, \alpha)\text{nfa, } \rho, \alpha\text{ list} \Rightarrow \rho\text{ set where}
  \]

  \[
  \begin{align*}
  \text{transit auto q } [] &= \{q\} \\
  \text{transit auto q } (a \# w) &= \bigcup_{q' \in \text{trans auto q } a} \text{transit auto q' } w
  \end{align*}
  \]

- **Language accepted by NFA**

  \[
  \text{definition words :: } \alpha\text{ set } \Rightarrow \alpha\text{ list set where}
  \]

  \[
  \text{words alph } \equiv \{w. \text{ set } w \subseteq \text{alph}\}
  \]

  \[
  \text{definition language :: } (\rho, \alpha)\text{nfa } \Rightarrow' \text{ a list set where}
  \]

  \[
  \text{language auto } \equiv \{w \in \text{words (alpha auto)}. \exists q \in \text{init auto}. \text{transit auto q } w \cap \text{final auto } \neq \{}\}
  \]

- **Universality of NFA**

  \[
  \text{definition universal :: } (\rho, \alpha)\text{nfa } \Rightarrow \text{ bool where}
  \]

  \[
  \text{universal auto } \equiv \text{language auto } = \text{words(alpha auto)}
  \]
Abstract Algorithm for Deciding Universality

- **Auxiliary definitions**

**definition maxi :: \( \rho \) set set \( \Rightarrow \) \( \rho \) set set where**

\[ \text{maxi } s \equiv \{ v \in s. \forall v' \in s. v \subseteq v' \rightarrow v' \subseteq v \} \]

**definition cpre :: \([ \rho, \alpha )nfa, \rho \) set, \alpha ] \( \Rightarrow \) \( \rho \) set where**

\[ \text{cpre } auto \ s \ a \equiv \{ q \in \text{states auto. trans auto } q \ a \subseteq s \} \]

**definition CPre :: \([ \rho, \alpha )nfa, \rho \) set set \] \( \Rightarrow \) \( \rho \) set set where**

\[ \text{CPre } auto \ S \equiv \{ \text{cpre } auto \ s \ a | s \ a. s \in S \land a \in \text{alpha auto} \} \]

- **Core of decision procedure: sequence of sets \( S_i \)**

**fun Bhat :: \([ \rho, \alpha )nfa, nat] \( \Rightarrow \) \( \rho \) set set where**

\[ \text{Bhat } auto \ 0 = \{ \text{states auto } - \text{final auto} \} \]

\[ | \text{Bhat } auto (\text{Suc } k) = \text{maxi}(\text{Bhat } auto \ k \cup \text{CPre } auto (\text{Bhat } auto \ k)) \]
Correctness Proof (1)

- **Elementary lemmas**
  - $\text{maxi } s$ subset of $s$, contains only maximal sets: trivial
  - for any $v \in s$ there is some $v' \in \text{maxi } s$ such that $v \subseteq v'$: tedious
    [induction over finite sets, many case distinctions]
  - all elements of $\text{Bhat auto } k$ are subsets of $\text{states auto}$: easy
  - sets $\text{Bhat auto } k$ are "increasing": straightforward inductive proof
Correctness Proof (1)

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

- **Main correctness lemmas**

```text
lemma sound :
  assumes \( \text{wf \_nfa auto and } E \in \text{Bhat auto } i \)
  shows \( \exists w \in \text{words (alpha auto). } \forall q \in E. \)
    \( \text{transit auto } q \ w \subseteq \text{states auto } - \text{final auto} \)

lemma complete :
  assumes \( \text{wf \_nfa auto and } E \subseteq \text{states auto and } w \in \text{words (alpha auto)} \)
  and \( \forall q \in E. \text{transit auto } q \ w \subseteq \text{states auto } - \text{final auto} \)
  shows \( \exists E' \in \text{Bhat auto (length w). } E \subseteq E' \)
```
Correctness Proof (2)

- **Final theorem**

  ```isar
theorem univ :
  assumes wfnfa auto
  shows universal auto = (∀i. ∀E ∈ Bhat auto i. ¬(init auto ⊆ 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?**
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

```plaintext
record (ρ, α)exec_nfa =
  states_ls :: ρ exec_set
  alpha_ls :: α exec_set
  init_ls :: ρ exec_set
  final_ls :: ρ exec_set
  trans_ls :: [ρ, α] ⇒ ρ exec_set

definition α_auto :: (ρ, α)exec_nfa ⇒ (ρ, α)nfa where
  α_auto auto ≡ (| states = α_set (states_ls auto),
                   alpha = α_set (alpha_ls auto),
                   init = α_set (init_ls auto),
                   final = α_set (final_ls auto),
                   trans = (λq a. α_set (trans_ls auto q a)) |)
```

- Write algorithm for `exec_nfa`, state correctness in terms of `α_auto`
Basic Operations

- **Compute predecessors: explicit iteration**

  \[
  \text{definition } \text{exec\_cpre} :: \{(\rho, \alpha)\text{exec\_nfa}, \rho \text{ exec\_set}, \alpha\} \Rightarrow \rho \text{ exec\_set} \text{ where}
  \]

  \[
  \text{exec\_cpre auto } S \ a \equiv \text{set\_iterate} (\lambda q \text{ pre. if } (\text{exec\_subset} (\text{trans\_ls auto q a}) \ S) \\
  \text{then exec\_ins q pre else pre}) (\text{states\_ls auto}) \text{ exec\_empty}
  \]

  \[
  \text{lemma } \text{cpre\_corres} : \alpha_{\text{set}} (\text{exec\_cpre auto } S \ a) = \text{cpre} (\alpha_{\text{auto auto}}) (\alpha_{\text{set}} S) \ a
  \]

- **Similar definitions for CPRe and maxi**
Implementation of Decision Procedure

- Procedure operates on triples \((S_{old}, S_{new}, k)\)

```plaintext
definition Bhat_init where
Bhat_init auto ≡
  (exec_empty,
   exec_ins (exec_diff (states_ls auto) (final_ls auto)) exec_empty,
   0)
definition Bhat_step where
Bhat_step auto s ≡
  (fst (snds),
   exec_maxi (exec_union (fst (snd s)) (exec_CPre auto (fst (snd s))))),
   Suc (snd (snd s)))
definition while_Bhat where
while_Bhat auto ≡
  fst (while (λ(old, new, cnt). (~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'
                   \land \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 \text{universal} (\alpha_{auto} auto) = universal_impl auto
```

- Elements of correctness proof (~ 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