Incremental Minimization Of Symbolic Automata

Jonathan Homburg (UCONN)
Parasara Sridhar Duggirala (UNC)
What Is Symbolic Automata?

- DFA's on steroids (very large alphabet, possibly infinite)

How to represent transitions? Use predicates

\((A, Q, q_0, F, \Delta)\)
Would Any Predicates Work?

\( \text{No; The predicates should form an effective Boolean Algebra} \)

\[ A = (D, \Psi, \forall \neg, \bot, \top, \land, \lor, \exists) \]

- \( \Psi \) is closed under Boolean Ops.
- \( \Psi : \Psi \rightarrow 2^D \)
- \( \llbracket \bot \rrbracket = \emptyset; \llbracket \top \rrbracket = D \)
- \( \llbracket \land \rrbracket = \llbracket \lor \rrbracket = \llbracket \cup \rrbracket \)
- \( \llbracket \exists \rrbracket = D \setminus \llbracket \forall \rrbracket \)
Are SA, Just DFA Over Predicates?

- Yes & NO
  - The predicate alphabet is large and defeats the purpose of S.A.

Transitions can be interpreted as transitions on a new alphabet of predicates:

\[
\text{REx}(\mathbf{n}) \land \text{DIV}\cdot3(\mathbf{n}) \land (\mathbf{n} \leq 2), \quad 2\text{Exp}(\mathbf{n}) \land 7 \text{DIV}\cdot3(\mathbf{n}) \land \mathbf{n} = 2, \quad 2\text{Exp}(\mathbf{n}) \land \text{DIV}\cdot3(\mathbf{n}) \land 7(n \leq 3)
\]

Alphabet size = 8

SA is a new abstraction to represent DFA's over large alphabets.

- Minimal SA exists and is unique.
- Applying “usual” algorithms does not work.
- New algorithms for minimization.
- Show that new algorithms scale very well.
Overview

- What is SA?
- Related Work
- Incremental min. with oracle
- Improved alg.
- Oracle implementation
- Evaluation
- Conclusions
An Incremental Algorithm for Minimization Of Symbolic Automata

Two conditions

1) The procedure can be interrupted at any time to obtain a (possibly) partially minimized automaton.

2) When allowed to run un-interrupted, it will eventually return the minimal automaton.

*Almeida et al. CIAA 2010.*
Simple Incr. Alg. With Oracle

Assume: an oracle IsEquiv(p,q) returns if \( p \) and \( q \) are equivalent.

\[ p = q \text{ if } \{p\} = \{q\} \]

\[ L(p) = \{ \omega \mid p \xrightarrow{\omega} p', p' \in F \} \]

For every pair of stats \( (p,q) \):

- If IsEquiv(p,q):
  - Merge states \( p \) \& \( q \).
- Else:
  - Continue.

\[ \checkmark \] Interruption Cond.
\[ \checkmark \] Termination Cond.
Observation 1: If $p = q$ then,

If $p = q$ and $p' = q'$ then $p' = q'$

Message: Equivalence of one pair results in equivalence of more pairs.
Observation 2: If \( p \neq q \) then:

\[ \exists w, p \xrightarrow{w} p', q \xrightarrow{w} q', \text{ s.t. } p' \leq_F \beta \wedge q' \leq_F \alpha \wedge p' \neq Fq' \leq_F \]

Suppose \( w = a_1 a_2 a_3 \ldots a_k \)

\[ \begin{array}{cccccc}
p & a_1 & p & a_2 & p & \cdots & a_k \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \cdots & \uparrow \\
q & a_1 & q & a_2 & q & \cdots & q \\
\end{array} \]

Message: Non-equivalence of one pair results in non-equivalence of additional pairs.
Better Inc. Alg.

- Equiv Pairs - additional equivalent pairs inferred
- Path Pairs - additional non-equivalent pairs inferred.
- Non Equiv Pairs - bookkeeping of non-equivalent pairs

```plaintext
For all pairs (p,q) not in NonEquiv Pairs:

- Equiv Pairs ← ∅; Path Pairs ← ∅;
- If IsEquiv(p,q):
- Merge p&q and all pairs in Equiv Pairs;
- Else
- Non Equiv Pairs ← U Path Pairs
```
How To Implement $\text{IsEquiv}(p, q)$?

- keep track of dependencies and use recursion.

$p \equiv q$ iff 
\[ p \rightarrow q, \quad q \rightarrow p' \quad \text{and} \quad q \rightarrow q' \rightarrow p' \rightarrow q' \]

- Pick 0
  
  \[ p \xrightarrow{0} p, \quad q \xrightarrow{0} p \]
  
  check if $p \equiv p$ (recursion)

- Pick 1
  
  \[ p \xrightarrow{1} q, \quad q \xrightarrow{1} q \]
  
  check if $q \equiv q$ (recursion)

- Only if equivalence is established in both cases $p \equiv q$
IsEquiv(p, q):

- Pick 0
  \[ s_1 \rightarrow s_2, \quad s_1 \rightarrow s_3 \]
  recursive call IsEquiv(s_3, s_4)

- Pick 1
  \[ s_1 \rightarrow s_4, \quad s_2 \rightarrow s_3 \]
  recursive call IsEquiv(s_4, s_3)

- Only if both recursive calls return true, \( s_1 \equiv s_2 \).

Question: If alphabet is possibly infinite, would this procedure terminate?
IsEquiv\((p, q)\) using predicates

1. **Function** Equiv-p\((p, q)\):
   
   ```
   if \((p, q) \in \text{eq} \) then
     return False
   
   if \((p, q) \in \text{path} \) then
     return True
   
   path = path \cup \{(p, q)\}
   
   Out_p = \{\varphi \in \Psi_A \mid \exists p', (p, \varphi, p') \in \Delta\}
   
   Out_q = \{\psi \in \Psi_A \mid \exists q', (q, \psi, q') \in \Delta\}
   
   while Out_p \cup Out_q \neq \emptyset do
     Let \(a \in \llbracket (\forall_{\varphi \in \text{Out}_p} \varphi) \land (\forall_{\psi \in \text{Out}_q} \psi) \rrbracket\)
     
     \((p', q') = \text{Normalize}((\text{Find}(\delta(p, a)), \text{Find}(\delta(q, a))))\)
     
     if \(p' \neq q'\) and \((p', q') \notin \text{equiv} \) then
       \text{equiv} = \text{equiv} \cup \{(p', q')\}
       
       if not Equiv-p\((p', q')\) then
         return False
       
     else
       path = path \setminus \{(p', q')\}
       
     Let \(\varphi \in \text{Out}_p\) with \(a \in \llbracket \varphi \rrbracket\)
     
     Let \(\psi \in \text{Out}_q\) with \(a \in \llbracket \psi \rrbracket\)
     
     Out_p = Out_p \setminus \{\varphi\} \cup \{\varphi \land \lnot \psi\}
     
     Out_q = Out_q \setminus \{\psi\} \cup \{\psi \land \lnot \varphi\}
     
     \text{equiv} = \text{equiv} \cup \{(p, q)\}
   
   return True
   ```

2. **Check equivalence by recursive call.**
   
3. **Remove the corresponding \(\varphi \land \psi\) from out-predicates.**

4. **Pick the symbol.**
Contributions (Opinion)

- SA minimization with new features
- Correctness and termination proofs
- Experimental evaluation.
Evaluation: Part I

Comparison of SFA Minimization Algorithms

Average Minimization Time (ms)

Number of States

- Symbolic Incremental
- Symbolic Hopcroft
- Symbolic Moore
Evaluation: Part 2

Comparison of Incremental Minimization Rates

- Symbolic Incremental
- 'Naive' Incremental

Average Amount of Minimization Completed

Number of States

Evaluation: Part 3

Progress of Symbolic Incremental Minimization

Number of States

Percentage of Time Passed

Percentage of Automata Minimized
Conclusions.

- Incremental Alg. for SA minimization.
- Implementation and evaluation.
  - Merging top-down & bottom-up.
  - Incremental S-NFA minimization.

Thank You

psd@cs.unc.edu.