Incremental Minimization Of Symbolic Automata

adverb

verb

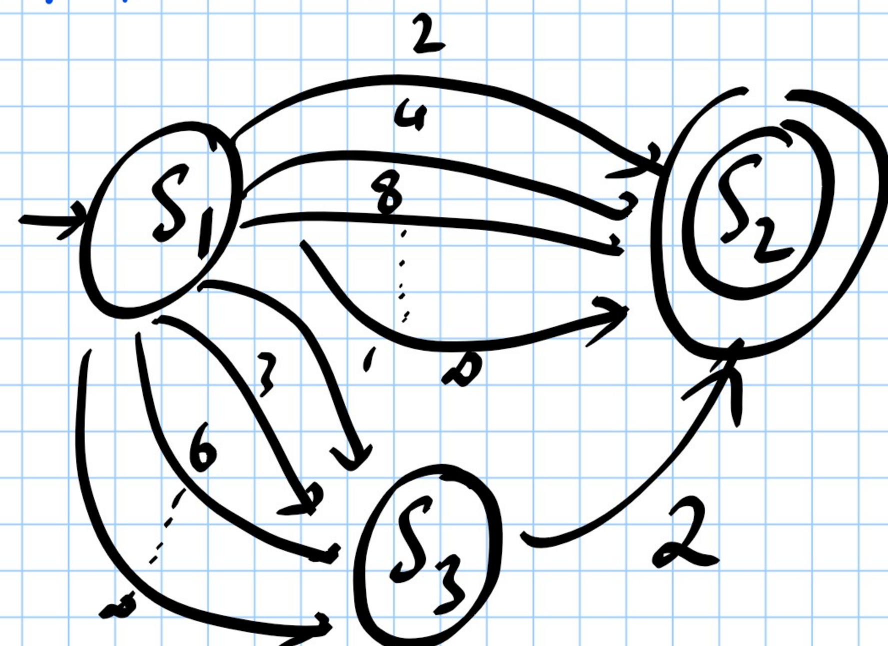
subject

Jonathan Homburg (UCONN)
Parasara **Sridhar** Duggirala (UNC)

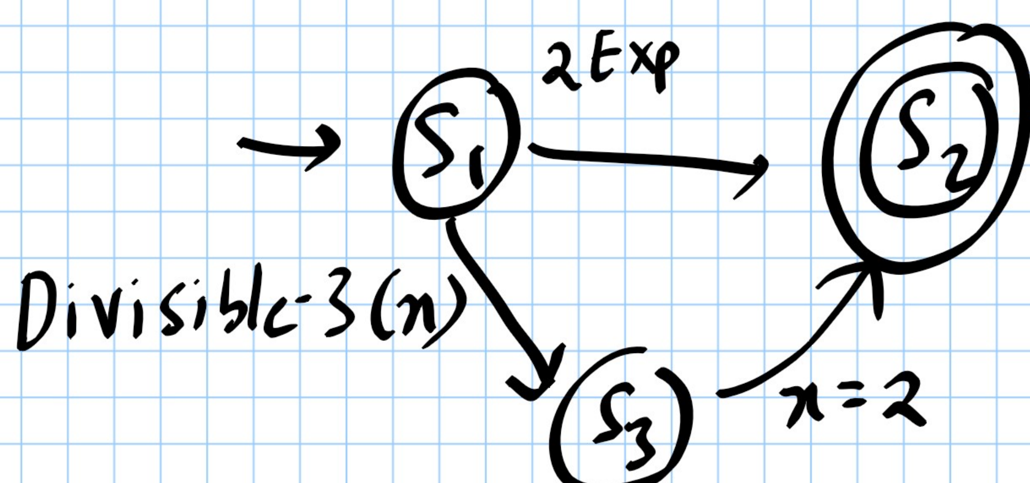
What Is Symbolic Automata?

- DFAs on steroids

Very large alphabet (possibly infinite)



- How to represent transitions? Use predicates



$(\mathcal{A}, Q, q_0, F, \Delta)$

$\mathcal{A}$: Predicates from .....

Q: States

q_0: Initial state

F: Accepting states

Δ: Transitions

Would Any Predicates Work?

⊗ No; The predicates should form an

EFFECTIVE BOOLEAN ALGEBRA

$$\mathcal{A} = (\underbrace{D}_{\text{Domain}}, \underbrace{\Psi}_{\text{Predicates}}, \underbrace{[\![\_]\!]}_{\text{Interpretation}}, \underbrace{\bot}_{\text{Bottom}}, \underbrace{\top}_{\text{Top}}, \underbrace{\wedge}_{\text{Conjunction}}, \underbrace{\vee}_{\text{Disjunction}}, \overbrace{\neg}^{\text{Negation}})$$

- Ψ is closed under Boolean Ops.
- $[\![\_]\!] : \Psi \to 2^D$
- $[\![\bot]\!] = \emptyset$; $[\![\top]\!] = D$
- $[\![\eta \wedge \mu]\!] = [\![\eta]\!] \cap [\![\mu]\!]$;
- $[\![\eta \vee \mu]\!] = [\![\eta]\!] \cup [\![\mu]\!]$
- $[\![\neg \eta]\!] = D \setminus [\![\eta]\!]$

Aren't 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

$\neg \mathrm{Exp}(x) \wedge \mathrm{Div\text{-}3}(x) \wedge (x=2)$, $\quad 2\mathrm{Exp}(x) \wedge \neg \mathrm{Div\text{-}3}(x) \wedge x=2$, $\quad 2\mathrm{Exp}(x) \wedge \mathrm{Div\text{-}3}(x) \wedge \neg(x=2)$

....

Alphabet size = 8

- SA is a new abstraction to represent DFAs over large alphabets.

Prior Work: D'Antoni POPL '14.

- ⊛ Minimal SA exists and is unique.
- ⊛ Applying "usual" algorithms does not work.
- ⊛ New algorithms for minization.
- ⊛ Show that new algorithms scale very well.

Overview

- [x] What is SA?
- [x] 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.

$p \equiv q$ iff $L(p) = L(q)$

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

For every pair of sts. (p,q):
- If IsEquiv(p,q):
  - Merge states p & q.
- Else:
  - Continue.

☑ Interruption Cond.

☑ Termination Cond.

Observation 1: If $p \equiv q$ then,

$$p \xrightarrow{a} p'$$

$$q \xrightarrow{a} q'$$

If $p \equiv q$ and $p' \not\equiv q'$

then

$p' \equiv q'$

Message: Equivalence of one pair results in equivalence of more pairs.

Observation 2 : If $p \not\equiv q$ then;

$\exists\, w,\ p \xrightarrow{w} p',\ q \xrightarrow{w} q'$ s.t. $p' \in F \wedge q' \notin F$ or $p' \notin F \wedge q' \in F$

Suppose $w = a_1 a_2 a_3 \dots a_k$

$$
\begin{tikzcd}
p \arrow[r, "a_1"] & p_1 \arrow[r, "a_2"] \arrow[d, phantom, "\not\equiv"] & p_2 \arrow[r] \arrow[d, phantom, "\not\equiv"] & \cdots \arrow[r, "a_k"] & p' \arrow[d, phantom, "\not\equiv"] \\
q \arrow[r, "a_1"] & q_1 \arrow[r, "a_2"] & q_2 \arrow[r] & \cdots \arrow[r, "a_k"] & q'
\end{tikzcd}
$$

Message : Non-equivalence of one pair results in non-equivalence of additional pairs

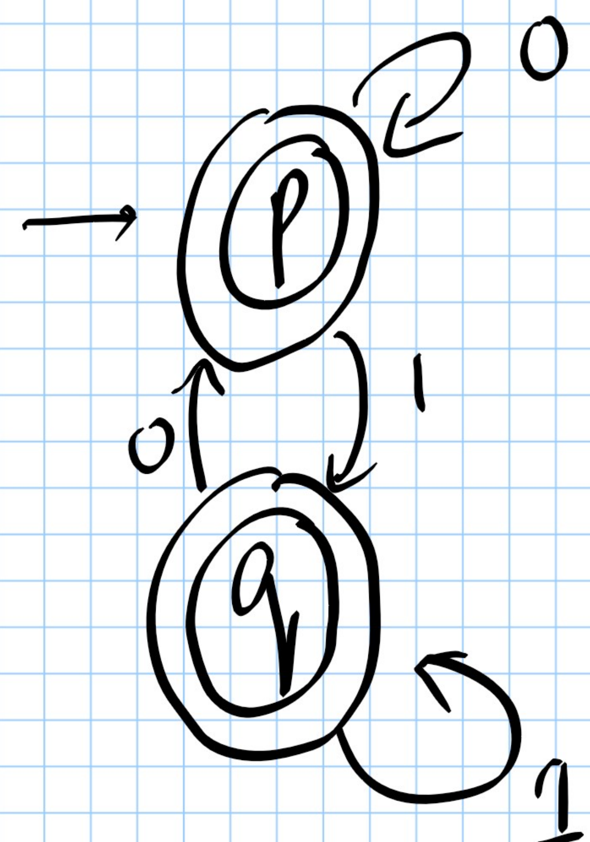
Better Inc. Alg.

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

```
For all pairs (p,q) not in NonEquivPairs:
    EquivPairs ← ∅; PathPairs ← ∅;
    If IsEquiv(p,q):
        Merge p&q and all pairs in EquivPairs;
    Else
        NonEquivPairs ←∪ PathPairs
```

How To Implement IsEquiv(p, q) ?

(*) keep track of dependencies and use recursion.



$p \equiv q$ iff $\forall a,\ p \xrightarrow{a} p',\ q \xrightarrow{a} q'.\ p' \equiv q'$ (*)

- Pick 0

  $p \xrightarrow{0} p,\quad q \xrightarrow{0} p$

  check if $p \equiv p$ (recursion)

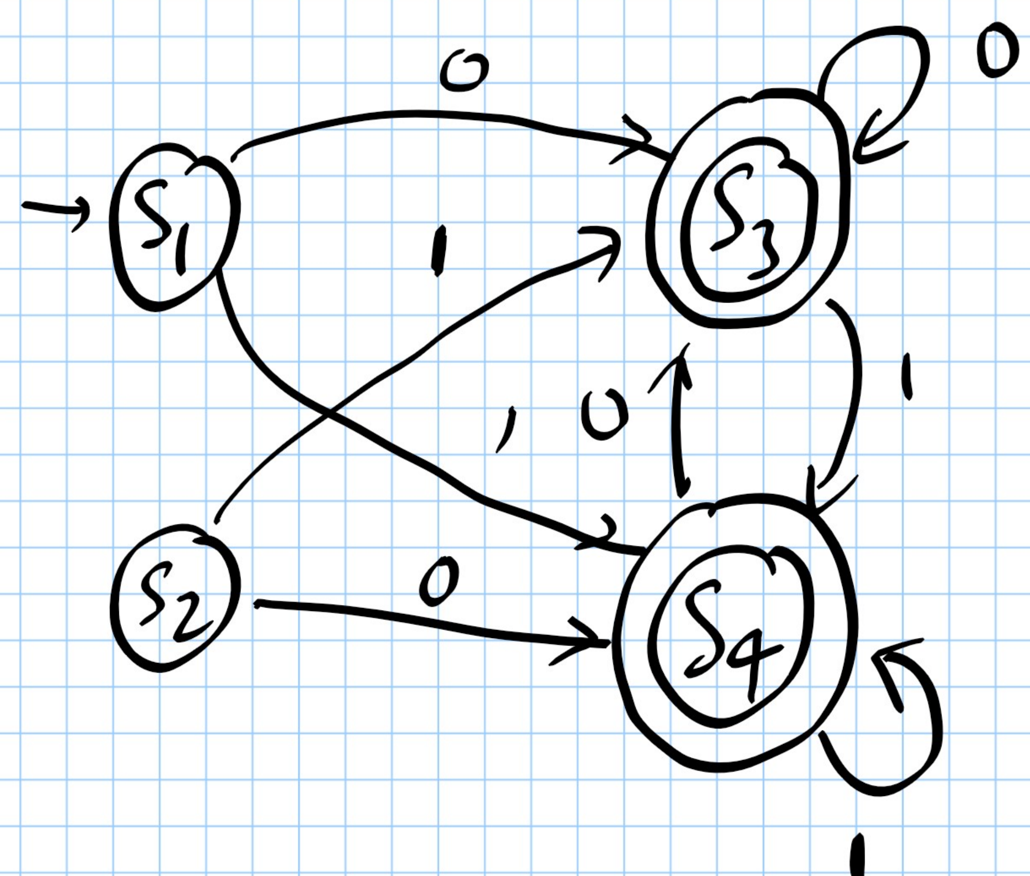
- Pick 1

  $p \xrightarrow{1} q,\quad q \xrightarrow{1} q$

  check if $q \equiv q$ (recursion)

- Only if equivalence is established in both cases $p \equiv q$

IsEquiv(p, q):



IsEquiv(s_1, s_2)

- Pick 0
  $s_1 \xrightarrow{0} s_3$, $s_2 \xrightarrow{0} s_4$
  recursive call IsEquiv(s_3, s_4)
- Pick 1
  $s_1 \xrightarrow{1} s_4$, $s_2 \xrightarrow{1} s_3$
  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.

Function Equiv-p(p, q):
 if $(p, q) \in neq$ **then**
  return False
 if $(p, q) \in path$ **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 $Out_p \cup Out_q \neq \varnothing$ **do**
  Let $a \in [\![(\bigvee_{\varphi \in Out_p} \varphi) \wedge (\bigvee_{\psi \in Out_q} \psi)]\!]$
  $(p', q') = \texttt{Normalize}(\texttt{Find}(\delta(p, a)), \texttt{Find}(\delta(q, a)))$
  if $p' \neq q'$ **and** $(p', q') \notin equiv$ **then**
   $equiv = equiv \cup \{(p', q')\}$
   if not Equiv-p(p', q') **then**
    return False
   else
    $path = path \setminus \{(p', q')\}$
  Let $\varphi \in Out_p$ with $a \in [\![\varphi]\!]$
  Let $\psi \in Out_q$ with $a \in [\![\psi]\!]$
  $Out_p = Out_p \setminus \{\varphi\} \cup \{\varphi \wedge \neg\psi\}$
  $Out_q = Out_q \setminus \{\psi\} \cup \{\psi \wedge \neg\varphi\}$
 $equiv = equiv \cup \{(p, q)\}$
 return True

pick the symbol

check equivalence by recursive call.

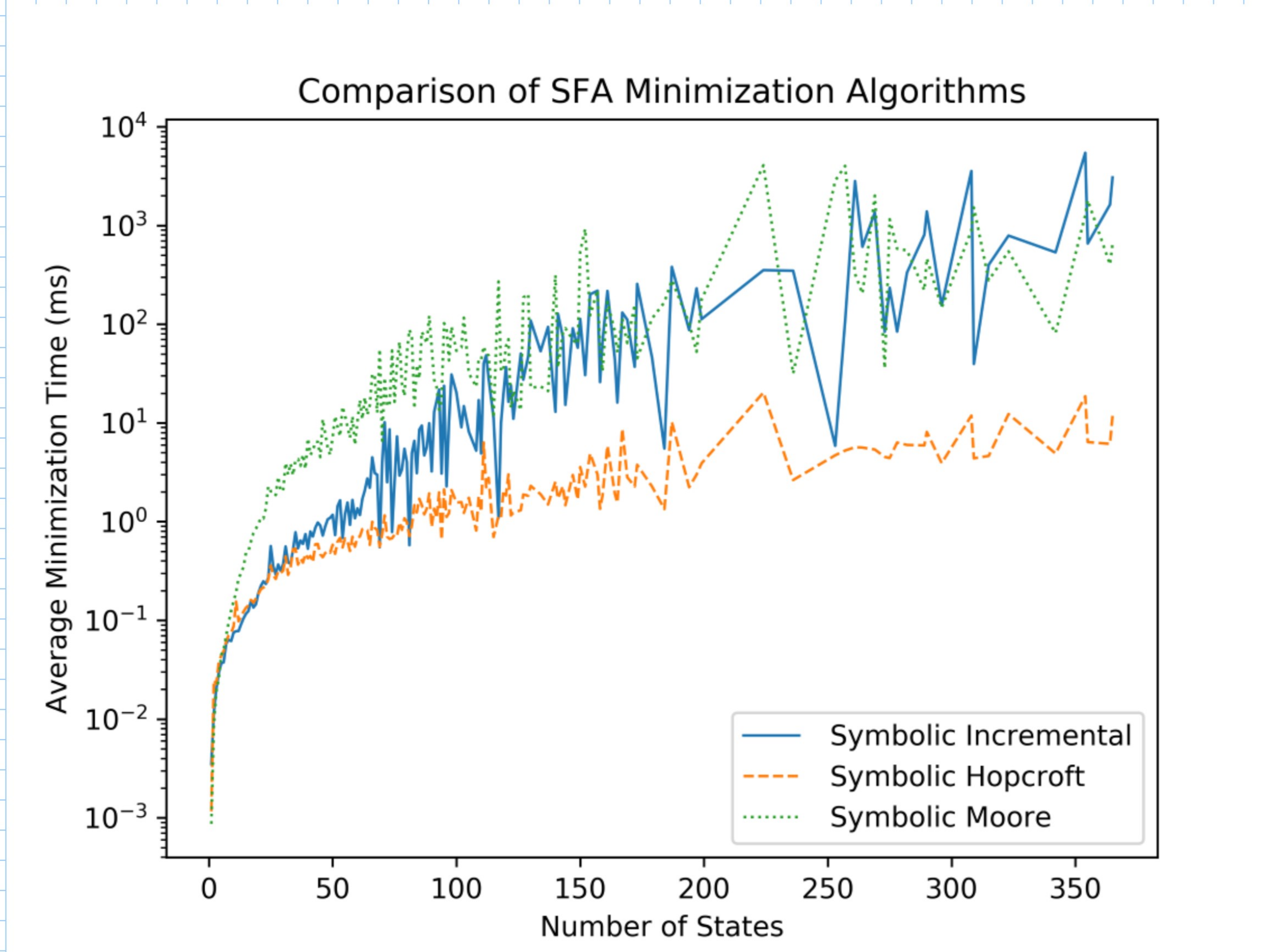
Remove the corresponding $\varphi \wedge \psi$ from out-predicates

® Adopted from Almeida et.al CIAA 2010

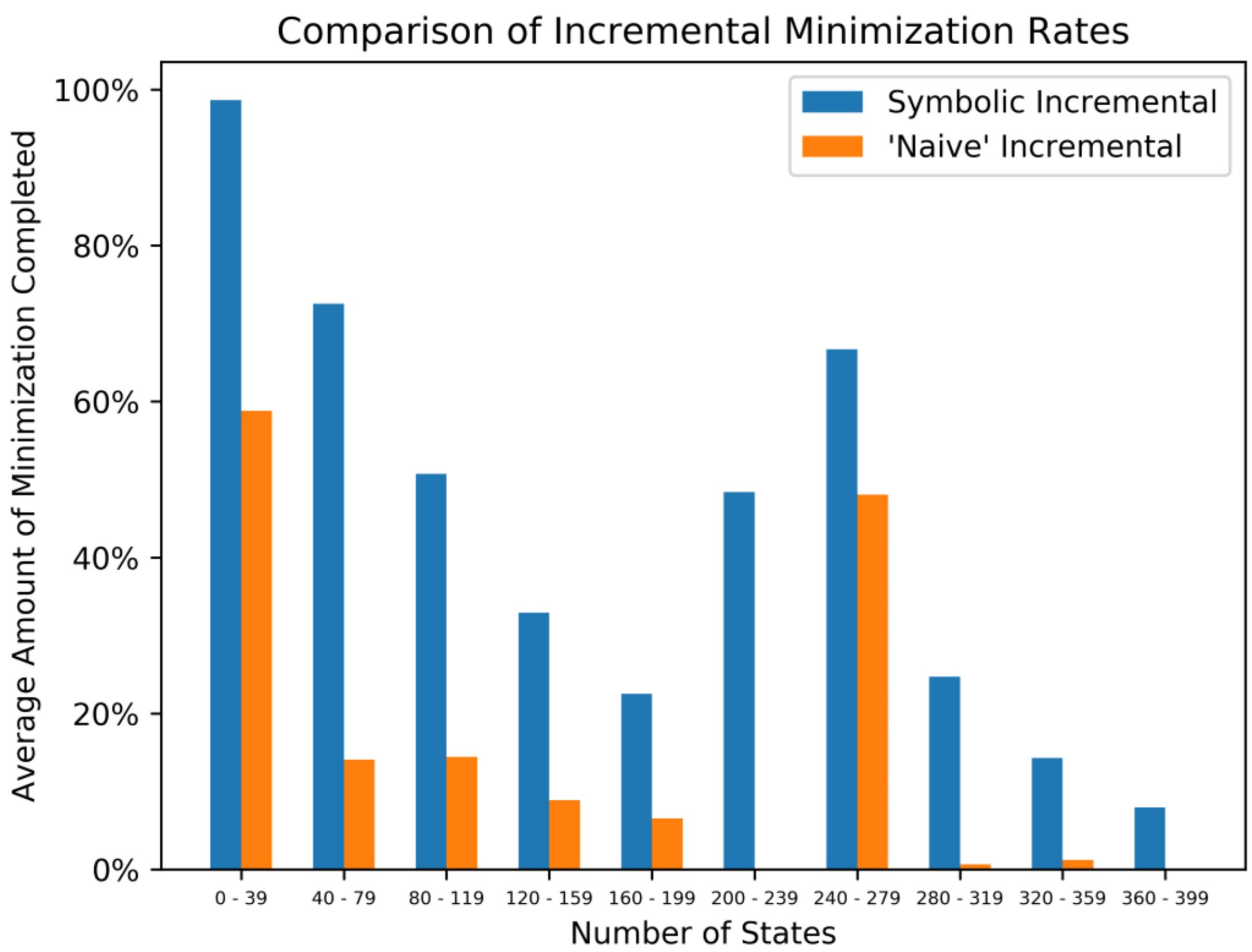
Contributions (Opinion)

- SA minimization with new features
- Correctness and termination proofs
- Experimental evaluation.

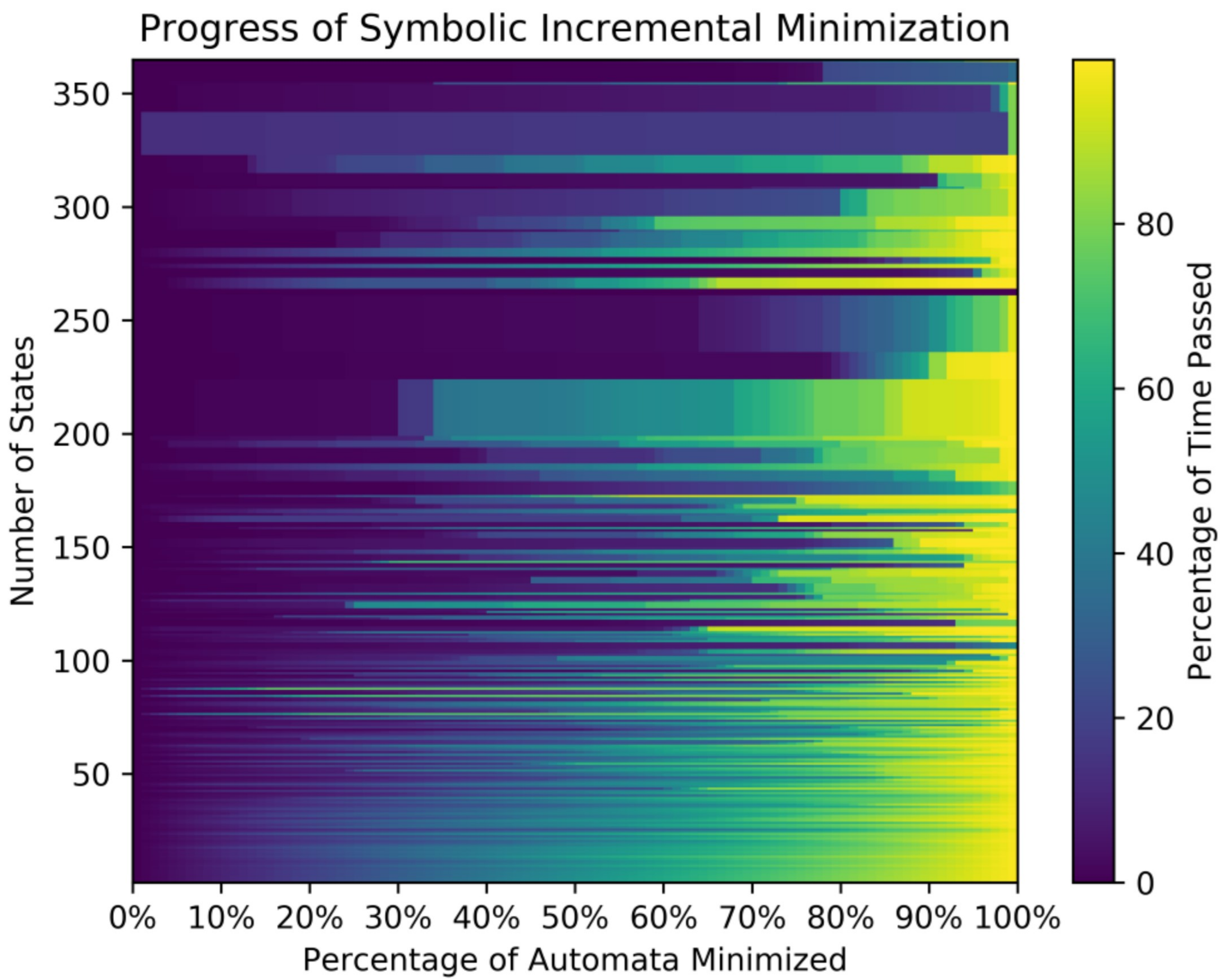
Evaluation: Part I



Evaluation : Part 2



Evaluation : Part 3



Conclusions.

* Incremental Alg. for SA minimization.
* Implementation and evaluation.

- Merging top-down & bottom-up.
- Incremental S-NFA minimization.

Thank You

psd@cs.unc.edu.