A Fixpoint Antichain Algorithm

A faster algorithm to check universality of NFA

Albert-Ludwigs-Universität Freiburg

Felix Freyland

Seminar on Automata Theory at the chair of Software Engineering.
Winter semester 2016/2017

Content

Content

Content

Universality of NFA

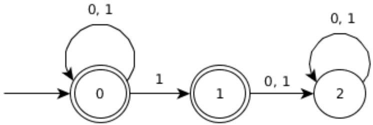
Universality

- An NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$ is universal $\Leftrightarrow L(\mathscr{A}) = \Sigma^*$
- $\mathscr{A}$ accepts every finite word over Σ^*

Universality of NFA

Universality

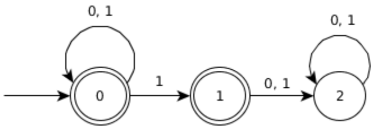
- An NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$ is universal $\Leftrightarrow L(\mathscr{A}) = \Sigma^*$
- $\mathscr{A}$ accepts every finite word over Σ^*

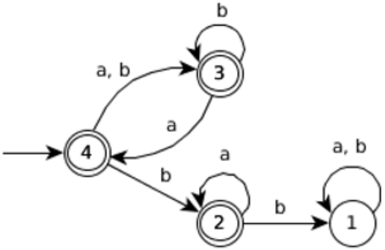


Universality of NFA

Universality

- An NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$ is universal $\Leftrightarrow L(\mathscr{A}) = \Sigma^*$
- $\mathscr{A}$ accepts every finite word over Σ^*





Content

Classical subset construction algorithm

- Consider NFA $\mathscr{A}$ with n states.
- Build corresponding DFA $\mathscr{A}'$ with 2^n states.
- Traverse the DFA $\mathscr{A}'$ starting in $\{Init\}$.
- If a non accepting state is found, $\mathscr{A}'$ hence $\mathscr{A}$ is **not** universal.
- **Problem:** Exponential blow-up of the set of states.

Content

Content

$cpre^{\mathscr{A}}_{\sigma}(s)$ exclusive predecessors of a state set

Definition

Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

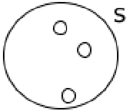
$$cpre^{\mathscr{A}}_{\sigma}(s) = \{l \in Loc \,|\, \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

$cpre_\sigma^{\mathscr{A}}(s)$ exclusive predecessors of a state set

Definition

Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

$$cpre_\sigma^{\mathscr{A}}(s) = \{l \in Loc \mid \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

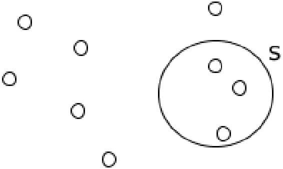


$cpre_\sigma^{\mathscr{A}}(s)$ exclusive predecessors of a state set

Definition

Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

$$cpre_\sigma^{\mathscr{A}}(s) = \{l \in Loc \mid \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

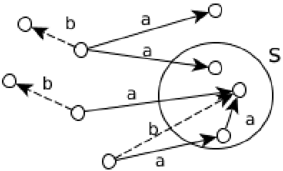


$cpre_\sigma^{\mathscr{A}}(s)$ exclusive predecessors of a state set

Definition

Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

$$cpre_\sigma^{\mathscr{A}}(s) = \{l \in Loc \,|\, \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

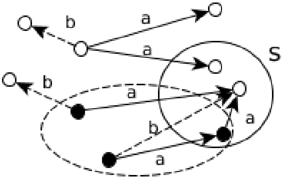


$cpre_{\sigma}^{\mathscr{A}}(s)$ exclusive predecessors of a state set

Definition

Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

$$cpre_{\sigma}^{\mathscr{A}}(s) = \{l \in Loc \,|\, \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

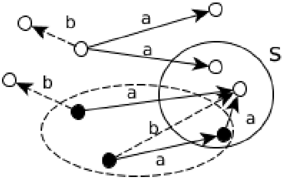


$cpre^{\mathscr{A}}_{\sigma}(s)$ exclusive predecessors of a state set

Definition

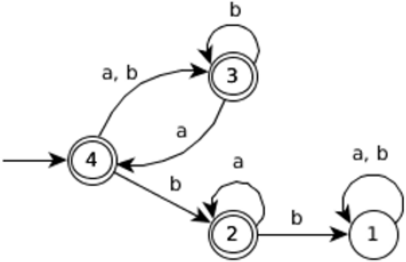
Consider NFA $\mathscr{A} = (Loc, Init, Fin, \delta, \Sigma)$
For $s \subseteq Loc$ we define:

$$cpre^{\mathscr{A}}_{\sigma}(s) = \{l \in Loc \,|\, \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s\}$$

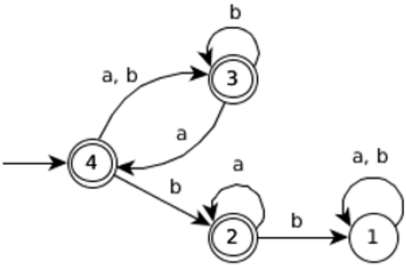


Thus $cpre^{\mathscr{A}}_{a}(s)$ contains all states that with letter a have a transition to some state in s and nowhere else.

$cpre_{\sigma}^{\mathscr{A}}(s)$ and $post_{\sigma}^{\mathscr{A}}(s)$



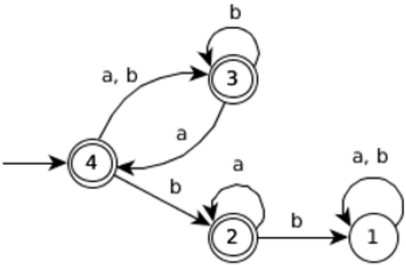
$cpre_\sigma^{\mathscr{A}}(s)$ and $post_\sigma^{\mathscr{A}}(s)$



Example $cpre_\sigma^{\mathscr{A}}(s)$:

- $cpre_a^{\mathscr{A}}(\{1\}) = \{1\}$

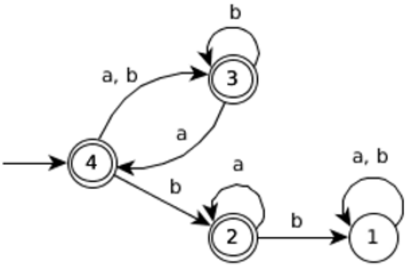
$cpre_\sigma^{\mathscr{A}}(s)$ and $post_\sigma^{\mathscr{A}}(s)$



Example $cpre_\sigma^{\mathscr{A}}(s)$:

- $cpre_a^{\mathscr{A}}(\{1\}) = \{1\}$
- $cpre_b^{\mathscr{A}}(\{1\}) = \{1,2\}$

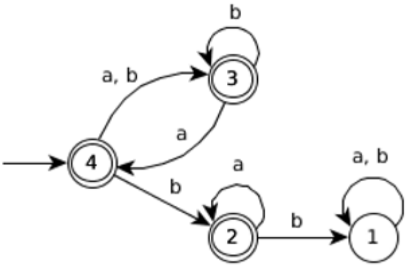
$cpre_{\sigma}^{\mathscr{A}}(s)$ and $post_{\sigma}^{\mathscr{A}}(s)$



Example $cpre_{\sigma}^{\mathscr{A}}(s)$:

- $cpre_{a}^{\mathscr{A}}(\{1\}) = \{1\}$
- $cpre_{b}^{\mathscr{A}}(\{1\}) = \{1,2\}$
- $cpre_{a}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$

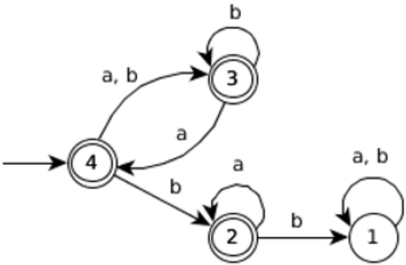
$cpre_{\sigma}^{\mathscr{A}}(s)$ and $post_{\sigma}^{\mathscr{A}}(s)$



Example $cpre_{\sigma}^{\mathscr{A}}(s)$:

- $cpre_{a}^{\mathscr{A}}(\{1\}) = \{1\}$
- $cpre_{b}^{\mathscr{A}}(\{1\}) = \{1,2\}$
- $cpre_{a}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$
- $cpre_{b}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$

$cpre_{\sigma}^{\mathscr{A}}(s)$ and $post_{\sigma}^{\mathscr{A}}(s)$



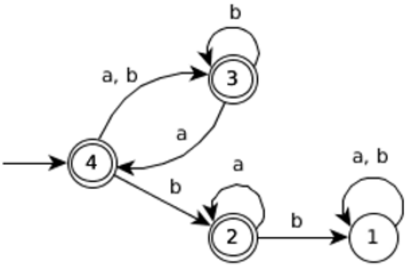
Example $cpre_{\sigma}^{\mathscr{A}}(s)$:

- $cpre_{a}^{\mathscr{A}}(\{1\}) = \{1\}$
- $cpre_{b}^{\mathscr{A}}(\{1\}) = \{1,2\}$
- $cpre_{a}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$
- $cpre_{b}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$

Example $post_{\sigma}^{\mathscr{A}}(s)$:

- $post_{a}^{\mathscr{A}}(\{1,2\}) = \{1,2\}$

$cpre_\sigma^{\mathscr{A}}(s)$ and $post_\sigma^{\mathscr{A}}(s)$



Example $cpre_\sigma^{\mathscr{A}}(s)$:

- $cpre_a^{\mathscr{A}}(\{1\}) = \{1\}$
- $cpre_b^{\mathscr{A}}(\{1\}) = \{1,2\}$
- $cpre_a^{\mathscr{A}}(\{1,2\}) = \{1,2\}$
- $cpre_b^{\mathscr{A}}(\{1,2\}) = \{1,2\}$

Example $post_\sigma^{\mathscr{A}}(s)$:

- $post_a^{\mathscr{A}}(\{1,2\}) = \{1,2\}$
- $post_b^{\mathscr{A}}(\{1,2\}) = \{1\}$

Content

A partial order $\sqsubseteq$ on Antichains

Definition

Let L denote the set of all antichains over 2^{Loc}

$$\forall q, q' \in L : q \sqsubseteq q' \Leftrightarrow \forall s \in q \, \exists s' \in q' : s \subseteq s'$$

A partial order $\sqsubseteq$ on Antichains

Definition

Let L denote the set of all antichains over 2^{Loc}

$$\forall q, q' \in L : q \sqsubseteq q' \Leftrightarrow \forall s \in q \exists s' \in q' : s \subseteq s'$$

- $q \sqsubseteq q'$ iff every $s \in q$ is subset of some $s' \in q'$

A partial order $\sqsubseteq$ on Antichains

Definition

Let L denote the set of all antichains over 2^{Loc}

$$\forall q, q' \in L : q \sqsubseteq q' \Leftrightarrow \forall s \in q \exists s' \in q' : s \subseteq s'$$

- $q \sqsubseteq q'$ iff every $s \in q$ is subset of some $s' \in q'$
- $\sqsubseteq$ is a partial order (reflexive, transitiv, antisymmetric)

A partial order $\sqsubseteq$ on Antichains

Definition

Let L denote the set of all antichains over 2^{Loc}

$$\forall q, q' \in L : q \sqsubseteq q' \Leftrightarrow \forall s \in q \exists s' \in q' : s \subseteq s'$$

- $q \sqsubseteq q'$ iff every $s \in q$ is subset of some $s' \in q'$
- $\sqsubseteq$ is a partial order (reflexive, transitiv, antisymmetric)

Example

- $Loc = \{1, 2, 3, 4\}$

A partial order $\sqsubseteq$ on Antichains

Definition

Let L denote the set of all antichains over 2^{Loc}

$$\forall q, q' \in L : q \sqsubseteq q' \Leftrightarrow \forall s \in q \exists s' \in q' : s \subseteq s'$$

- $q \sqsubseteq q'$ iff every $s \in q$ is subset of some $s' \in q'$
- $\sqsubseteq$ is a partial order (reflexive, transitiv, antisymmetric)

Example

- $Loc = \{1,2,3,4\}$
- $\{\{1\},\{2\},\{3\}\} \sqsubseteq \{\{1,2\},\{2,3\}\}$

Least upper bound $\sqcup$ on Antichains

Definition

For two antichains $q, q' \in L$ the least upper bound (lub) is:

$$q \sqcup q' = Max(\{s \mid s \in q \vee s \in q'\})$$

Least upper bound $\sqcup$ on Antichains

Definition

For two antichains $q, q' \in L$ the least upper bound (lub) is:

$$q \sqcup q' = Max(\{s \mid s \in q \vee s \in q'\})$$

- Thus the antichain $q \sqcup q'$ is the maximum (with regard to set inclusion order) of the union of the two antichains q and q'

Least upper bound $\sqcup$ on Antichains

Definition

For two antichains $q, q' \in L$ the least upper bound (lub) is:

$$q \sqcup q' = Max(\{s \mid s \in q \vee s \in q'\})$$

- Thus the antichain $q \sqcup q'$ is the maximum (with regard to set inclusion order) of the union of the two antichains q and q'

Example

- $q = \{\{1\},\{2\},\{3\}\}$, $q' = \{\{1,2\}\}$

Least upper bound $\sqcup$ on Antichains

Definition

For two antichains $q, q' \in L$ the least upper bound (lub) is:

$$q \sqcup q' = Max(\{s \mid s \in q \vee s \in q'\})$$

- Thus the antichain $q \sqcup q'$ is the maximum (with regard to set inclusion order) of the union of the two antichains q and q'

Example

- $q = \{\{1\},\{2\},\{3\}\}$, $q' = \{\{1,2\}\}$
- $q \sqcup q' = Max(\{\{1\},\{2\},\{3\},\{1,2\}\}) = \{\{1,2\},\{3\}\}$

A Lattice on Antichains

- We have a partial order $(L, \sqsubseteq)$ on antichains

A Lattice on Antichains

- We have a partial order $(L, \sqsubseteq)$ on antichains
- We have a least upper bound (lub) for two antichains

A Lattice on Antichains

- We have a partial order $(L, \sqsubseteq)$ on antichains
- We have a least upper bound (lub) for two antichains
- A greatest lower bound (glb) can suitably be defined, such... that we get a lattice on antichains.

A Lattice on Antichains

- We have a partial order $(L, \sqsubseteq)$ on antichains
- We have a least upper bound (lub) for two antichains
- A greatest lower bound (glb) can suitably be defined, such... that we get a lattice on antichains.
- A lattice is a partially ordered set, where every two elements have a lub and a glb

A Lattice on Antichains

- We have a partial order $(L, \sqsubseteq)$ on antichains
- We have a least upper bound (lub) for two antichains
- A greatest lower bound (glb) can suitably be defined, such... that we get a lattice on antichains.
- A lattice is a partially ordered set, where every two elements have a lub and a glb
- Lattice property is needed later on for correctness of the algorithm

Content

Monotone function on antichains $CPre^{\mathscr{A}}(q)$

Definition

The concept of predecessors is extended to antichains by:

$$CPre^{\mathscr{A}} : L \rightarrow L$$

$$CPre^{\mathscr{A}}(q) = Max(\{s \mid \exists\, s' \in q\ \exists\, \sigma \in \Sigma : s = cpre_{\sigma}^{\mathscr{A}}(s')\})$$

Monotone function on antichains $CPre^{\mathscr{A}}(q)$

Definition

The concept of predecessors is extended to antichains by:

$$CPre^{\mathscr{A}} : L \to L$$

$$CPre^{\mathscr{A}}(q) = Max(\{s \mid \exists\, s' \in q\ \exists \sigma \in \Sigma : s = cpre_{\sigma}^{\mathscr{A}}(s')\})$$

- Monotonicity: $q \sqsubseteq q' \Rightarrow CPre^{\mathscr{A}}(q) \sqsubseteq CPre^{\mathscr{A}}(q')$

Monotone function on antichains $CPre^{\mathscr{A}}(q)$

Definition

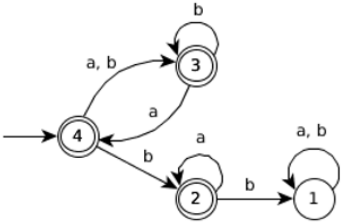
The concept of predecessors is extended to antichains by:

$$CPre^{\mathscr{A}} : L \to L$$

$$CPre^{\mathscr{A}}(q) = Max(\{s \mid \exists\, s' \in q\ \exists \sigma \in \Sigma : s = cpre_{\sigma}^{\mathscr{A}}(s')\})$$

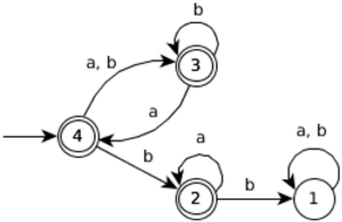
- Monotonicity: $q \sqsubseteq q' \Rightarrow CPre^{\mathscr{A}}(q) \sqsubseteq CPre^{\mathscr{A}}(q')$
- follows from subset inclusion order and Def. of $cpre_{\sigma}^{\mathscr{A}}(s)$

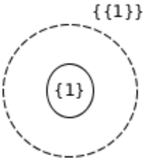
Monotone function on antichains $CPre^{\mathscr{A}}(q)$



Example: $CPre^{\mathscr{A}}(\{\{1\}\})$

Monotone function on antichains $CPre^{\mathscr{A}}(q)$

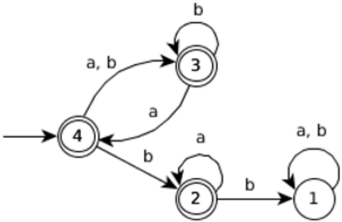


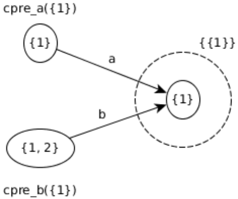


Example: $CPre^{\mathscr{A}}(\{\{1\}\})$

- we start with the antichain $\{\{1\}\}$

Monotone function on antichains $CPre^{\mathscr{A}}(q)$

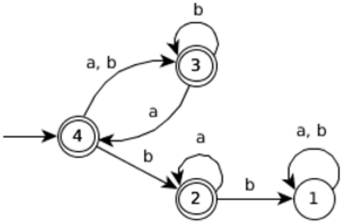


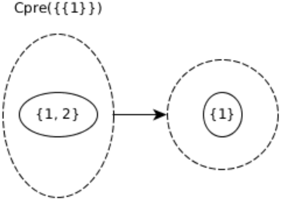


Example: $CPre^{\mathscr{A}}(\{\{1\}\})$

- we start with the antichain $\{\{1\}\}$
- calculate $cpre_a(\{1\}) = \{1\}$ and $cpre_b(\{1\}) = \{1,2\}$

Monotone function on antichains $CPre^{\mathscr{A}}(q)$





Example: $CPre^{\mathscr{A}}(\{\{1\}\})$

- we start with the antichain $\{\{1\}\}$
- calculate $cpre_a(\{1\}) = \{1\}$ and $cpre_b(\{1\}) = \{1,2\}$
- $CPre^{\mathscr{A}}(\{\{1\}\}) = Max(\{\{1,2\},\{1\}\}) = \{\{1,2\}\}$

Content

Content

The general idea

- Start with antichain $F = \{\overline{Fin}\}$ and set $Frontier = F$

The general idea

- Start with antichain $F = \{\overline{Fin}\}$ and set $Frontier = F$
- Repeatedly compute $F = F \sqcup CPre^{\mathscr{A}}(Frontier)$ in a loop

The general idea

- Start with antichain $F = \{\overline{Fin}\}$ and set $Frontier = F$
- Repeatedly compute $F = F \sqcup CPre^{\mathscr{A}}(Frontier)$ in a loop
- *Tarski's Fixpoint Theorem* implies that the monotone function $CPre^{\mathscr{A}}(q)$ on a complete lattice has a least fixpoint

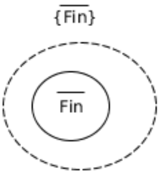
The general idea

- Start with antichain $F = \{\overline{Fin}\}$ and set $Frontier = F$
- Repeatedly compute $F = F \sqcup CPre^{\mathscr{A}}(Frontier)$ in a loop
- *Tarski's Fixpoint Theorem* implies that the monotone function $CPre^{\mathscr{A}}(q)$ on a complete lattice has a least fixpoint
- Thus after some iteration n, F stops growing, i.e. $F_n = F_{n-1}$

The general idea

- Start with antichain $F = \{\overline{Fin}\}$ and set $Frontier = F$
- Repeatedly compute $F = F \sqcup CPre^{\mathscr{A}}(Frontier)$ in a loop
- *Tarski's Fixpoint Theorem* implies that the monotone function $CPre^{\mathscr{A}}(q)$ on a complete lattice has a least fixpoint
- Thus after some iteration n, F stops growing, i.e. $F_n = F_{n-1}$
- Iff $\{Init\} \sqsubseteq F$ $\mathscr{A}$ is not universal.

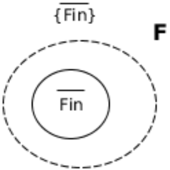
Algorithm 0



Initialization

- We start with the antichain of the set of non accepting states

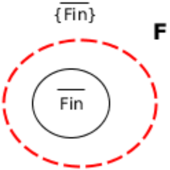
Algorithm 0



Initialization

- We start with the antichain of the set of non accepting states
- $F \leftarrow \{\overline{Fin}\}$

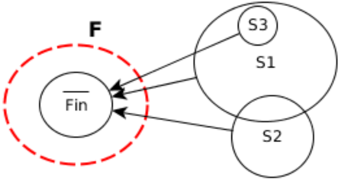
Algorithm 0



Initialization

- We start with the antichain of the set of non accepting states
- $F \leftarrow \{\overline{Fin}\}$
- *Frontier* $\leftarrow F$

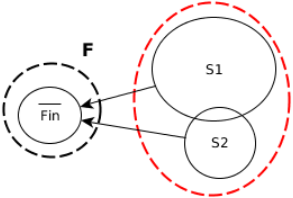
Algorithm 1



First Iteration

- s_1, s_2, s_3 are $cpre_\sigma(s)$ for all σ and all $s \in Frontier$

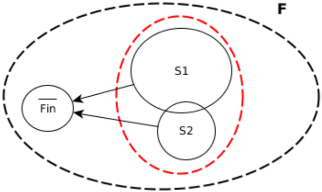
Algorithm 1



First Iteration

- s_1, s_2, s_3 are $cpre_\sigma(s)$ for all σ and all $s \in Frontier$
- $Frontier = CPre^{\mathscr{A}}(Frontier) = \{s_1, s_2\}$

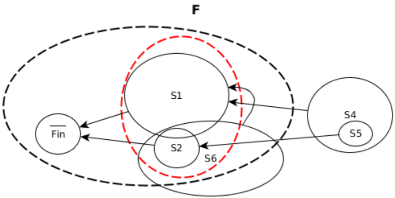
Algorithm 1



First Iteration

- s_1, s_2, s_3 are $cpre_\sigma(s)$ for all σ and all $s \in Frontier$
- $Frontier = CPre^{\mathscr{A}}(Frontier) = \{s_1, s_2\}$
- $F \leftarrow F \sqcup Frontier$

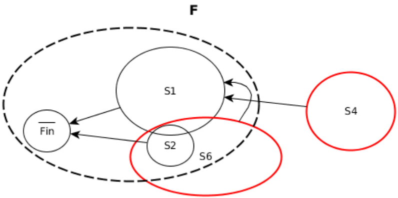
Algorithm 2



Second Iteration

- s_4, s_5, s_6 are $cpre(s)$ for all σ and all $s \in Frontier$

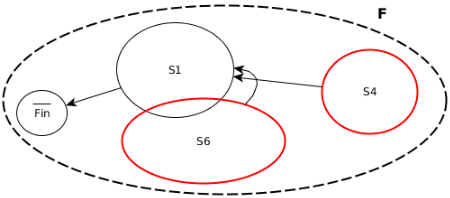
Algorithm 2



Second Iteration

- s_4, s_5, s_6 are $cpre(s)$ for all σ and all $s \in Frontier$
- $Frontier = CPre^{\mathscr{A}}(Frontier) = \{s_4, s_6\}$

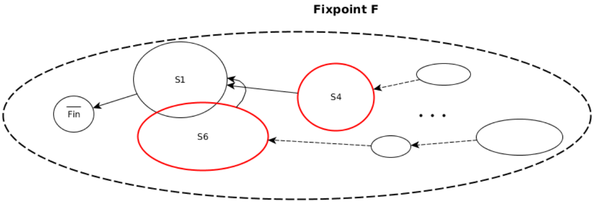
Algorithm 2



Second Iteration

- s_4, s_5, s_6 are $cpre(s)$ for all σ and all $s \in Frontier$
- $Frontier = CPre^{\mathscr{A}}(Frontier) = \{s_4, s_6\}$
- $F \leftarrow F \sqcup Frontier$

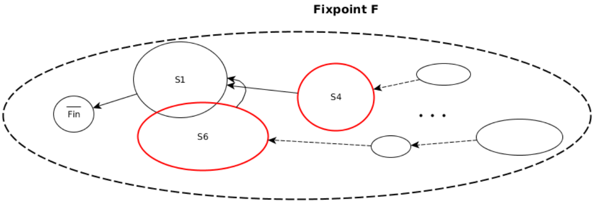
Algorithm Termination



Termination

- The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathscr{F}$ where $q_i = CPre^{\mathscr{A}}(q_{i-1}) \sqcup \{\overline{Fin}\}$

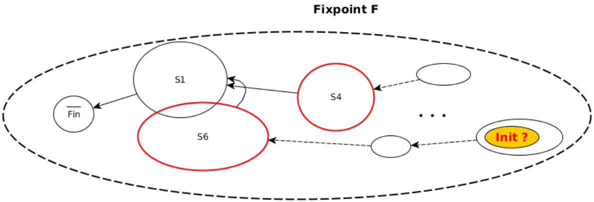
Algorithm Termination



Termination

- The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathscr{F}$ where $q_i = CPre^{\mathscr{A}}(q_{i-1}) \sqcup \{\overline{Fin}\}$
- Tarski's Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint F.

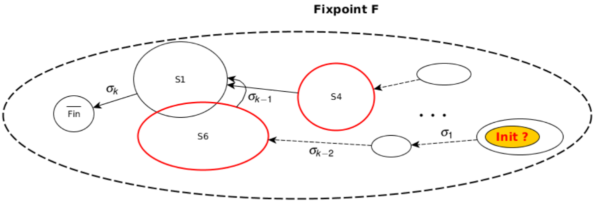
Algorithm Termination



Termination

- The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathscr{F}$ where $q_i = CPre^{\mathscr{A}}(q_{i-1}) \sqcup \{\overline{Fin}\}$
- Tarski's Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint F.
- $Lang(\mathscr{A}) \neq \Sigma^* \Leftrightarrow \{Init\} \sqsubseteq F$

Algorithm Termination



Termination

- The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathscr{F}$ where $q_i = CPre^{\mathscr{A}}(q_{i-1}) \sqcup \{\overline{Fin}\}$
- Tarski's Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint F.
- $Lang(\mathscr{A}) \neq \Sigma^* \Leftrightarrow \{Init\} \sqsubseteq F$

Content

Comparison of Classical and Antichain Algorithm

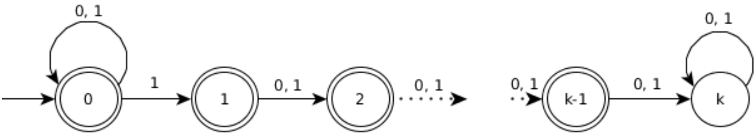
Theorem

For the familiy of $\mathscr{A}_k$, $k \geq 2$ with $k+1$ states, the Backward Antichain Algorithm is polynomial in k, whereas the classical subset construction algorithm is exponential in k

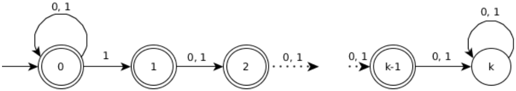
Comparison of Classical and Antichain Algorithm

Theorem

For the familiy of $\mathscr{A}_k$, $k \geq 2$ with $k+1$ states, the Backward Antichain Algorithm is polynomial in k, whereas the classical subset construction algorithm is exponential in k

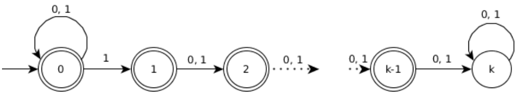


Classical



Classical

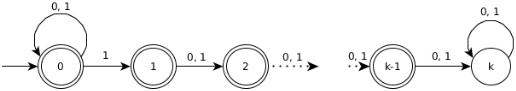
Classical



Classical

- DFA of 2^{k+1} states

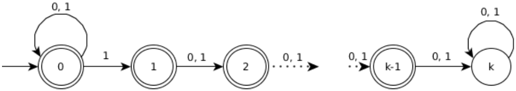
Classical



Classical

- DFA of 2^{k+1} states
- 2^k reachable states, which are all accepting.

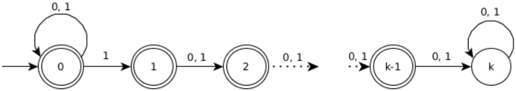
Classical



Classical

- DFA of 2^{k+1} states
- 2^k reachable states, which are all accepting.
- Algorithm traverses the tree of 2^k accepting states

Classical



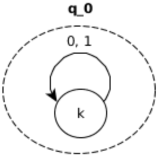
Classical

- DFA of 2^{k+1} states
- 2^k reachable states, which are all accepting.
- Algorithm traverses the tree of 2^k accepting states
- runtime is exponential in k

Comparison of Classical and Antichain Algorithm

Antichain Algorithm

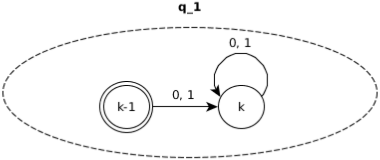
Comparison of Classical and Antichain Algorithm



Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$

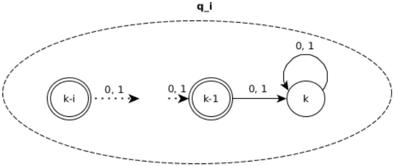
Comparison of Classical and Antichain Algorithm



Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$

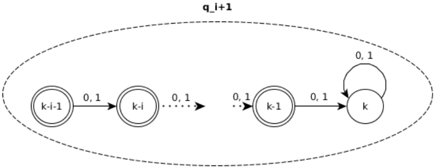
Comparison of Classical and Antichain Algorithm



Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$
- In each iteration:
  $q_{i+1} = CPre(q_i) \sqcup \{\{I_k\}\}, = \{\{k-(i+1), k-i, \ldots, k\}\} \quad for \quad i < k$

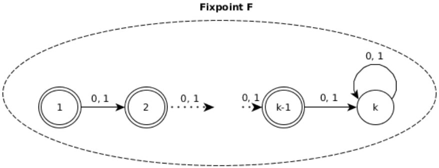
Comparison of Classical and Antichain Algorithm



Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$
- In each iteration:
  $q_{i+1} = CPre(q_i) \sqcup \{\{l_k\}\}, = \{\{k-(i+1), k-i, \dots, k\}\} \quad for \quad i < k$

Comparison of Classical and Antichain Algorithm



Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$
- In each iteration:
  $q_{i+1} = CPre(q_i) \sqcup \{\{l_k\}\}, = \{\{k-(i+1), k-i, \ldots, k\}\} \quad for \quad i < k$
- Stops after k iterations with $q_k = q_{k-1} = \{\{1, \ldots, k\}\}$

Comparison of Classical and Antichain Algorithm

Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$
- In each iteration:
  $q_{i+1} = CPre(q_i) \sqcup \{\{l_k\}\}, = \{\{k-(i+1), k-i, \ldots, k\}\} \quad for \quad i < k$
- Stops after k iterations with $q_k = q_{k-1} = \{\{1, \ldots, k\}\}$
- Checks in each Iteration $\{\{l_0\}\} \sqsubseteq q_i$ in linear time

Comparison of Classical and Antichain Algorithm

Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k-1, k\}\}$
- In each iteration:
  $q_{i+1} = CPre(q_i) \sqcup \{\{l_k\}\}, = \{\{k-(i+1), k-i, \ldots, k\}\} \quad \mathit{for} \quad i < k$
- Stops after k iterations with $q_k = q_{k-1} = \{\{1, \ldots, k\}\}$
- Checks in each Iteration $\{\{l_0\}\} \sqsubseteq q_i$ in linear time
- The computation of the $CPre()$ in each iteration takes linear time

Conclusion

- The Backward antichain fixpoint algorithm is considerably faster for a certain family of NFA

Conclusion

- The Backward antichain fixpoint algorithm is considerably faster for a certain family of NFA
- Empirical comparisons of antichain and classical algorithm on randomly generated NFA show, that antichain is up to 200 times faster.

Conclusion

- The Backward antichain fixpoint algorithm is considerably faster for a certain family of NFA
- Empirical comparisons of antichain and classical algorithm on randomly generated NFA show, that antichain is up to 200 times faster.
- The higher the density of accepting states the more advantageous is the antichain approach.

Conclusion

- The Backward antichain fixpoint algorithm is considerably faster for a certain family of NFA
- Empirical comparisons of antichain and classical algorithm on randomly generated NFA show, that antichain is up to 200 times faster.
- The higher the density of accepting states the more advantageous is the antichain approach.
- Antichain algorithms are also applied to other problems like language inclusion

References

- M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. (2006). Antichains: A new algorithm for checking universality of finite automata. In Proc. of CAV: Computer Aided Verification, LNCS 4144, pages 17-30. Springer, 2006.
- L. Doyen and J.-F. Raskin. (2010). Antichain Algorithms for Finite Automata. In Proc. of TACAS: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 6015, pages 2-22. Springer, 2010.