A Fixpoint Antichain Algorithm
A faster algorithm to check universality of NFA

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

Basic Problem
    Universality of NFA
    Classical subset construction

Preliminaries
    Predecessors on state sets
    Lattice of Antichains
    A monotone predecessor function on Antichains

Antichain Algorithm to check universality
    The Algorithm at work
    Antichain Algorithm vs. Classical
Content

Basic Problem
- Universality of NFA
- Classical subset construction

Preliminaries
- Predecessors on state sets
- Lattice of Antichains
- A monotone predecessor function on Antichains

Antichain Algorithm to check universality
- The Algorithm at work
- Antichain Algorithm vs. Classical
Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
Universality of NFA

Universality

- An NFA $\mathcal{A} = (\text{Loc}, \text{Init}, \text{Fin}, \delta, \Sigma)$ is universal $\iff L(\mathcal{A}) = \Sigma^*$
- $\mathcal{A}$ accepts every finite word over $\Sigma^*$
Universality of NFA

An NFA $A = (Loc, Init, Fin, \delta, \Sigma)$ is universal $\iff L(A) = \Sigma^*$

$A$ accepts every finite word over $\Sigma^*$
Universality of NFA

- An NFA $A = (Loc, Init, Fin, \delta, \Sigma)$ is universal $\iff L(A) = \Sigma^*$
- $A$ accepts every finite word over $\Sigma^*$
Basic Problem

Universality of NFA
Classical subset construction

Preliminaries

Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality

The Algorithm at work
Antichain Algorithm vs. Classical
Consider NFA $\mathcal{A}$ with $n$ states.

Build corresponding DFA $\mathcal{A}'$ with $2^n$ states.

 Traverse the DFA $\mathcal{A}'$ starting in $\{\text{Init}\}$.

If a non accepting state is found, $\mathcal{A}'$ hence $\mathcal{A}$ is not universal.

**Problem:** Exponential blow-up of the set of states.
Content

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
Content

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
Definition

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

$$cpre_\sigma^\mathcal{A}(s) = \{ l \in \text{Loc} \mid \forall l' \in \text{Loc} : \delta(l, \sigma, l') \Rightarrow l' \in s \}$$
$\text{cpre}_\sigma^A(s)$ exclusive predecessors of a state set

**Definition**

Consider NFA $A = (\text{Loc}, \text{Init}, \text{Fin}, \delta, \Sigma)$

For $s \subseteq \text{Loc}$ we define:

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

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

$$cpre_\sigma^\mathcal{A}(s) = \{ l \in \text{Loc} \mid \forall l' \in \text{Loc} : \delta(l, \sigma, l') \Rightarrow l' \in s \}$$
\[ \text{cpre}_\sigma(s) \text{ exclusive predecessors of a state set} \]

**Definition**

Consider NFA \( \mathcal{A} = (\text{Loc}, \text{Init}, \text{Fin}, \delta, \Sigma) \)

For \( s \subseteq \text{Loc} \) we define:

\[
\text{cpre}_\sigma(s) = \{ l \in \text{Loc} \mid \forall l' \in \text{Loc} : \delta(l, \sigma, l') \Rightarrow l' \in s \}
\]
$cpre_{\sigma} (s)$ exclusive predecessors of a state set

**Definition**

Consider NFA $\mathcal{A} = (Loc, Init, Fin, \delta, \Sigma)$

For $s \subseteq Loc$ we define:

$$cpre_{\sigma} (s) = \{ l \in Loc \mid \forall l' \in Loc : \delta(l, \sigma, l') \Rightarrow l' \in s \}$$
\( cpre_{\sigma}^A (s) \) exclusive predecessors of a state set

**Definition**

Consider NFA \( A = (\text{Loc}, \text{Init}, \text{Fin}, \delta, \Sigma) \)

For \( s \subseteq \text{Loc} \) we define:

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

Thus \( cpre_{\sigma}^A (s) \) contains all states that with letter \( a \) have a transition to some state in \( s \) and nowhere else.
$cpre_{\mathcal{A}}(s)$ and $post_{\mathcal{A}}(s)$
$cpre_{\sigma} (s)$ and $post_{\sigma} (s)$

Example $cpre_{\sigma} (s)$:

- $cpre_{a} (\{1\}) = \{1\}$
The text in the image discusses "cpre_{\sigma} (s) and post_{\sigma} (s)" with a graph illustrating these concepts. An example is provided for both cpre and post functions, showing how they operate on sets. The example includes two cases:

- For cpre_{a}({1}) = {1}
- For cpre_{b}({1}) = {1, 2}

The graph contains nodes labeled with numbers (1, 2, 3, 4) and edges labeled with 'a' and 'b' connecting these nodes in a specific pattern.
$cpre_{\sigma} (s)$ and $post_{\sigma} (s)$

Example $cpre_{\sigma} (s)$:

- $cpre_a (\{1\}) = \{1\}$
- $cpre_b (\{1\}) = \{1, 2\}$
- $cpre_a (\{1, 2\}) = \{1, 2\}$
$cpre_{\sigma}(s)$ and $post_{\sigma}(s)$

Example $cpre_{\sigma}(s)$:

- $cpre_a(\{1\}) = \{1\}$
- $cpre_b(\{1\}) = \{1, 2\}$
- $cpre_a(\{1, 2\}) = \{1, 2\}$
- $cpre_b(\{1, 2\}) = \{1, 2\}$
cpre_σ^A (s) and post_σ^A (s)

Example cpre_σ^A (s):

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

Example post_σ^A (s):

- \( post_a^A (\{1, 2\}) = \{1, 2\} \)
$cpre_\sigma^A(s)$ and $post_\sigma^A(s)$

Example $cpre_\sigma^A(s)$:
- $cpre_a^A(\{1\}) = \{1\}$
- $cpre_b^A(\{1\}) = \{1, 2\}$
- $cpre_a^A(\{1, 2\}) = \{1, 2\}$
- $cpre_b^A(\{1, 2\}) = \{1, 2\}$

Example $post_\sigma^A(s)$:
- $post_a^A(\{1, 2\}) = \{1, 2\}$
- $post_b^A(\{1, 2\}) = \{1\}$
Content

Basic Problem
  Universality of NFA
  Classical subset construction

Preliminaries
  Predecessors on state sets
  **Lattice of Antichains**
  A monotone predecessor function on Antichains

Antichain Algorithm to check universality
  The Algorithm at work
  Antichain Algorithm vs. Classical
A partial order $\sqsubseteq$ on Antichains

**Definition**

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

\[
\forall q, q' \in L : q \sqsubseteq q' \iff \forall s \in q \ \exists s' \in q' : s \subseteq s'
\]
A partial order $\sqsubseteq$ on Antichains

**Definition**

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

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

$q \sqsubseteq q'$ 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 $2^{Loc}$

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

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

**Definition**

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

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

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

**Example**

- $\text{Loc} = \{1, 2, 3, 4\}$
A partial order \( \sqsubseteq \) on Antichains

**Definition**

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

\[
\forall q, q' \in L : q \sqsubseteq q' \iff \forall s \in q \exists s' \in q' : s \subseteq s'
\]

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

$$q \sqcup q' = \text{Max}(\{s \mid s \in q \lor s \in q'\})$$
Least upper bound ⊔ on Antichains

**Definition**

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

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

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:

\[
q \sqcup q' = \text{Max}(\{s | s \in q \lor s \in q'\})
\]

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:

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

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**

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

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
Monotone function on antichains $C\text{Pre}^{\mathcal{A}}(q)$

**Definition**

The concept of predecessors is extended to antichains by:

$$
C\text{Pre}^{\mathcal{A}} : L \to L
$$

$$
C\text{Pre}^{\mathcal{A}}(q) = \text{Max}(\{s \mid \exists s' \in q \exists \sigma \in \Sigma : s = \text{cpre}^{\mathcal{A}}_\sigma(s')\})
$$
Monotone function on antichains $CPre^\mathcal{A}(q)$

**Definition**

The concept of predecessors is extended to antichains by:

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

$$CPre^\mathcal{A}(q) = \text{Max}(\{s \mid \exists s' \in q \exists \sigma \in \Sigma : s = cpre^\sigma_\sigma(s')\})$$

- Monotonicity: $q \subseteq q' \Rightarrow CPre^\mathcal{A}(q) \subseteq CPre^\mathcal{A}(q')$
Monotone function on antichains $\text{CPre}^A(q)$

Definition

The concept of predecessors is extended to antichains by:

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

$$\text{CPre}^A(q) = \text{Max}(\{s \mid \exists s' \in q \exists \sigma \in \Sigma : s = \text{cpre}_\sigma^A(s')\})$$

- Monotonicity: $q \sqsubseteq q' \Rightarrow \text{CPre}^A(q) \sqsubseteq \text{CPre}^A(q')$
- follows from subset inclusion order and Def. of $\text{cpre}_\sigma^A(s)$
Monotone function on antichains $CPre^A(q)$

Example: $CPre^A(\{1\})$
Monotone function on antichains $CPre^\mathcal{A}(q)$

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

- we start with the antichain $\{1\}$
Monotone function on antichains $CPre^A(q)$

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

- we start with the antichain $\{\{1\}\}$
- calculate $cpre_a(\{1\}) = \{1\}$ and $cpre_b(\{1\}) = \{1, 2\}$
Monotone function on antichains \( CPre^A(q) \)

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

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

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
Content

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
The general idea

- Start with antichain $F = \{\text{Fin}\}$ and set $\text{Frontier} = F$
The general idea

- Start with antichain $F = \{\text{Fin}\}$ and set $\text{Frontier} = F$
- Repeatedly compute $F = F \sqcup C\text{Pre}^\prec (\text{Frontier})$ in a loop
The general idea

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

- Start with antichain $F = \{\text{Fin}\}$ and set $\text{Frontier} = F$
- Repeatedly compute $F = F \sqcup CPre^A(\text{Frontier})$ in a loop
- Tarski’s Fixpoint Theorem implies that the monotone function $CPre^A(q)$ 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 $F = \{ \overline{\text{Fin}} \}$ and set $\text{Frontier} = F$
- Repeatedly compute $F = F \sqcup \text{CPre}^\mathcal{A} (\text{Frontier})$ in a loop
- Tarski’s Fixpoint Theorem implies that the monotone function $\text{CPre}^\mathcal{A} (q)$ on a complete lattice has a least fixpoint
- Thus after some iteration $n$, $F$ stops growing, i.e. $F_n = F_{n-1}$
- Iff $\{ \text{Init} \} \sqsubseteq F \mathcal{A}$ is not universal.
Algorithm 0

Initialization

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

- We start with the antichain of the set of non accepting states
- \( F \leftarrow \{ \overline{\text{Fin}} \} \)
Initialization

- We start with the antichain of the set of non accepting states
- \( F \leftarrow \{\overline{\text{Fin}}\} \)
- \( \text{Frontier} \leftarrow F \)
Algorithm 1

First Iteration

- $s_1, s_2, s_3$ are $\text{cpre}_\sigma(s)$ for all $\sigma$ and all $s \in \text{Frontier}$
Algorithm 1

First Iteration

- $s_1, s_2, s_3$ are $\text{cpre}_\sigma(s)$ for all $\sigma$ and all $s \in \text{Frontier}$
- $\text{Frontier} = \text{CPre}^\mathcal{A} (\text{Frontier}) = \{s_1, s_2\}$
Algorithm 1

First Iteration

- $s_1, s_2, s_3$ are $cpre_\sigma(s)$ for all $\sigma$ and all $s \in \text{Frontier}$
- $\text{Frontier} = CPre^\mathcal{A}(\text{Frontier}) = \{s_1, s_2\}$
- $F \leftarrow F \sqcup \text{Frontier}$
Algorithm 2

Second Iteration

- $s_4, s_5, s_6$ are $\text{cpre}(s)$ for all $\sigma$ and all $s \in \text{Frontier}$
Algorithm 2

Second Iteration

- \( s_4, s_5, s_6 \) are \( cpre(s) \) for all \( \sigma \) and all \( s \in \text{Frontier} \)
- \( \text{Frontier} = CPre^\mathcal{A} (\text{Frontier}) = \{ s_4, s_6 \} \)
Second Iteration

- $s_4, s_5, s_6$ are $cpre(s)$ for all $\sigma$ and all $s \in \text{Frontier}$
- $\text{Frontier} = CPre^\sigma(\text{Frontier}) = \{s_4, s_6\}$
- $F \leftarrow F \sqcup \text{Frontier}$
Algorithm Termination

The Algorithm computes a series of antichains

\[ q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathcal{F} \] where \( q_i = CPre^\mathcal{A}(q_{i-1}) \sqcup \{\text{Fin}\} \)

Termination

- The Algorithm computes a series of antichains

\[ q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathcal{F} \] where \( q_i = CPre^\mathcal{A}(q_{i-1}) \sqcup \{\text{Fin}\} \)
The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathcal{F}$ where $q_i = CPre^\mathcal{A}(q_{i-1}) \cup \{\text{Fin}\}$.

Tarski’s Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint $F$. 

Termination
The Algorithm computes a series of antichains $q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathcal{F}$ where $q_i = \text{CPre}^\mathcal{A}(q_{i-1}) \cup \{\text{Fin}\}$.

Tarski’s Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint $F$.

$\text{Lang}(\mathcal{A}) \neq \Sigma^* \iff \{\text{Init}\} \sqsubseteq F$
The Algorithm computes a series of antichains
\[ q_0 \sqsubseteq q_1 \sqsubseteq \cdots \sqsubseteq q_n = \mathcal{F} \text{ where } q_i = \text{CPre}_{\mathcal{A}}(q_{i-1}) \cup \{\text{Fin}\} \]

Tarski’s Fixpoint Theorem implies that every monotone function on a complete lattice has a least fixpoint \( F \).

\[ \text{Lang}(\mathcal{A}) \neq \Sigma^* \iff \{\text{Init}\} \sqsubseteq F \]
Content

Basic Problem
Universality of NFA
Classical subset construction

Preliminaries
Predecessors on state sets
Lattice of Antichains
A monotone predecessor function on Antichains

Antichain Algorithm to check universality
The Algorithm at work
Antichain Algorithm vs. Classical
**Theorem**

For the family of $\mathcal{A}_k$, $k \geq 2$ 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 family of $A_k$, $k \geq 2$ with $k + 1$ states, the Backward Antichain Algorithm is polynomial in $k$, whereas the classical subset construction algorithm is exponential in $k$.
Classical DFA of $2^k + 1$ states, which are all accepting. Algorithm traverses the tree of $2^k$ accepting states with runtime exponential in $k$. 

February 2017 Felix Freyland – Fixpoint Antichain Algorithm 28 / 31
Classical

- DFA of $2^{k+1}$ states
Classical

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

- 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**

<table>
<thead>
<tr>
<th>Antichain Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Antichain Algorithm

- Starts with initial antichain $q_0 = \{\{k\}\}$
Comparison of Classical and Antichain Algorithm

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

- Starts with initial antichain $q_0 = \{\{k\}\}$
- In first iteration $q_1 = CPre(\{\{k\}\}) \sqcup \{\{k\}\} = \{\{k - 1, k\}\}$
- In each iteration:
  \[ q_{i+1} = CPre(q_i) \sqcup \{\{l_k\}\}, = \{\{k - (i + 1), k - i, \ldots, k\}\} \quad \text{for} \quad i < k \]
Comparison of Classical and Antichain Algorithm

Antichain Algorithm

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

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

- Stops after $k$ iterations with $q_k = q_{k-1} = \{ \{ 1, \ldots, k \} \}$
- Checks in each Iteration $\{ \{ l_0 \} \} \subseteq q_i$ in linear time
Antichain Algorithm

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