Antichain-based Inclusion Checking on Finite Nondeterministic Word and Tree Automata

Tomáš Vojnar
FIT, Brno University of Technology, Czech Republic
Plan of the Lecture

- Antichain-based **Universality** Checking on Word Automata
- Antichain-based **Upward Universality** Checking on Tree Automata
- Antichain-based **Inclusion** Checking on Word Automata
- Antichains and Simulations in **Inclusion** Checking on Word Automata
- Antichains and Simulations in **Upward Inclusion** Checking on Tree Automata
- Antichains and Simulations in **Downward Inclusion** Checking on Tree Automata
  - A separate presentation.
Universality Checking on Word Automata
Word Automata Universality

- Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.
- "Classic" approach: determinisation (subset construction), complementation, . . . .
- "On-the-fly" universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Word Automata Universality

- Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.
- “Classic” approach: determinisation (subset construction), complementation, . . . .
- “On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Word Automata Universality

- Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.
- “Classic” approach: determinisation (subset construction), complementation, . . . .
- “On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.

“Classic” approach: determinisation (subset construction), complementation, . . . .

“On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Universality and inclusion are \textbf{PSPACE-complete} for NFA, \textbf{EXPTIME-complete} for TA.

“Classic” approach: determinisation (subset construction), complementation, . . . .

“On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.

“Classic” approach: determinisation (subset construction), complementation, . . . .

“On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:
Word Automata Universality

- Universality and inclusion are **PSPACE-complete** for NFA, **EXPTIME-complete** for TA.
- “Classic” approach: determinisation (subset construction), complementation, . . . .
- “On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:

![Subset Construction Diagram]

- Antichain-based universality checking for word automata:
  - [Doyen, Henzinger, and Raskin – CAV’06],
  - Keep only the states of the subset automaton needed for proving universality.
A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.
A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.
A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.
A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.

Given a set $S$ partially ordered by $\geq$, an antichain over $S$ is any $A \subseteq S$ such that for any $r, s \in A$, neither $r \leq s$ nor $r \geq s$.

Antichains for universality: subsets of $2^Q$ ordered by $\subseteq$. 
Backward Antichain-based Universality

- Backward antichain-based universality – a dual construction:
  - start with non-final states,
  - compute controllable predecessors,
    - sets of predecessors that cannot continue outside of the given set,
  - try to cover initial states,
  - smaller sets can be discarded.
Universality Checking on Tree Automata
The described forward antichain construction for word automata smoothly carries over to an upward antichain construction on NTA.

The only difference is in how the subset construction (i.e., the computation of new states) is done.

Word case
\[q \xrightarrow{a} t\]
\[r \xrightarrow{a} u\]
\[s \xrightarrow{a} v\]

Tree case
\[(q, s, u) \xrightarrow{a} x\]
\[(r, t, v) \xrightarrow{a} y\]

Downward universality for TA cannot be done as a simple generalization of backward universality on NFA: dealing with tuples of tuples of ... of states!
Inclusion Checking on Word Automata
**Classical Inclusion Checking on FA**

- The classical approach to checking $L(A) \subseteq L(B)$:
  - check emptiness of $A \cap \text{determinize}_{\text{using_subset_construction}} B$,

![Diagram of classical inclusion checking](image_url)
Classical Inclusion Checking on FA

- The classical approach to checking $L(A) \subseteq L(B)$:
  - check emptiness of $A \cap \text{determinize}_{\text{using_subset_construction}} B$,
  - can involve minimisation of determinised automata: not a good solution anyway,
Classical Inclusion Checking on FA

- The classical approach to checking $L(A) \subseteq L(B)$:
  - check emptiness of $A \cap \text{determinize}_{\text{using_subset_construction}} B$,

- can involve minimisation of determinised automata: not a good solution anyway,

- The constructed product automaton is built of macro-states $(r, P)$ such that:
  - if some $w$ can reach $r$ in $A$, $P$ is the set of all states reached by $w$ in $B$,
  - $(r, P)$ is accepting iff $r \in F_A$ and $P \cap F_B = \emptyset$. 

Antichain-based Inclusion on NFA and NTA – p.10/23
On-the-Fly Inclusion Checking

- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from \( (q_0^A, \{q_0^B\}) \))
    - while checking for reachability of an accepting macro-state.
On-the-Fly Inclusion Checking

The first possible optimisation:

- do not determinise, then complement, then compose, then check emptiness,
- instead do all the steps at the same time:
  - incrementally generate reachable macro-states (starting from \((q_0^A, \{q_0^B\})\))
  - while checking for reachability of an accepting macro-state.

\[ (r, \{p\}) \]
On-the-Fly Inclusion Checking

- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from \((q_0^A, \{q_0^B\})\))
    - while checking for reachability of an accepting macro-state.

![Diagram](image-url)
On-the-Fly Inclusion Checking

- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from \( q_0^A, \{ q_0^B \} \))
    - while checking for reachability of an accepting macro-state.

![Diagram of a DFA and its transitions]
On-the-Fly Inclusion Checking

The first possible optimisation:

- do not determinise, then complement, then compose, then check emptiness,
- instead do all the steps at the same time:
  - incrementally generate reachable macro-states (starting from \((q_0^A, \{q_0^B\})\))
  - while checking for reachability of an accepting macro-state.
On-the-Fly Inclusion Checking

- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from \((q_0^A, \{q_0^B\})\))
    - while checking for reachability of an accepting macro-state.

- Can be stopped as soon as a counterexample to inclusion is found.
  - No improvement when the inclusion holds, but a basis for further optimisations.
On-the-Fly Inclusion with Antichains

[De Wulf, Doyen, Henzinger, Raskin – CAV’06]

❖ For the same left component, keep only those macro-states whose right components are mutually incomparable wrt. inclusion (and hence antichains).

❖ If \((p, R_1)\) and \((p, R_2)\) such that \(R_1 \subseteq R_2\) are generated, discard \((p, R_2)\).
  • Indeed, if a counterexample to the inclusion query can be found from \((p, R_2)\), a counterexample can be found from \((p, R_1)\) too.
On-the-Fly Inclusion with Antichains

[De Wulf, Doyen, Henzinger, Raskin – CAV’06]

- For the same left component, keep only those macro-states whose right components are mutually incomparable wrt. inclusion (and hence antichains).

- If \((p, R_1)\) and \((p, R_2)\) such that \(R_1 \subseteq R_2\) are generated, discard \((p, R_2)\).
  - Indeed, if a counterexample to the inclusion query can be found from \((p, R_2)\), a counterexample can be found from \((p, R_1)\) too.
On-the-Fly Inclusion with Antichains

[De Wulf, Doyen, Henzinger, Raskin – CAV’06]

❖ For the same left component, keep only those macro-states whose right components are mutually incomparable wrt. inclusion (and hence antichains).

❖ If \((p, R_1)\) and \((p, R_2)\) such that \(R_1 \subseteq R_2\) are generated, discard \((p, R_2)\).

  • Indeed, if a counterexample to the inclusion query can be found from \((p, R_2)\), a counterexample can be found from \((p, R_1)\) too.
Antichains for Universality x Inclusion

- **Universality:**
  - Antichains over $2^Q$ with $\subseteq$.
  - $\{q_1, \ldots, q_n\} \subseteq 2^Q$ is reachable. $\iff$ $q_1, \ldots, q_n$ are all the states in which the automaton $A$ can end up after reading some word $w$.
  - Is any $S \subseteq Q \setminus F$ reachable?

- **Inclusion:** $L(A) \subseteq L(B)$
  - Antichains over $Q_A \times 2^{Q_B}$ with $= \times \subseteq$.
  - $(r, \{q_1, \ldots, q_n\})$ is reachable. $\iff$ After reading some word $w$, $A$ can end up in a state $r$ and $B$ ends up in one of $q_1, \ldots, q_n$.
  - Is any $S \subseteq F_A \times 2^{Q_B \setminus F_B}$ reachable?
Experiments with Antichains

- Determinisation-based and antichain-based inclusion checking on TA from ARTMC:

![Graph showing time (s) vs. number of states for determinisation-based and antichain-based methods.](image)
Antichains and Simulations in Inclusion Checking on Word Automata
Simulation and Inclusion Checking

- Simulation cannot be directly used for checking inclusion:
  - If $q_0^A \xrightarrow{F} q_0^B$, then $L(A) \subseteq L(B)$, but the converse does not hold!
Simulation and Inclusion Checking

- Simulation cannot be directly used for checking inclusion:
  - If $q_0^A F q_0^B$, then $L(A) \subseteq L(B)$, but the converse does not hold!
  - Can be used as an auxiliary incomplete test only.

- One can compute antichains on simulation-reduced automata,
  - but this requires using simulation equivalence,
  - which means taking a symmetric restriction,
  - which is not nice for a problem as asymmetric as inclusion checking,
  - the obtained reduction is unnecessarily diminished.
Simulation Meets Antichains (1)

[Aabdulla, Chen, Holík, Mayr, V. – TACAS’10], [Doyen, Raskin – TACAS’10]

❖ A macro-state \((p, P)\) needs not be explored if:

1. there is a macro-state \((r, R)\) such that \(p \sim r\) and \(\forall r' \in R \exists p' \in P : r' \sim p'\),
   • intuitively, \(p\) is less “accepting” than \(r\) while \(P\) is more “accepting” than \(R\),
Simulation Meets Antichains (1)

[Abdulla, Chen, Holík, Mayr, V. – TACAS’10], [Doyen, Raskin – TACAS’10]

❖ A macro-state \((p, P)\) needs not be explored if:

1. there is a macro-state \((r, R)\) such that \(p \not\leq r\) and \(\forall r' \in R \exists p' \in P : r' \not\leq p'\),
   - intuitively, \(p\) is less “accepting” than \(r\) while \(P\) is more “accepting” than \(R\),

2. \(\exists p' \in P : p \not\leq p'\),
   - intuitively, \(p\) cannot even “beat” \(p'\) alone.
A macro-state \((p, P)\) needs not be explored if:

1. there is a macro-state \((r, R)\) such that \(p \not F r\) and \(\forall r' \in R \exists p' \in P : r' \not F p'\),
   - intuitively, \(p\) is less “accepting” than \(r\) while \(P\) is more “accepting” than \(R\),
2. \(\exists p' \in P : p \not F p'\),
   - intuitively, \(p\) cannot even “beat” \(p'\) alone.
A macro-state \((p, P)\) needs not be explored if:

1. there is a macro-state \((r, R)\) such that \(p F r\) and \(\forall r' \in R \exists p' \in P : r' F p'\),
   - intuitively, \(p\) is less “accepting” than \(r\) while \(P\) is more “accepting” than \(R\),
2. \(\exists p' \in P : p F p'\),
   - intuitively, \(p\) cannot even “beat” \(p'\) alone.
A macro-state \((p, P)\) needs not be explored if:

1. there is a macro-state \((r, R)\) such that \(p \sim r\) and \(\forall r' \in R \exists p' \in P : r' \sim p'\),
   - intuitively, \(p\) is less “accepting” than \(r\) while \(P\) is more “accepting” than \(R\),
2. \(\exists p' \in P : p \sim p'\),
   - intuitively, \(p\) cannot even “beat” \(p'\) alone.
Simulation Meets Antichains (2)

- Another simulation-based optimisation is to prune the sets in product states:
  - \((p, Q)\) can be replaced by \((p, Q \setminus \{q_1\})\) whenever \(\exists q_2 \in Q \setminus \{q_1\} : q_1 F q_2\).
  - Intuitively, \(q_1\) cannot contribute anything compared to \(q_2\).

- One can also combine backward antichains with backward simulations.

- Even combinations of forward antichains and backward simulations (and vice versa) are possible, but such combinations do not improve the computation [Doyen, Raskin – TACAS’10].
Some Experimental Results

- Language inclusion checking on NFAs generated from ARMC:

![Graph showing experimental results with Antichain and Simulation markers.](image-url)
Antichains and Simulations in Upward Inclusion Checking on Tree Automata
Tree Antichains

[Bouajjani, Habermehl, Holík, Touili, V. – CIAA’08]

- For tree automata, an upward antichain construction may be used:
  - Start with leaf rules.
  - To compute successors via \( n \)-ary rules, take all \( n \)-tuples of generated macro-states \((p_1, R_1), \ldots, (p_n, R_n)\) and
    - on the \( A \) part, iterate through all rules \((p_1, \ldots, p_n) \xrightarrow{a} p,\)
    - for each of them, on the \( B \) part, consider all rules \((r_1, \ldots, r_n) \xrightarrow{a} r\) where \( r_i \in R_i \) for \(1 \leq i \leq n\).
Tree antichains are built by computing successors of tuples of macro-states, which amounts to computing successors of tuples of states on the left and right of macro-states:

\[ (p_3, \{q_5, q_6, q_7, \ldots\}) \]

\[ (p_1, \{q_1, q_2\}) \quad (p_2, \{q_3, q_4\}) \]

A crucial notion is the set (language) of trees accepted from a given tuple of states.

A suitable simulation \( S \) to be combined with upward antichains should respect languages of tuples of trees:

- If \( p_i S r_i \) for some \( 1 \leq i \leq n \), then \( \mathcal{L}(p_1, \ldots, p_n) \subseteq \mathcal{L}(r_1, \ldots, r_n) \).

- For this, we may require: If \( p S r \), then whenever \( (q_1, \ldots, q_i = p, \ldots, q_n) \xrightarrow{a} q' \), then also \( (q_1, \ldots, q_i = r, \ldots, q_n) \xrightarrow{a} r' \) where \( p' S r' \).
  - This leads to \( S = U_{Id} \)!
  - Upward simulations induced by larger simulations are not suitable.
Some Experimental Results

Language inclusion checking on TA generated from ARTMC:

<table>
<thead>
<tr>
<th>Size</th>
<th>Antichains (sec.)</th>
<th>Simulation (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 200</td>
<td>1.05</td>
<td>0.75</td>
</tr>
<tr>
<td>200 – 400</td>
<td>11.7</td>
<td>4.7</td>
</tr>
<tr>
<td>400 – 600</td>
<td>65.2</td>
<td>19.9</td>
</tr>
<tr>
<td>600 – 800</td>
<td>3019.3</td>
<td>568.7</td>
</tr>
<tr>
<td>800 – 1000</td>
<td>4481.9</td>
<td>840.4</td>
</tr>
<tr>
<td>1000 – 1200</td>
<td>11761.7</td>
<td>1720.9</td>
</tr>
</tbody>
</table>