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.

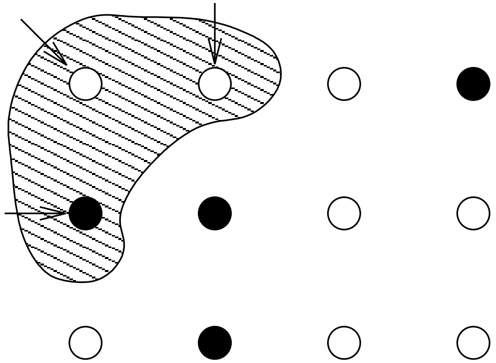
Universality Checking on Word Automata

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:

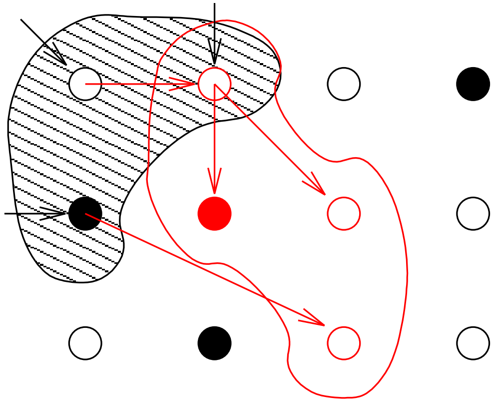
Word Automata Universality

- 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:



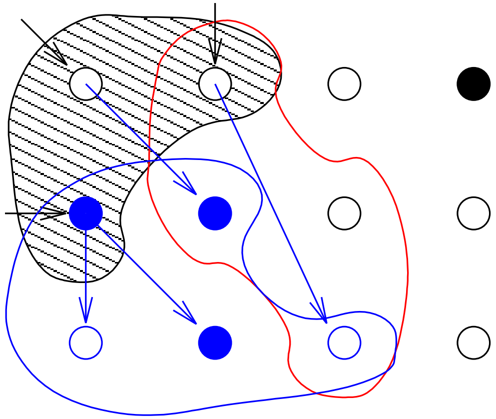
Word Automata Universality

- 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:



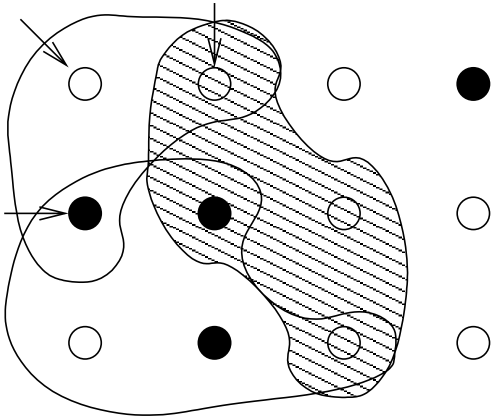
Word Automata Universality

- 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:

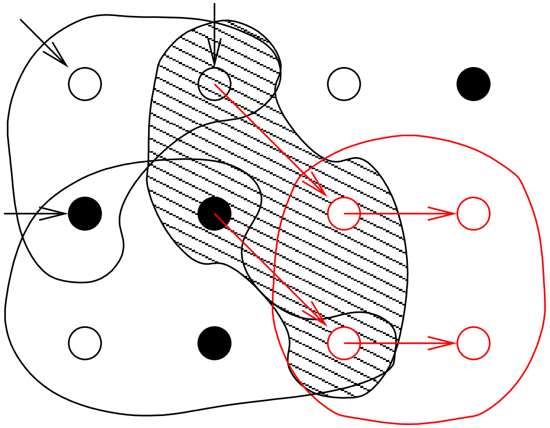


Word Automata Universality

- Universality and inclusion are PSPACE-complete for NFA, EXPTIME-complete for TA.

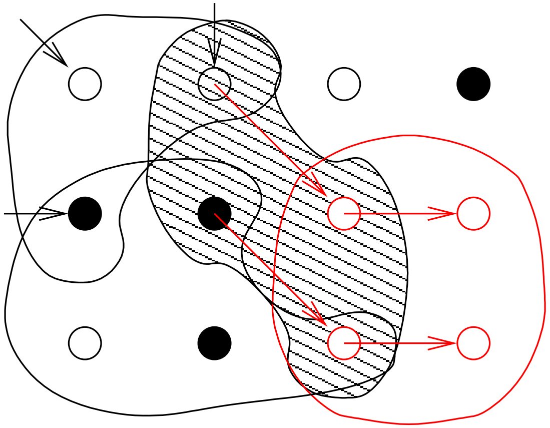
- “Classic” approach: determinisation (subset construction), complementation, ...

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



Word Automata Universality

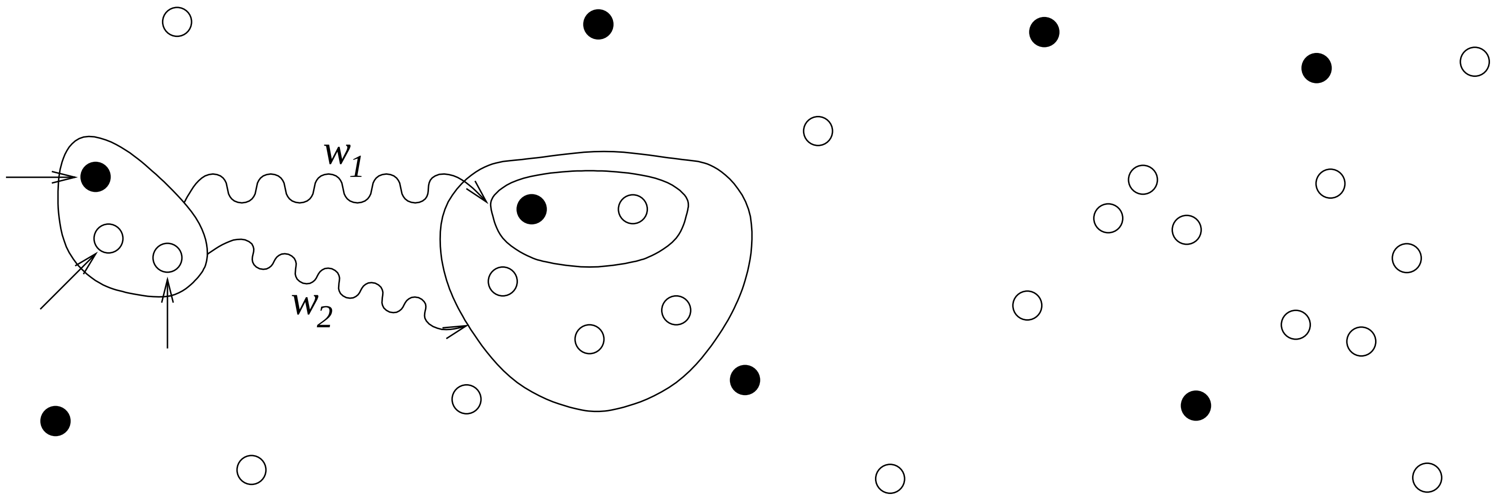
- Universality and inclusion are PSPACE-complete for NFA, EXPTIME-complete for TA.
- “Classic” approach: determinisation (subset construction), complementation, ...
- “On-the-fly” universality checking during subset construction – can be stopped as soon as a non-accepting set gets generated:



- 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.

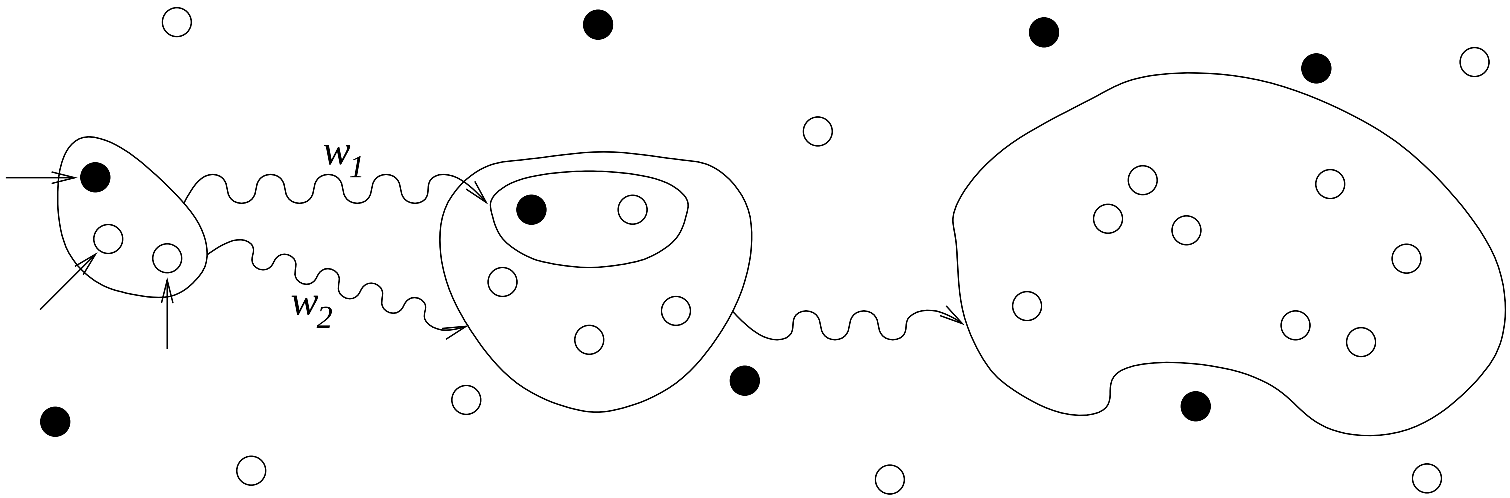
Antichains in the Subset Construction

- A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.



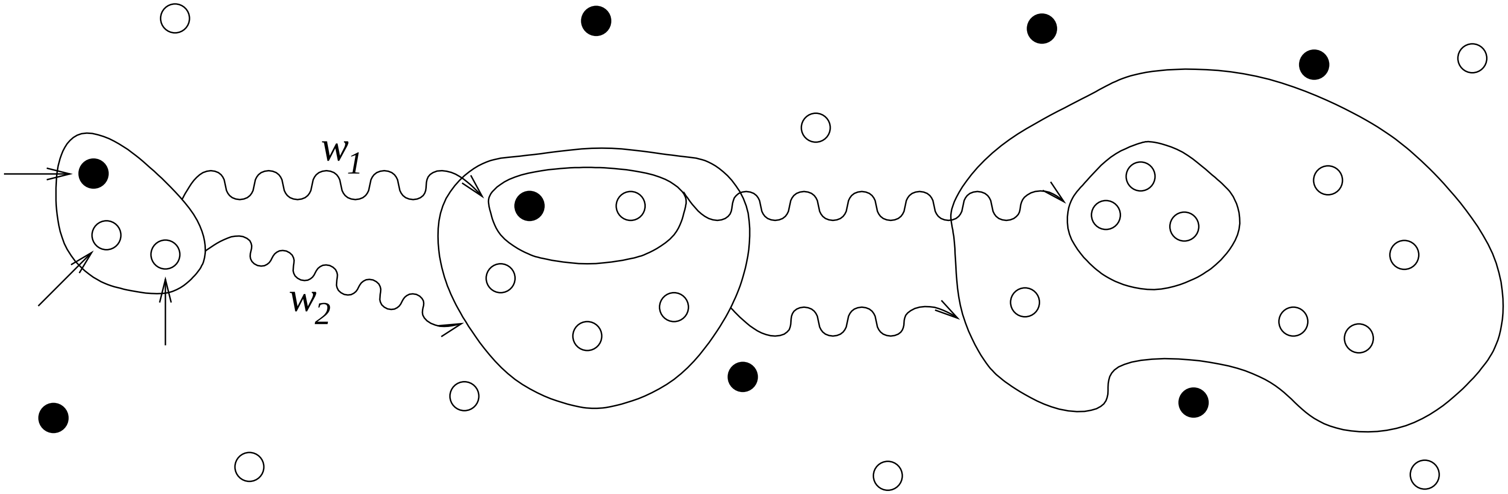
Antichains in the Subset Construction

- A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.



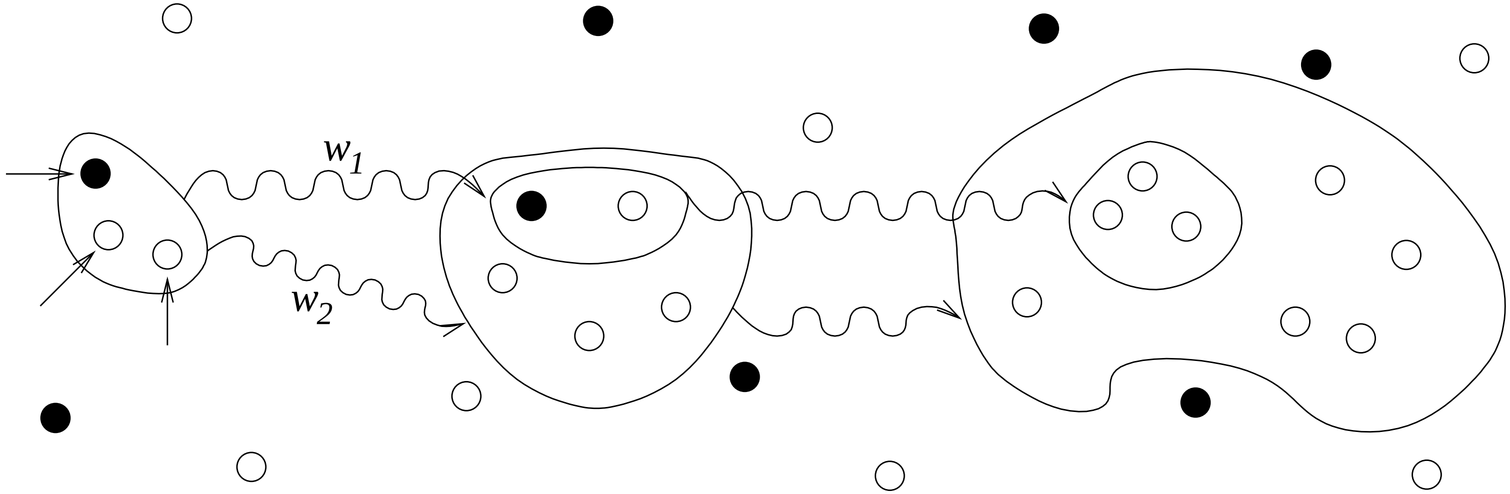
Antichains in the Subset Construction

- A key observation: We do not need to keep computed subsets of states that are supersets of other computed subsets.



Antichains in the Subset Construction

❖ 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 $\subseteq$.

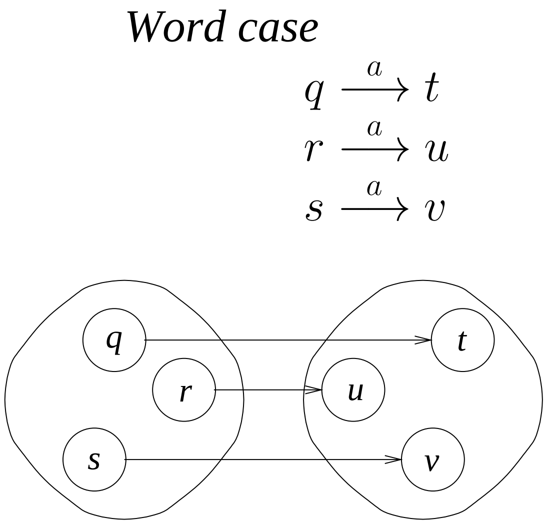
Backward Antichain-based Universality

- 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

Antichains for Tree Universality

- 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.

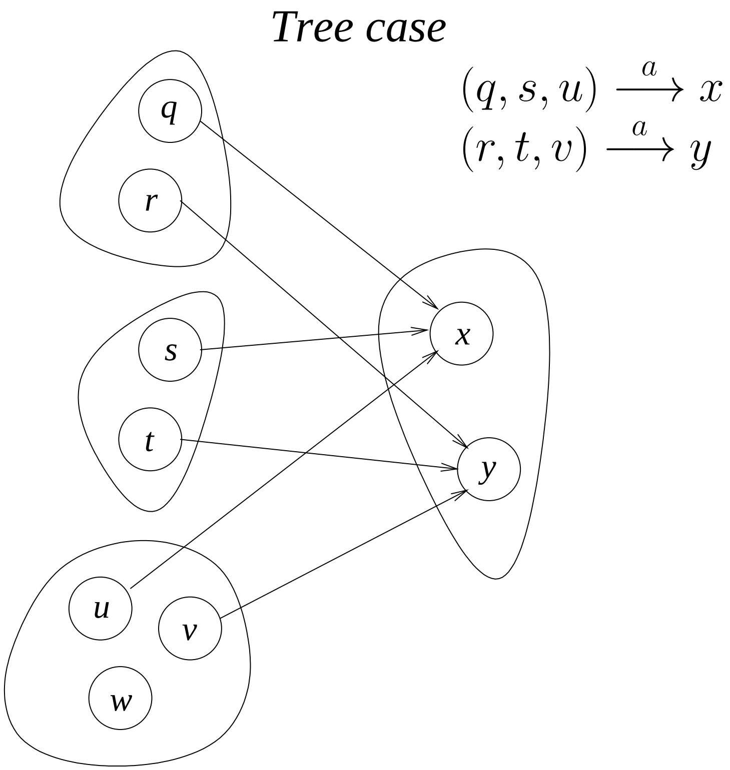


Word case

$q \xrightarrow{a} t$

$r \xrightarrow{a} u$

$s \xrightarrow{a} v$



Tree case

$(q, s, u) \xrightarrow{a} x$

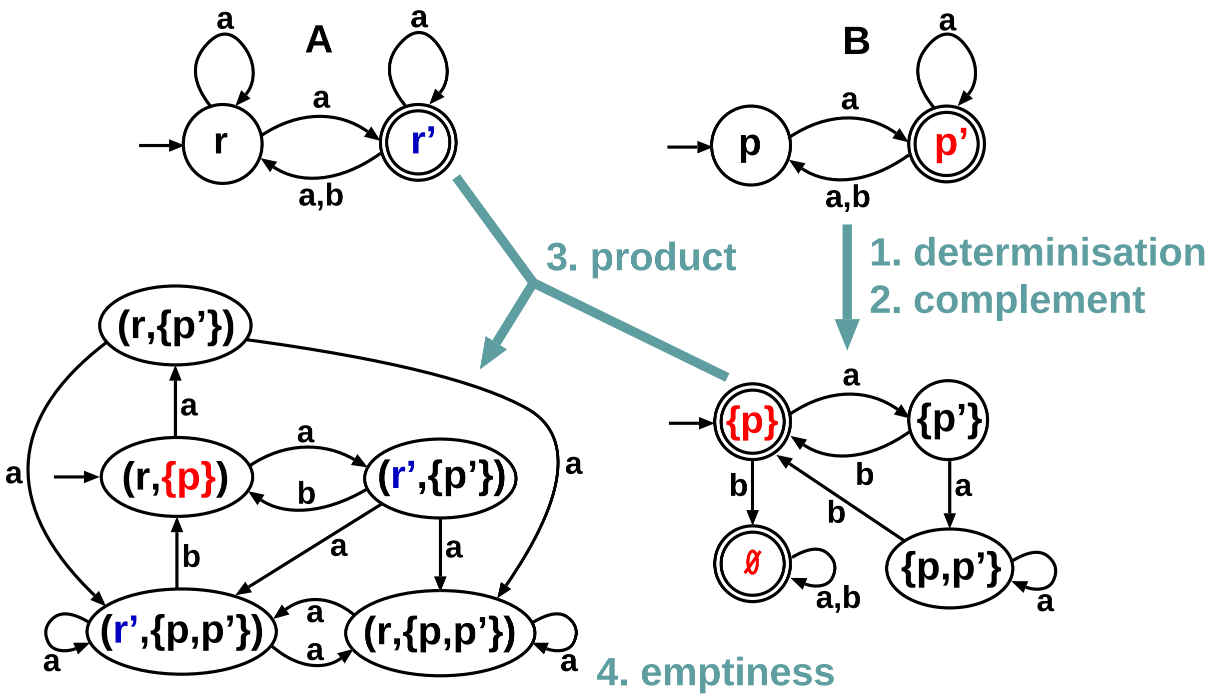
$(r, t, v) \xrightarrow{a} y$

- 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 $A \cap \overline{\text{determinize}_{\text{using\_subset\_construction}}\ B}$,



3. product

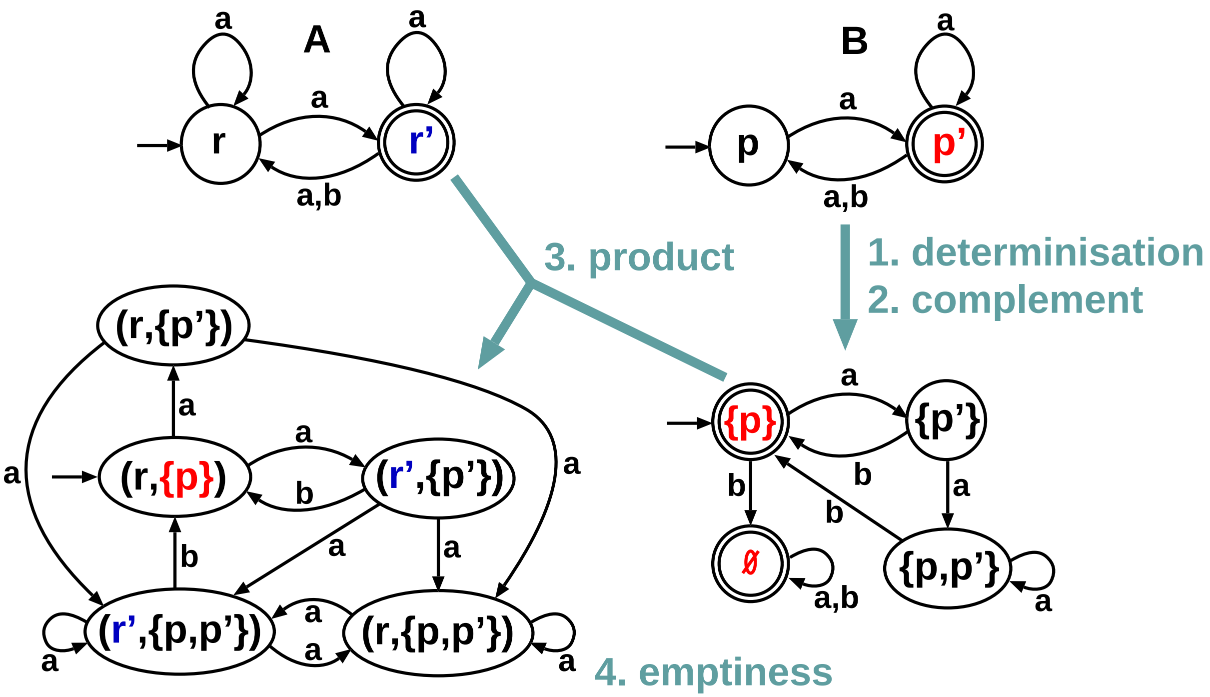
1. determinisation
2. complement

4. emptiness

Classical Inclusion Checking on FA

❖ The classical approach to checking $L(A) \subseteq L(B)$:

- check emptiness of $A \cap \overline{\text{determinize}_{\text{using\_subset\_construction}}\ B}$,



3. product

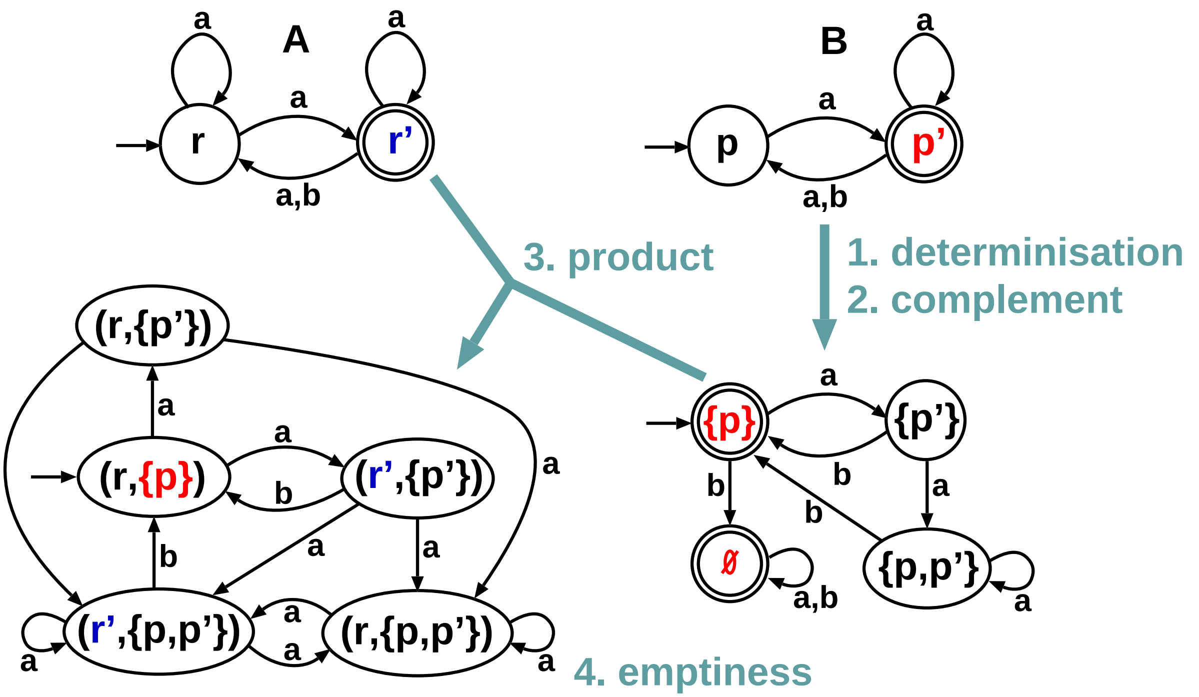
1. determinisation
2. complement

4. emptiness

- 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 $A \cap \overline{\text{determinize}_{\text{using\_subset\_construction}}\ B}$,



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

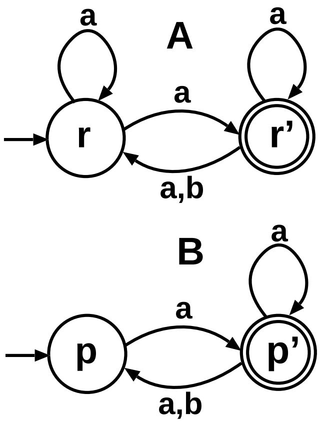
- The constructed product automaton is built of macro-states (r, P) such that:
  - if some w can reach r in A, P is the set of all states reached by w in B,
  - (r, P) is accepting iff $r \in F_A$ and $P \cap F_B = \emptyset$.

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

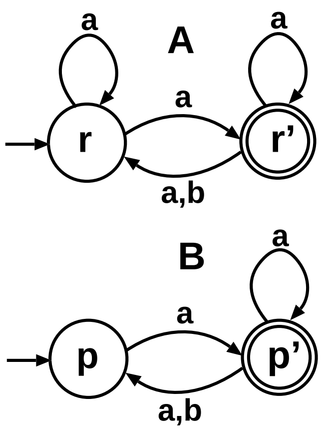
- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from $(q_0^A, \{q_0^B\})$)
    - while checking for reachability of an accepting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

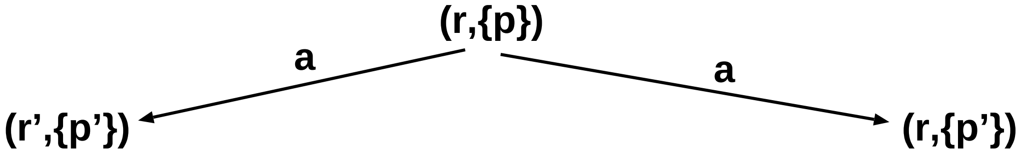


(r,{p})

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.




(r,{p})

a

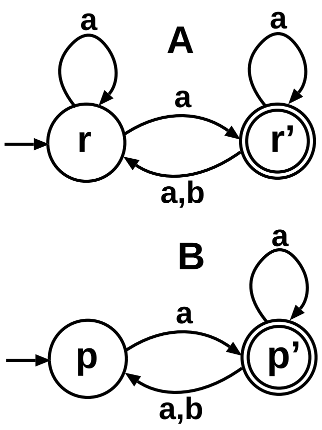
a

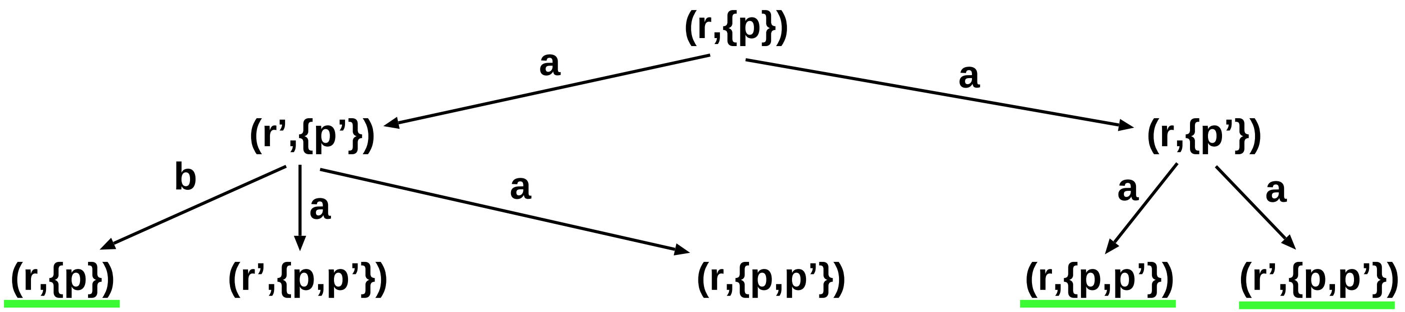
(r’,{p’})

(r,{p’})

On-the-Fly Inclusion Checking

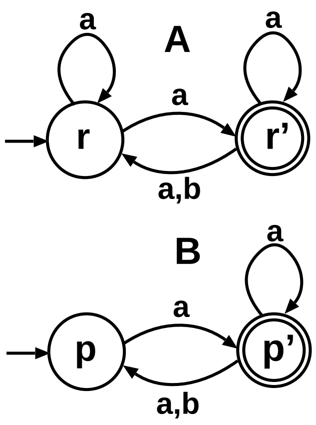
- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from $(q_0^A, \{q_0^B\})$)
    - while checking for reachability of an accepting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

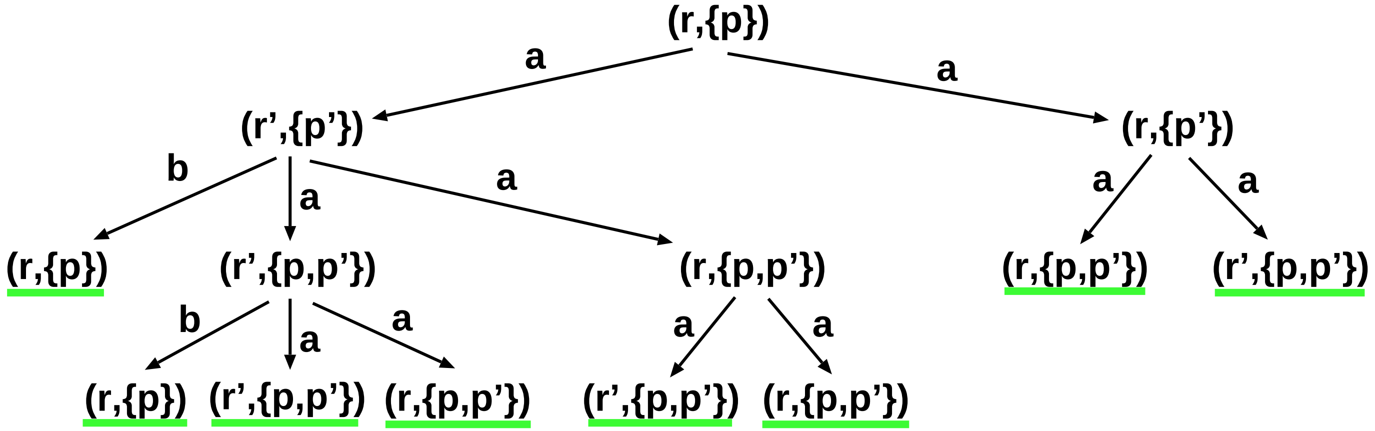




On-the-Fly Inclusion Checking

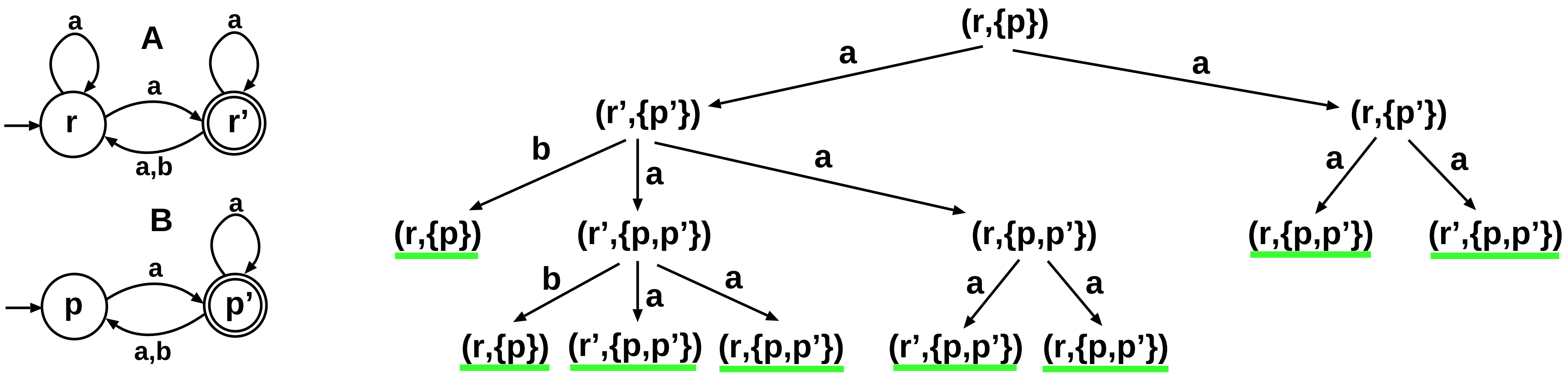
- The first possible optimisation:
  - do not determinise, then complement, then compose, then check emptiness,
  - instead do all the steps at the same time:
    - incrementally generate reachable macro-states (starting from $(q_0^A, \{q_0^B\})$)
    - while checking for reachability of an accepting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

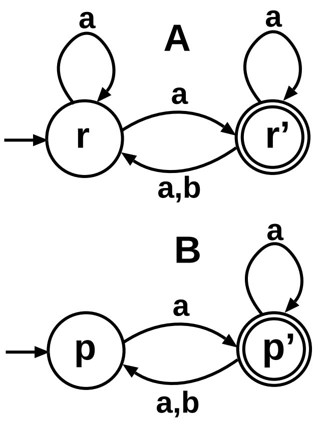
- If (p, R_1) and (p, R_2) such that $R_1 \subseteq R_2$ are generated, discard (p, R_2).
  - Indeed, if a counterexample to the inclusion query can be found from (p, R_2), a counterexample can be found from (p, R_1) too.

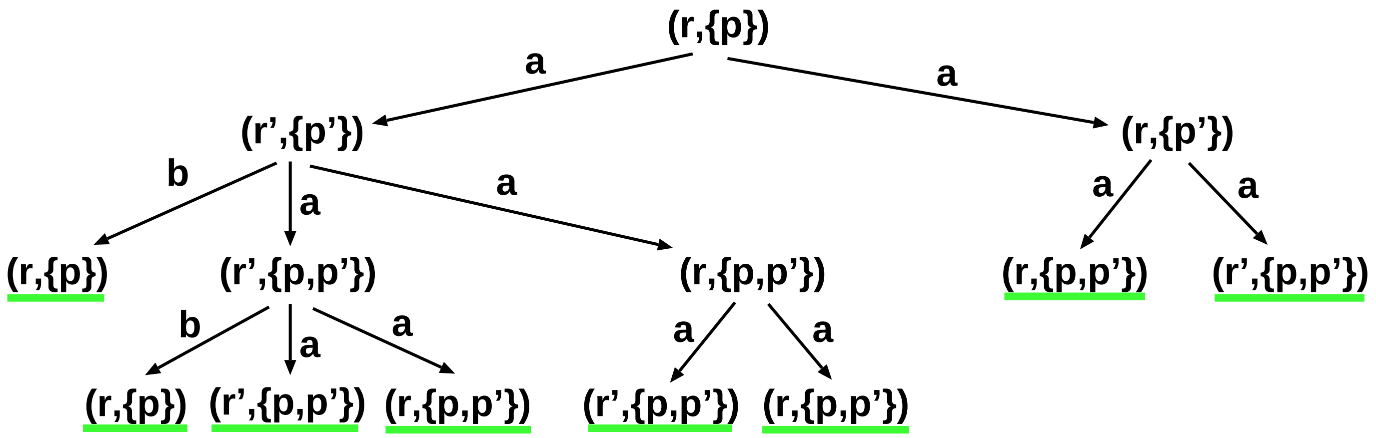
On-the-Fly Inclusion with Antichains

[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.

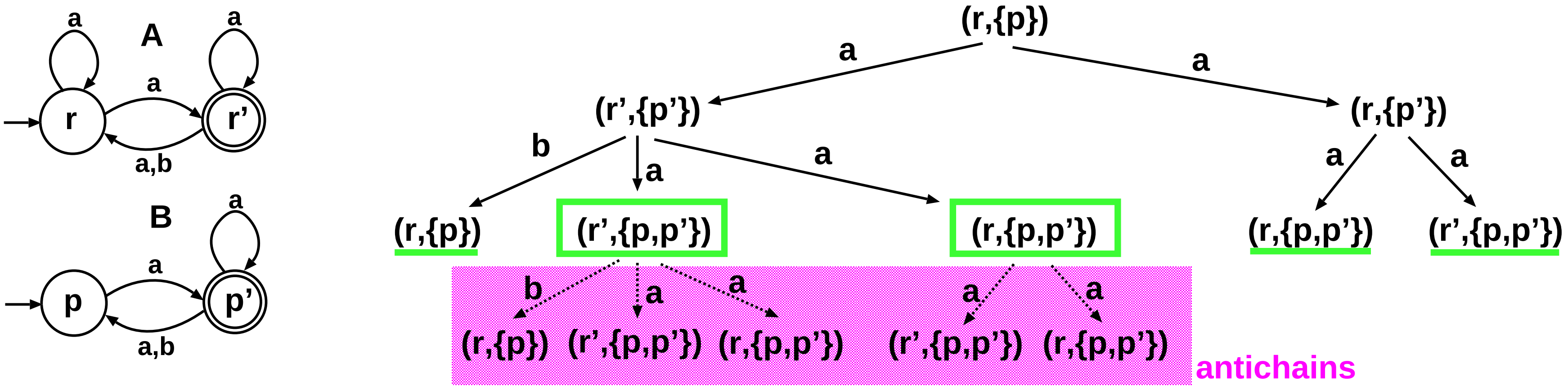




On-the-Fly Inclusion with Antichains

[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.



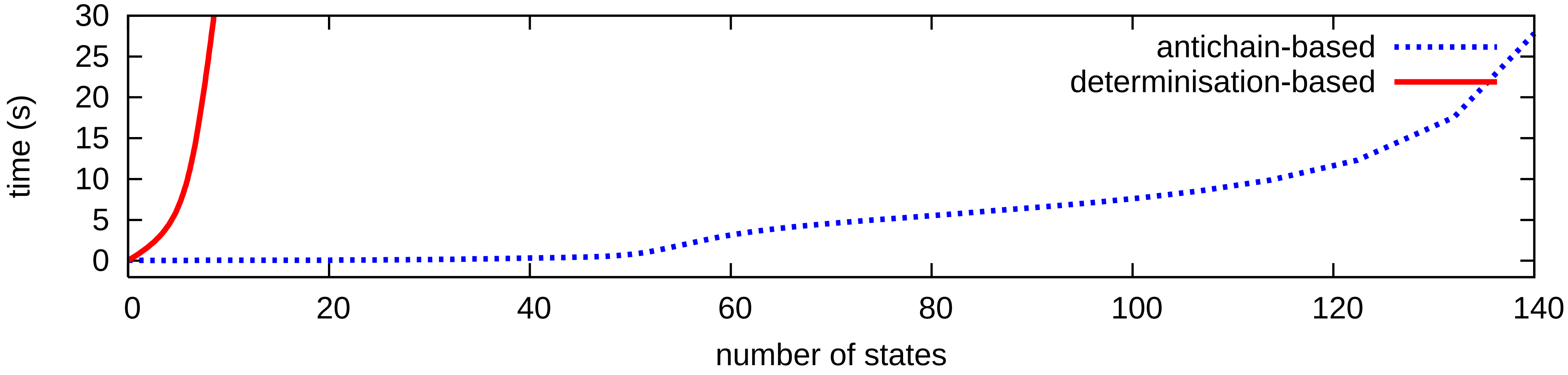
Antichains for Universality x Inclusion

- Universality:
  - Antichains over 2^Q with $\subseteq$.
  - $\{q_1, \ldots, q_n\} \subseteq 2^Q$ is reachable. $\Longleftrightarrow$ $q_1, \ldots, q_n$ 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: $L(A) \overset{?}{\subseteq} L(B)$
  - Antichains over $Q_A \times 2^{Q_B}$ with $= \times \subseteq$.
  - $(r, \{q_1, \ldots, q_n\})$ is reachable. $\Longleftrightarrow$ 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:



Antichains and Simulations in Inclusion Checking on Word Automata

Simulation and Inclusion Checking

- Simulation cannot be directly used for checking inclusion:
  - If $q_0^A \; F \; q_0^B$, 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)

[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 $p \; F \; r$ 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,

Simulation Meets Antichains (1)

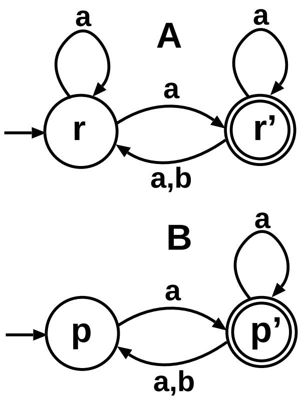
[Abdulla, Chen, Holík, Mayr, V. – TACAS'10], [Doyen, Raskin – TACAS'10]

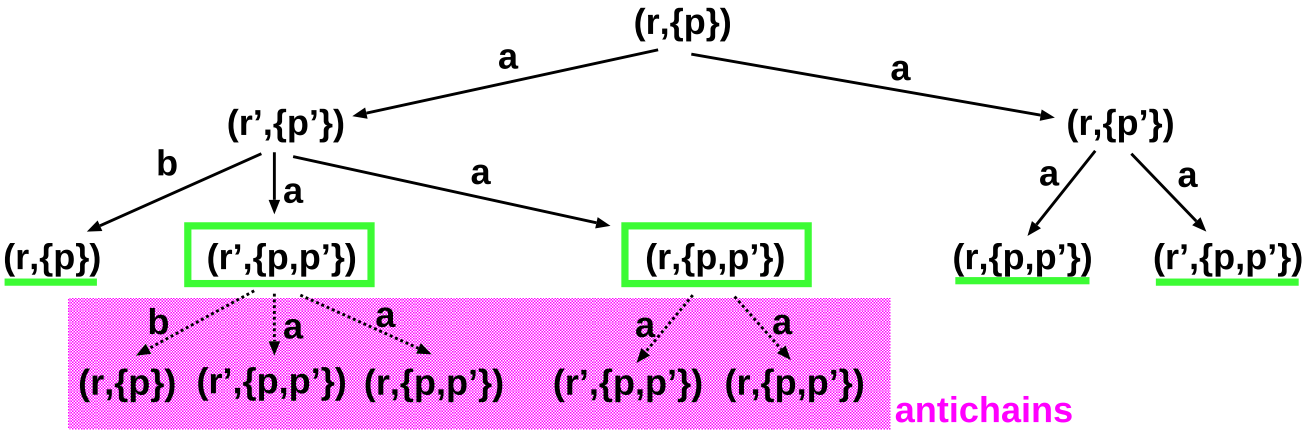
- A macro-state (p, P) needs not be explored if:
  1. there is a macro-state (r, R) such that $p\ F\ r$ and $\forall r' \in R\ \exists p' \in P :\ r'\ F\ p'$,
     - intuitively, p is less "accepting" than r while P is more "accepting" than R,
  2. $\exists p' \in P :\ p\ F\ p'$,
     - intuitively,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

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 $p \; F \; r$ 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.



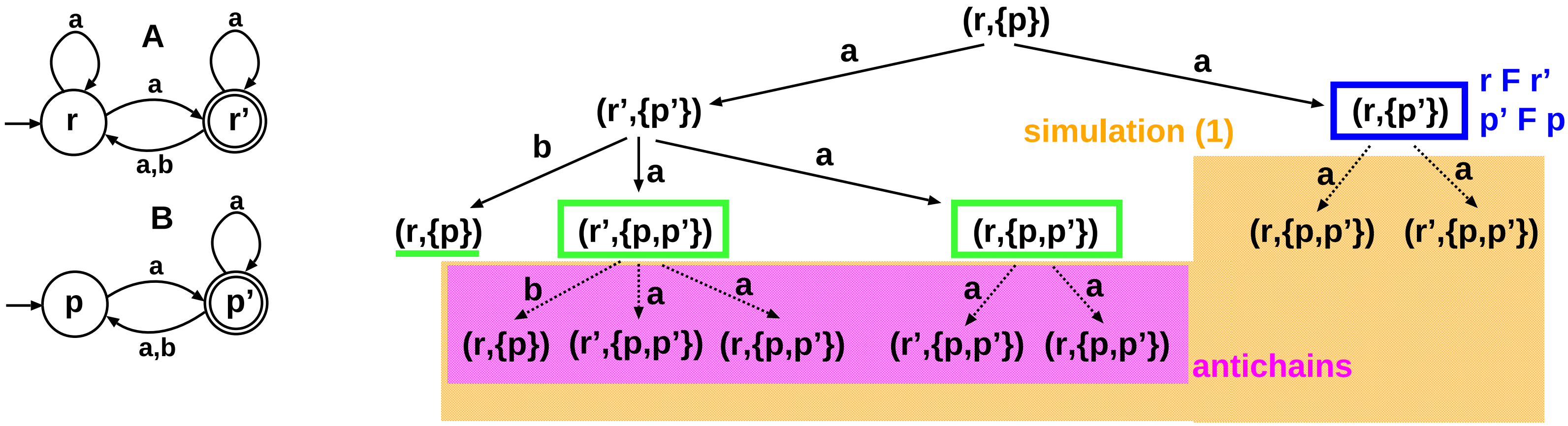


antichains

Simulation Meets Antichains (1)

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

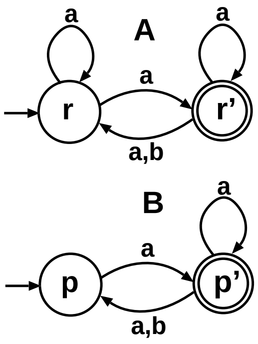
- A macro-state (p, P) needs not be explored if:
  1. there is a macro-state (r, R) such that $p\ F\ r$ and $\forall r' \in R\ \exists p' \in P:\ r'\ F\ p'$,
     - intuitively, p is less "accepting" than r while P is more "accepting" than R,
  2. $\exists p' \in P:\ p\ F\ p'$,
     - intuitively, p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

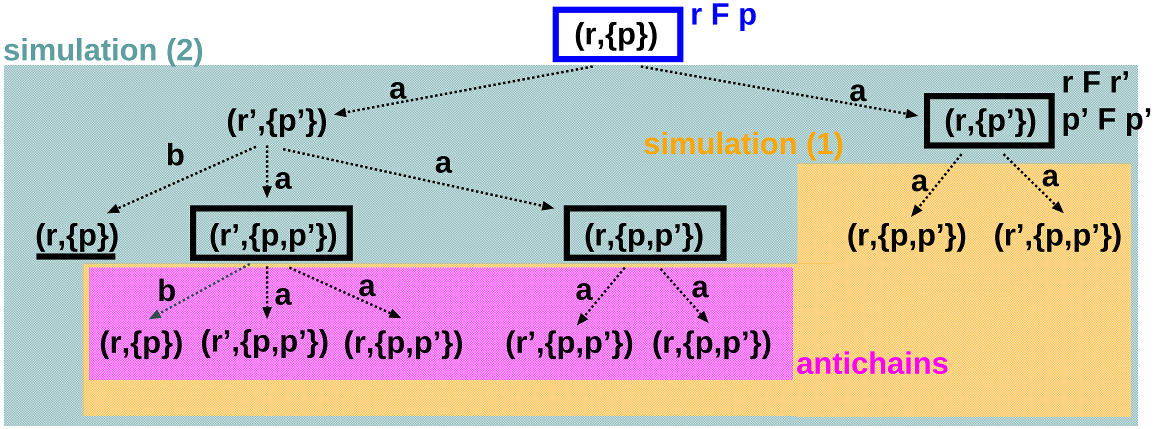


Simulation Meets Antichains (1)

[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 $p\ F\ r$ 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.





Simulation Meets Antichains (2)

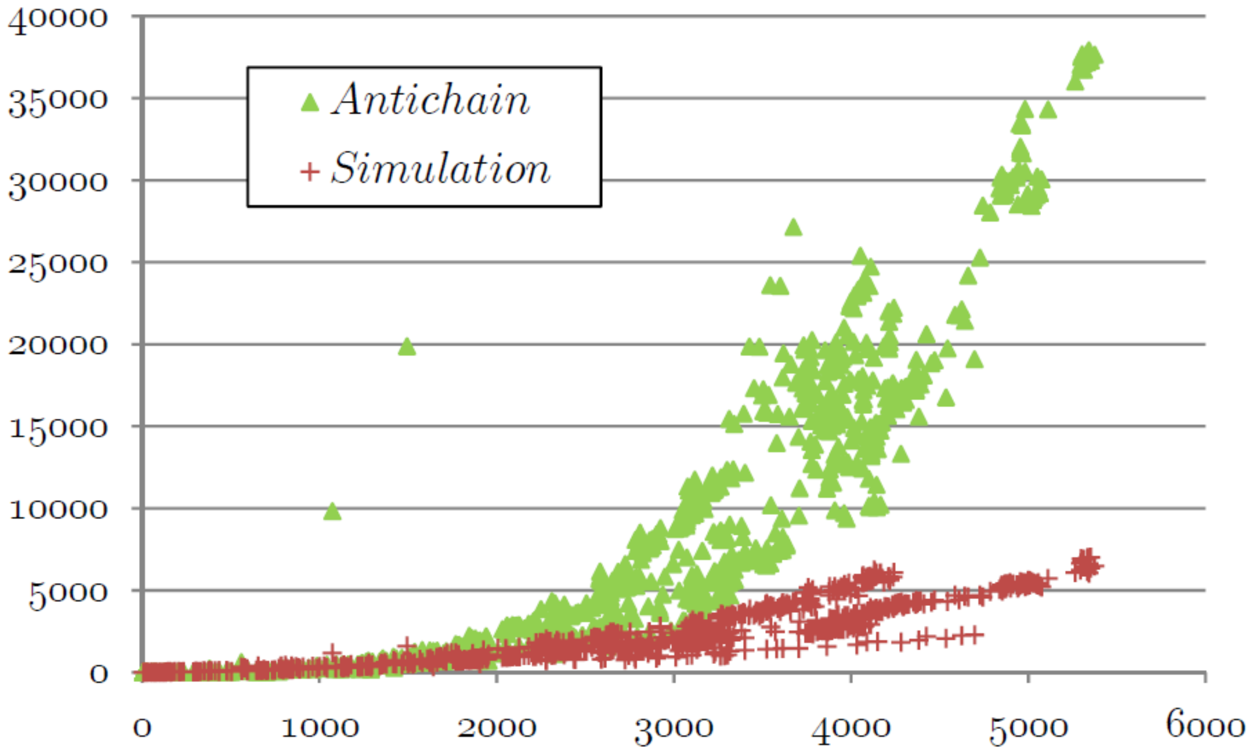
- Another simulation-based optimisation is to prune the sets in product states:
  - (p, Q) can be replaced by $(p, Q \setminus \{q_1\})$ whenever $\exists q_2 \in Q \setminus \{q_1\} : \; q_1 \; F \; q_2$.
  - Intuitively, q_1 cannot contribute anything compared to q_2.

- 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:

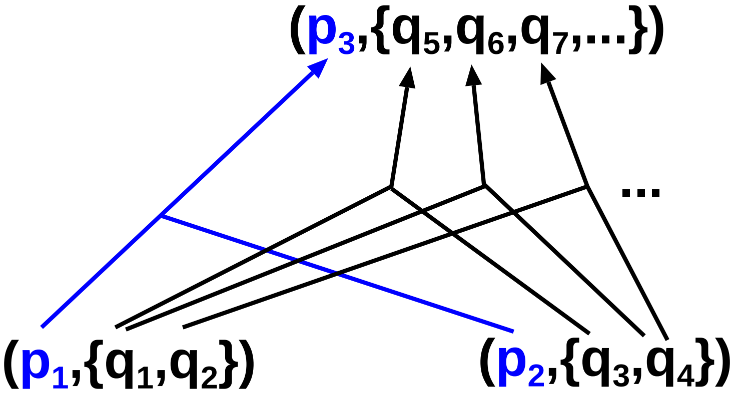


Antichains and Simulations in Upward Inclusion Checking on Tree Automata

Tree Antichains

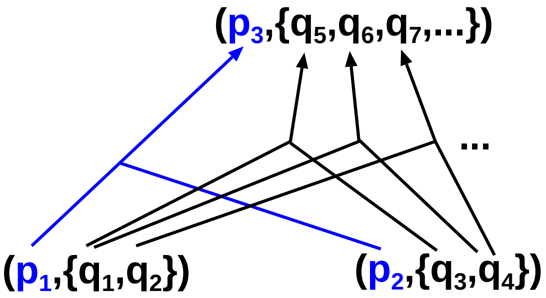
[Bouajjani, Habermehl, Holík, Touili, V. – CIAA'08]

- 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),..., (p_n, R_n) and
    - on the A part, iterate through all rules $(p_1, ..., p_n) \xrightarrow{a} p$,
    - for each of them, on the B part, consider all rules $(r_1, ..., r_n) \xrightarrow{a} r$ where $r_i \in R_i$ for $1 \leq i \leq n$.



Simulation Meets Antichains in Trees

❖ 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:



❖ 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 $\mathcal{L}((p_1,...,p_n)) \subseteq \mathcal{L}((r_1,...,r_n))$.
- For this, we may require: If $p \; S \; r$, then whenever $(q_1,...,q_i = p,...,q_n) \xrightarrow{a} q'$, then also $(q_1,...,q_i = r,...,q_n) \xrightarrow{a} r'$ 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:

Size	Antichains (sec.)	Simulation (sec.)
0 – 200	1.05	0.75
200 – 400	11.7	4.7
400 – 600	65.2	19.9
600 – 800	3019.3	568.7
800 – 1000	4481.9	840.4
1000 – 1200	11761.7	1720.9