Efficient Inclusion Checking over Tree Automata

Lukáš Holík[1,2] **Ondřej Lengál**[1] Jiří Šimáček[1,3] Tomáš Vojnar[1]

[1]Brno University of Technology, Czech Republic
[2]Uppsala University, Sweden
[3]VERIMAG, UJF/CNRS/INPG, Gières, France

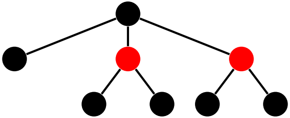
October 28, 2012

Outline

Trees

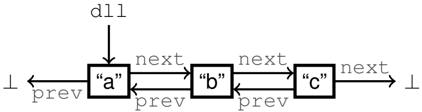
Very popular in computer science:

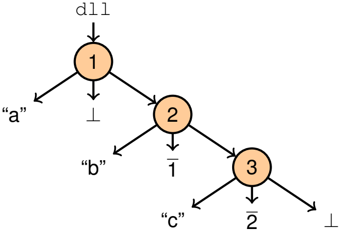
- data structures,
- computer network topologies,
- distributed protocols, ...



In formal verification:

- e.g. encoding of complex data structures
  - doubly linked lists, ...





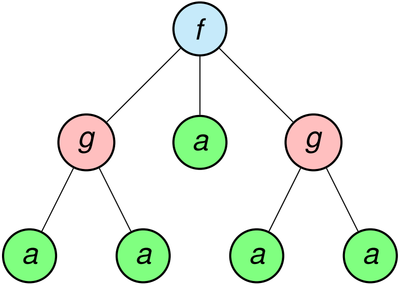
Tree Automata

Finite Tree Automaton (TA): $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- extension of finite automaton to trees:
  - Q … finite set of states,
  - Σ … finite alphabet of symbols with arity,
  - Δ … set of transitions in the form of $p \xrightarrow{b} (q_1, \ldots, q_n)$,
  - F … set of root states.

Example:

$$\Delta = \{ \quad \underline{s} \xrightarrow{f} (r, q, r), \quad r \xrightarrow{g} (q, q), \quad q \xrightarrow{a} \quad \}$$



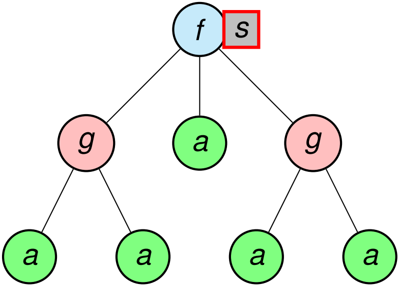
Tree Automata

Finite Tree Automaton (TA): $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- extension of finite automaton to trees:
  - Q . . . finite set of states,
  - Σ . . . finite alphabet of symbols with arity,
  - Δ . . . set of transitions in the form of $p \xrightarrow{b} (q_1, \ldots, q_n)$,
  - F . . . set of root states.

Example:

$$\Delta = \{ \quad \underline{s} \xrightarrow{f} (r, q, r), \quad r \xrightarrow{g} (q, q), \quad q \xrightarrow{a} \quad \}$$



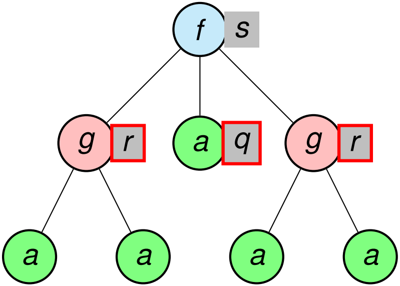
Tree Automata

Finite Tree Automaton (TA): $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- extension of finite automaton to trees:
  - Q ... finite set of states,
  - Σ ... finite alphabet of symbols with arity,
  - Δ ... set of transitions in the form of $p \xrightarrow{b} (q_1, \ldots, q_n)$,
  - F ... set of root states.

Example:

$$\Delta = \{ \underline{s} \xrightarrow{f} (r, q, r), \; r \xrightarrow{g} (q, q), \; q \xrightarrow{a} \}$$



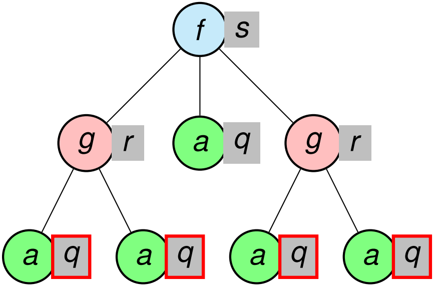
Tree Automata

Finite Tree Automaton (TA): $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- extension of finite automaton to trees:
  - Q . . . finite set of states,
  - Σ . . . finite alphabet of symbols with arity,
  - Δ . . . set of transitions in the form of $p \xrightarrow{b} (q_1, \ldots, q_n)$,
  - F . . . set of root states.

Example:

$$\Delta = \{ \quad \underline{s} \xrightarrow{f} (r, q, r), \quad r \xrightarrow{g} (q, q), \quad q \xrightarrow{a} \quad \}$$



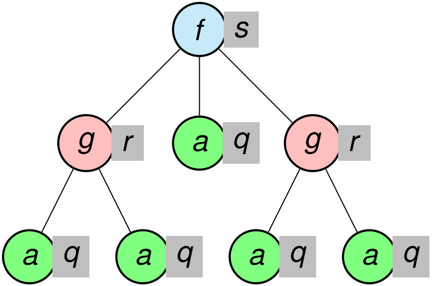
Tree Automata

Finite Tree Automaton (TA): $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- extension of finite automaton to trees:
  - Q . . . finite set of states,
  - Σ . . . finite alphabet of symbols with arity,
  - Δ . . . set of transitions in the form of $p \xrightarrow{b} (q_1, \ldots, q_n)$,
  - F . . . set of root states.

Example:

$$\Delta = \{ \quad \underline{s} \xrightarrow{f} (r, q, r), \quad r \xrightarrow{g} (q, q), \quad q \xrightarrow{a} \quad \}$$



Tree Automata

Tree Automata

- can represent (infinite) sets of trees with regular structure,
- used in XML DBs, language processing, . . . ,
- . . . formal verification, decision procedures of logics (WSkS), . . .

Tree automata in FV:

- often large due to determinisation
  - often advantageous to use non-deterministic tree automata,
  - manipulate them without determinisation,
  - even for operations such as language inclusion (ARTMC, . . .).

Checking Universality and Language Inclusion of TA

Universality of Tree Automata: $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} T_\Sigma$.

Language inclusion of TA: $\mathcal{L}(\mathcal{A}) \stackrel{?}{\subseteq} \mathcal{L}(\mathcal{B})$.

- **EXPTIME-complete**,
- Textbook approach:
  - universality: check $\overline{\mathcal{L}(\mathcal{A}^D)} \stackrel{?}{=} \emptyset$.
  - language inclusion: check $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{B}^D)} \stackrel{?}{=} \emptyset$
- More efficient approaches:
  - upward (bottom-up determinisation),
    - **On-the-fly**,
    - **Antichains** [Bouajjani, Habermehl, Holík, Touili, Vojnar. CIAA'08.],
    - **Antichains+Simulation** [Abdulla, Chen, Holík, Mayr, Vojnar. TACAS'10.].
  - downward.

TA Downward Universality Checking

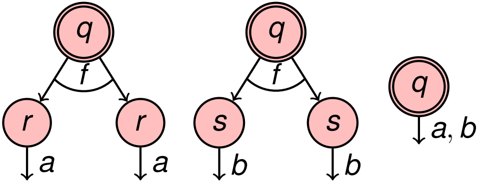
- TA Downward Universality Checking: [Holík, *et al*. ATVA'11]
- inspired by XML Schema containment checking:
  - [Hosoya, Vouillon, Pierce. ACM Trans. Program. Lang. Sys., 2005],
- does not follow the classic schema of universality algorithms:
  - can't determinise: top-down DTA are strictly less powerful than TA.
  - however, there exists a complementation procedure.

TA Downward Universality Checking

$\mathcal{A}$

$$\underline{q} \xrightarrow{f} (r, r) \qquad r \xrightarrow{a}$$
$$\underline{q} \xrightarrow{f} (s, s) \qquad s \xrightarrow{b}$$
$$\underline{q} \xrightarrow{a}$$
$$\underline{q} \xrightarrow{b}$$

$$\Sigma = \{f_2, a_0, b_0\}$$



$$\mathcal{L}(q) = T_\Sigma \text{ if and only if}$$

$$(\mathcal{L}(r) \times \mathcal{L}(r)) \cup (\mathcal{L}(s) \times \mathcal{L}(s)) = T_\Sigma \times T_\Sigma$$

(universality of tuples!)

TA Downward Universality Checking

Note that in general

$$(\mathcal{L}(v_1) \times \mathcal{L}(v_2)) \cup (\mathcal{L}(w_1) \times \mathcal{L}(w_2)) \neq (\mathcal{L}(v_1) \cup \mathcal{L}(w_1)) \times (\mathcal{L}(v_2) \cup \mathcal{L}(w_2))$$

TA Downward Universality Checking

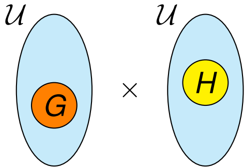
Note that in general

$$(\mathcal{L}(v_1) \times \mathcal{L}(v_2)) \cup (\mathcal{L}(w_1) \times \mathcal{L}(w_2)) \neq (\mathcal{L}(v_1) \cup \mathcal{L}(w_1)) \times (\mathcal{L}(v_2) \cup \mathcal{L}(w_2))$$

However, for universe $\mathcal{U}$ and $G, H \subseteq \mathcal{U}$:

$$G \times H = (G \times \mathcal{U}) \cap (\mathcal{U} \times H)$$

(let $\mathcal{U} = T_\Sigma$ ... all trees over Σ)



TA Downward Universality Checking

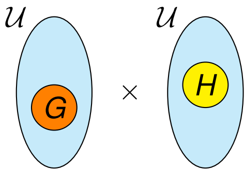
Note that in general

$$(\mathcal{L}(v_1)\times\mathcal{L}(v_2)) \cup (\mathcal{L}(w_1)\times\mathcal{L}(w_2)) \neq (\mathcal{L}(v_1) \cup \mathcal{L}(w_1))\times(\mathcal{L}(v_2) \cup \mathcal{L}(w_2))$$

However, for universe $\mathcal{U}$ and $G, H \subseteq \mathcal{U}$:

$$G \times H = (G \times \mathcal{U}) \cap (\mathcal{U} \times H)$$

(let $\mathcal{U} = T_\Sigma$ ... all trees over Σ)



$$(\mathcal{L}(v_1) \times \mathcal{L}(v_2)) \cup (\mathcal{L}(w_1) \times \mathcal{L}(w_2)) =$$

$$((\mathcal{L}(v_1)\times T_\Sigma) \cap (T_\Sigma\times\mathcal{L}(v_2))) \cup ((\mathcal{L}(w_1)\times T_\Sigma) \cap (T_\Sigma\times\mathcal{L}(w_2)))$$

TA Downward Universality Checking

- Using distributive laws and some further adjustments, we get

$$(\mathcal{L}(v_1) \times \mathcal{L}(v_2)) \cup (\mathcal{L}(w_1) \times \mathcal{L}(w_2)) = T_\Sigma \times T_\Sigma \iff$$

$$\begin{array}{lcll}
(\mathcal{L}(\{v_1, w_1\}) = T_\Sigma) & & & \wedge \\
((\mathcal{L}(\{v_1\}) = T_\Sigma) & \vee & (\mathcal{L}(\{w_2\}) = T_\Sigma)) & \wedge \\
((\mathcal{L}(\{w_1\}) = T_\Sigma) & \vee & (\mathcal{L}(\{v_2\}) = T_\Sigma)) & \wedge \\
 & & (\mathcal{L}(\underbrace{\{v_2, w_2\}}_{\text{macrostate}}) = T_\Sigma) &
\end{array}$$

TA Downward Universality Checking

- Using distributive laws and some further adjustments, we get

$$(\mathcal{L}(v_1)\times\mathcal{L}(v_2)) \cup (\mathcal{L}(w_1)\times\mathcal{L}(w_2)) = T_\Sigma \times T_\Sigma \iff$$

$$\begin{array}{lcll}
(\mathcal{L}(\{v_1, w_1\}) = T_\Sigma) & & & \wedge \\
((\mathcal{L}(\{v_1\}) = T_\Sigma) & \vee & (\mathcal{L}(\{w_2\}) = T_\Sigma)) & \wedge \\
((\mathcal{L}(\{w_1\}) = T_\Sigma) & \vee & (\mathcal{L}(\{v_2\}) = T_\Sigma)) & \wedge \\
 & & (\mathcal{L}(\underbrace{\{v_2, w_2\}}_{\text{macrostate}}) = T_\Sigma) &
\end{array}$$

- Can be generalised to arbitrary arity
  - using the notion of choice functions.

Basic Downward Universality Algorithm

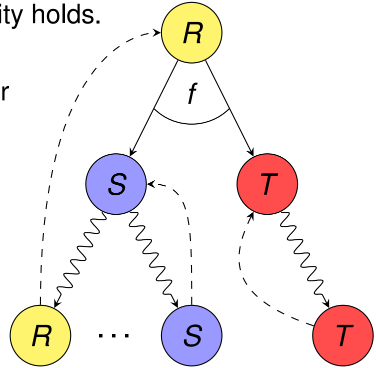
The **On-the-fly** algorithm:

- DFS, maintain *workset* of macrostates.
- Start the algorithm from macrostate F,
- Alternating structure:
  - for all clauses . . .
  - exists a position such that universality holds.

Basic Downward Universality Algorithm

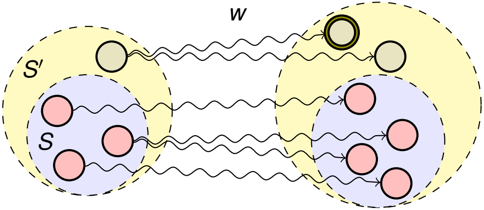
The **On-the-fly** algorithm:

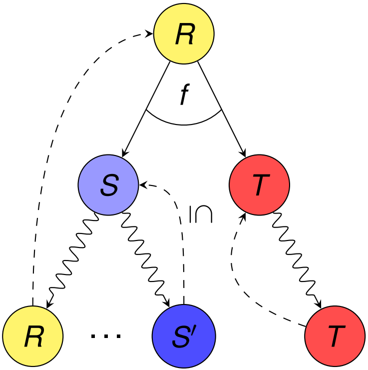
- DFS, maintain *workset* of macrostates.
- Start the algorithm from macrostate F,
- Alternating structure:
  - for all clauses . . .
  - exists a position such that universality holds.
- Cut the DFS when
  - there is no transition for a symbol, or
  - macrostate is already in *workset*.



Optimisations: **Antichains**

1. Cut the DFS on macrostate S' when
   - a smaller macrostate S, $S \subseteq S'$, is already in *workset*,
     - if S is universal, S' will also be universal.

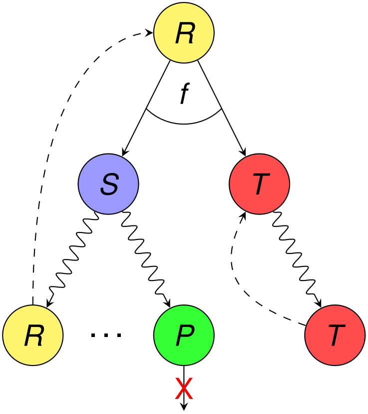




Optimisations of Downward TA Universality Algorithm

Optimisations: **Antichains**

2. If a macrostate P is found to be non-universal, cache it;
   - do not expand any new macrostate $P' \subseteq P$,
     - surely $\mathcal{L}(P') \neq T_\Sigma$.

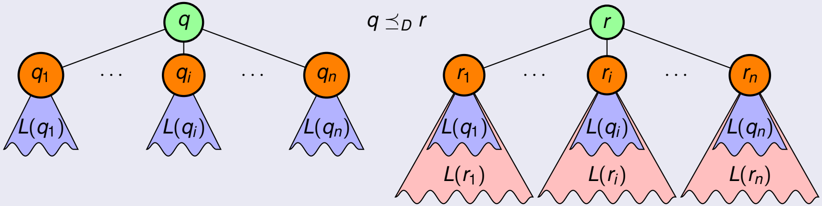


Optimisations of Downward TA Universality Algorithm

Optimisations: **Antichains + Simulation**

- Downward simulation
  - implies inclusion of (downward) tree languages of states,
  - usually quite rich.

Downward simulation $\preceq_D$



- In **Antichains**, instead of $\subseteq$ use $\preceq_D^{\forall\exists}$.
- further, minimise macrostates w.r.t. $\preceq_D$: $\{q, r, x\} \Rightarrow \{\quad r, x\}$

Experiments

- Comparison of different inclusion checking algorithms
  - `down` — downward, `up` — upward,
  - `+s` — using upward/downward simulation.
  - implemented in the VATA library.

	`down`	`down+s`	`up`	`up+s`
Winner	68.55 %	7.30 %	24.14 %	0.00 %
Timeouts	32.51 %	18.27 %	0.00 %	0.00 %

Conclusion

- A new class of efficient algorithms for downward checking of universality and language inclusion of tree automata.
- Process automata downwards, making it possible to exploit downward simulation.

Future work

- Further develop TA universality & inclusion checking algorithms
  - e.g. by the up-to congruence technique [Bonchi, Pous. POPL'13.].

- Develop algorithms for computations of simulations for both
  - explicitly, and
  - semi-symbolically represented TA.

Thank you for your attention.

Questions?