Efficient Techniques for Manipulation of Non-deterministic 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

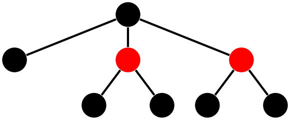
May 22, 2012

Outline

Trees

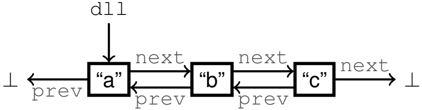
Very popular in computer science:

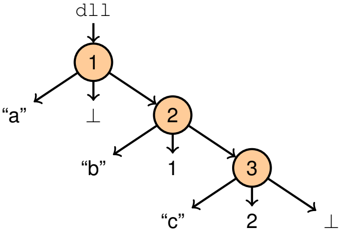
- data structures,
- computer network topologies,
- distributed protocols, . . .



In formal verification:

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





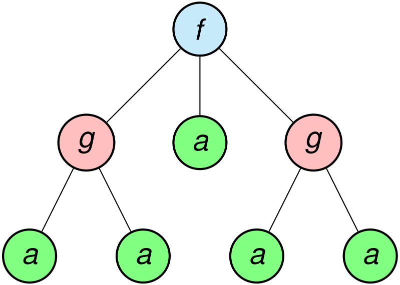
Tree Automata

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 $p \xrightarrow{a} (q_1, \ldots, q_n)$,
  - F ... set of initial/final (root) states.

Example:

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



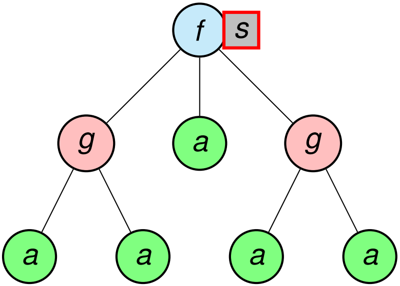
Tree Automata

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 $p \xrightarrow{a} (q_1, \ldots, q_n)$,
  - F ... set of initial/final (root) states.

Example:

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



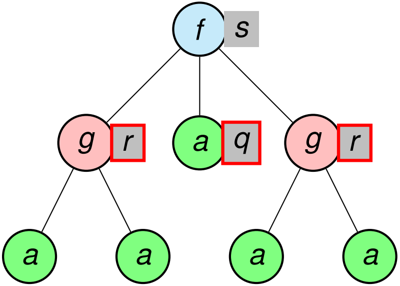
Tree Automata

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 $p \xrightarrow{a} (q_1, \ldots, q_n)$,
  - F ... set of initial/final (root) states.

Example:

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



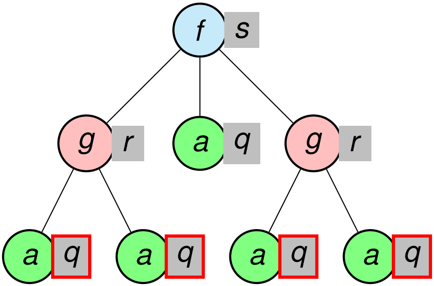
Tree Automata

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 $p \xrightarrow{a} (q_1, \ldots, q_n)$,
  - F ... set of initial/final (root) states.

Example:

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



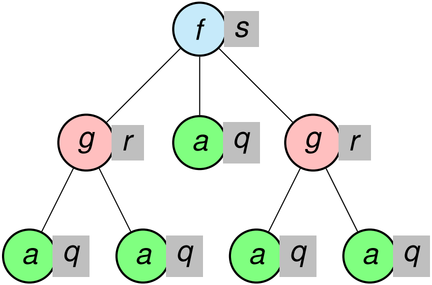
Tree Automata

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 $p \xrightarrow{a} (q_1, \ldots, q_n)$,
  - F ... set of initial/final (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 some logics, . . .

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, . . .),
- handling large alphabets (MSO, WSkS).

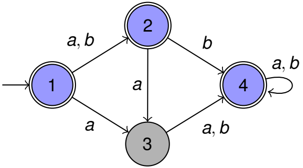
- We focus on the problem of checking language inclusion.

- For simplicity, we demonstrate the ideas on finite automata,
- their extension to tree automata is quite straight.

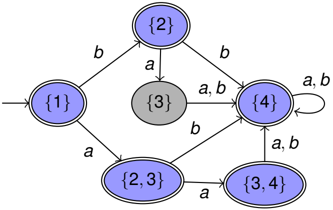
Finite Automata Universality Checking

- **PSPACE-complete**
- The Textbook algorithm for checking

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$

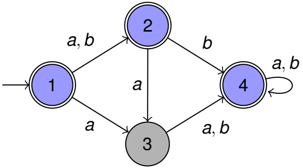


1. Determinise $\mathcal{A} \to \mathcal{A}^D$.
2. Complement $\mathcal{A}^D \to \overline{\mathcal{A}^D}$
   - by complementing the set of final states.
3. Check $\mathcal{L}(\overline{\mathcal{A}^D}) \stackrel{?}{=} \emptyset$,
   - search for a reachable final state.



Finite Automata Universality Checking

- **PSPACE-complete**

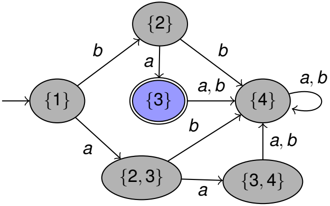


- The Textbook algorithm for checking

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$

1. Determinise $\mathcal{A} \to \mathcal{A}^D$.

2. Complement $\mathcal{A}^D \to \overline{\mathcal{A}^D}$
   - by complementing the set of final states.

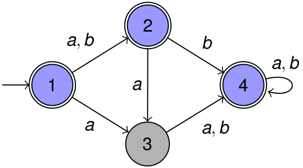


3. Check $\mathcal{L}(\overline{\mathcal{A}^D}) \stackrel{?}{=} \emptyset$,
   - search for a reachable final state.

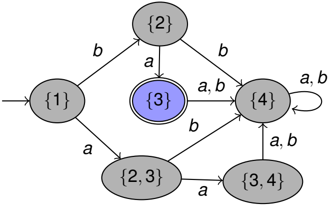
Finite Automata Universality Checking

- **PSPACE-complete**
- The Textbook algorithm for checking

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$



1. Determinise $\mathcal{A} \to \mathcal{A}^D$.
   - **exponential explosion!**
2. Complement $\mathcal{A}^D \to \overline{\mathcal{A}^D}$
   - by complementing the set of final states.
3. Check $\mathcal{L}(\overline{\mathcal{A}^D}) \stackrel{?}{=} \emptyset$,
   - search for a reachable final state.



Finite Automata Universality Checking

- **PSPACE-complete**
- The Textbook algorithm for checking

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$

1. Determinise $\mathcal{A} \rightarrow \mathcal{A}^D$.
   - **exponential explosion!**
2. Complement $\mathcal{A}^D \rightarrow \overline{\mathcal{A}^D}$
   - by complementing the set of final states.
3. Check $\mathcal{L}(\overline{\mathcal{A}^D}) \stackrel{?}{=} \emptyset$,
   - search for a reachable final state.

Inclusion checking

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{\supseteq} \mathcal{L}(\mathcal{B})$$

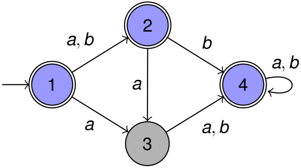
Inclusion checking

$$\mathcal{L}(\overline{\mathcal{A}^D}) \cap \mathcal{L}(\mathcal{B}) \stackrel{?}{=} \emptyset$$

Finite Automata Universality Checking

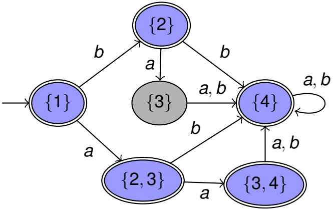
The On-the-fly algorithm for checking universality

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$



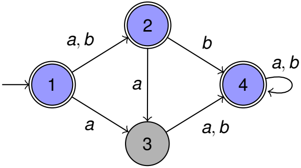
1. Traverse $\mathcal{A}$ from the initial states.
2. Perform on-the-fly determinisation, keep a *workset* of macrostates.
3. If encountered a macrostate P, such that $P \cap F = \emptyset$,
   - return **false**.
4. Otherwise, return **true**.

workset $= \{ \quad \}$



Finite Automata Universality Checking

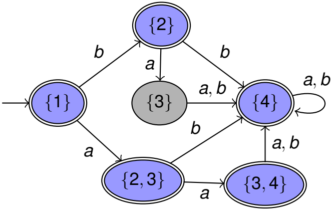
The On-the-fly algorithm for checking universality



$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$

1. Traverse $\mathcal{A}$ from the initial states.
2. Perform on-the-fly determinisation, keep a *workset* of macrostates.
3. If encountered a macrostate P, such that $P \cap F = \emptyset$,
   - return **`false`**.
4. Otherwise, return **`true`**.

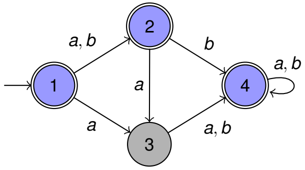
$$workset = \{ \overbrace{\{\underline{1}\}}^{Init} \qquad \}$$



Finite Automata Universality Checking

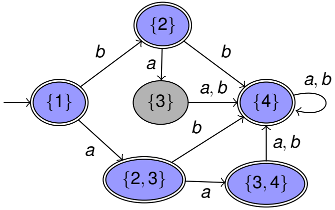
The On-the-fly algorithm for checking universality

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$



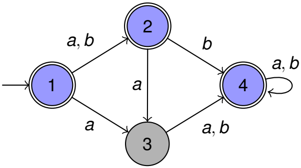
1. Traverse $\mathcal{A}$ from the initial states.
2. Perform on-the-fly determinisation, keep a *workset* of macrostates.
3. If encountered a macrostate P, such that $P \cap F = \emptyset$,
   - return **false**.
4. Otherwise, return **true**.

$$\mathit{workset} = \{ \overbrace{\{\underline{1}\}}^{\mathit{Init}}, \underbrace{\{\underline{2}, 3\}}_{\{1\} \xrightarrow{a}}, \overbrace{\{\underline{2}\}}^{\{1\} \xrightarrow{b}} \}$$



Finite Automata Universality Checking

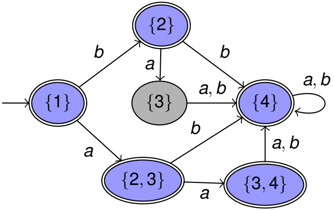
The On-the-fly algorithm for checking universality



$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$

1. Traverse $\mathcal{A}$ from the initial states.
2. Perform on-the-fly determinisation, keep a *workset* of macrostates.
3. If encountered a macrostate P, such that $P \cap F = \emptyset$,
   - return **`false`**.
4. Otherwise, return **`true`**.

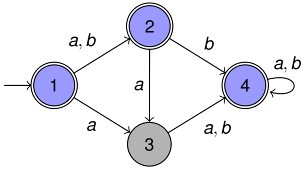
$$workset = \{ \overbrace{\{\underline{1}\}}^{Init}, \underbrace{\{\underline{2},3\}}_{\{1\}\xrightarrow{a}}, \overbrace{\{\underline{2}\}}^{\{1\}\xrightarrow{b}}, \underbrace{\{3,\underline{4}\}}_{\{2,3\}\xrightarrow{a}}, \overbrace{\{\underline{4}\}}^{\{2,3\}\xrightarrow{b}} \quad \}$$



Finite Automata Universality Checking

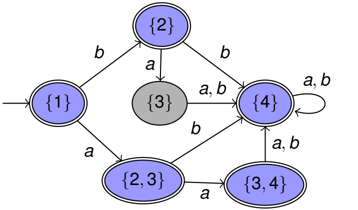
The On-the-fly algorithm for checking universality

$$\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \Sigma^*$$



1. Traverse $\mathcal{A}$ from the initial states.
2. Perform on-the-fly determinisation, keep a *workset* of macrostates.
3. If encountered a macrostate P, such that $P \cap F = \emptyset$,
   - return **false**.
4. Otherwise, return **true**.

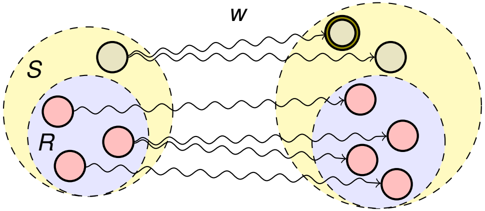
$$\textit{workset} = \{ \overbrace{\{\underline{1}\}}^{\textit{Init}}, \underbrace{\{\underline{2},3\}}_{\{1\}\xrightarrow{a}}, \overbrace{\{\underline{2}\}}^{\{1\}\xrightarrow{b}}, \underbrace{\{3,\underline{4}\}}_{\{2,3\}\xrightarrow{a}}, \overbrace{\{\underline{4}\}}^{\{2,3\}\xrightarrow{b}}, \underbrace{\{3\}}_{\{2\}\xrightarrow{a}} \}$$



Finite Automata Universality Checking

Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

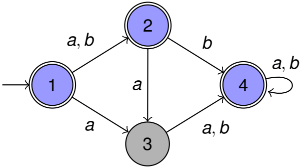


R has a bigger chance to encounter a non-final macrostate

Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

$$\textit{workset} = \{ \qquad\qquad\qquad\qquad \}$$

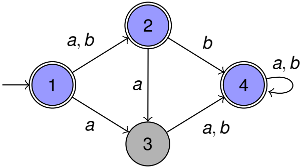


Finite Automata Universality Checking

Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

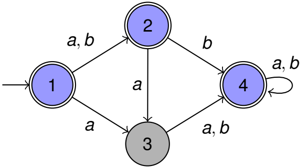
$$\textit{workset} = \{ \overbrace{\{\underline{1}\}}^{\textit{Init}} \qquad\qquad\qquad\qquad \}$$



Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

$$\textit{workset} = \{ \overbrace{\{\underline{1}\}}^{\textit{Init}}, \underbrace{\{\underline{2},3\}}_{\{1\}\xrightarrow{a}}, \overbrace{\{\underline{2}\}}^{\{1\}\xrightarrow{b}} \qquad \}$$

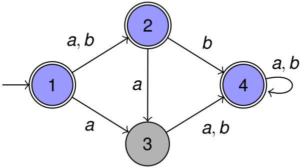


Finite Automata Universality Checking

Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

$$workset = \{ \overbrace{\{\underline{1}\}}^{Init} , \overbrace{\{\underline{2}\}}^{\{1\}\xrightarrow{b}} \}$$

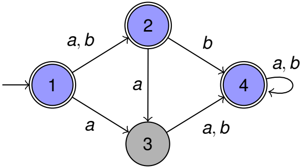


Finite Automata Universality Checking

Optimisations:

- The Antichains algorithm [De Wulf, Doyen, Henzinger, Raskin. CAV'06],
- keep only macrostates sufficient to encounter a non-final set:
  - if macrostates R and S, $R \subseteq S$, are both in *workset*,
    - remove S from *workset*.

$$\mathit{workset} = \{ \overbrace{\{\underline{1}\}}^{\mathit{Init}}, \overbrace{\{\underline{2}\}}^{\{1\}\xrightarrow{b}}, \underbrace{\{3\}}_{\{2\}\xrightarrow{a}}, \overbrace{\{\underline{4}\}}^{\{2\}\xrightarrow{b}} \}$$



Finite Automata Universality Checking

Optimisations:

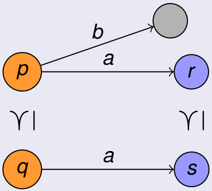
- The Antichains + Simulation algorithm [Abdulla, *et al.* TACAS'10],

Simulation

A preorder $\preceq$ such that

$$q \preceq p \implies$$

$$\left(\forall a \in \Sigma \,.\, q \xrightarrow{a} s \implies p \xrightarrow{a} r \land s \preceq r\right)$$



Note that $q \preceq p \implies \mathcal{L}(q) \subseteq \mathcal{L}(p)$!

- refine *workset* using simulation
  - if macrostates R and S, $R \preceq^{\forall\exists} S$, are both in *workset*
    - remove S from *workset*,
  - further, minimise macrostates w.r.t. $\preceq$: $\{p, q, x\} \Rightarrow \{p, \quad x\}$

Tree Automata Universality Checking

- **EXPTIME-complete**
- Checking whether $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} T_{\Sigma}$.
- The (upward) Textbook, On-the-fly, and Antichains algorithms:
  - straightforward extension of the algorithms for FA,
  - perform upward (i.e. bottom-up) determinisation of the TA,
  - need to find tuples of macrostates to perform an upward transition.
- The (upward) Antichains + Simulation algorithm:
  - needs to use upward simulation (implies inclusion of "*open trees*")
    - usually not very rich.

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.

TA Downward Universality Checking

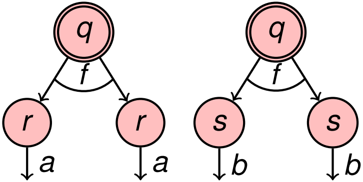
$\mathcal{A}$

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

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

$\underline{q} \xrightarrow{a}$

$\underline{q} \xrightarrow{b}$



$\mathcal{L}(q) = T_\Sigma$ 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

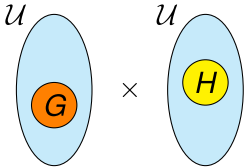
Note that in general

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

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

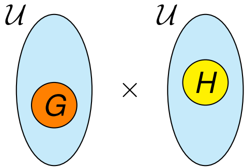
Note that in general

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

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)



$$\begin{array}{ccccccc} (\mathcal{L}(v_1) & \times & \mathcal{L}(v_2)) & \cup & (\mathcal{L}(w_1) & \times & \mathcal{L}(w_2)) = \\ ((\mathcal{L}(v_1)\times T_\Sigma) & \cap & (T_\Sigma\times\mathcal{L}(v_2))) & \cup & ((\mathcal{L}(w_1)\times T_\Sigma) & \cap & (T_\Sigma\times\mathcal{L}(w_2))) \end{array}$$

TA Downward Universality Checking

- Using distributive laws and some further adjustments, we get

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

$$\begin{array}{llll}
(\mathcal{L}(\{v_1, w_1\}) = T_\Sigma) & & & \wedge \\
((\mathcal{L}(v_1) = T_\Sigma) & \vee & (\mathcal{L}(w_2) = T_\Sigma)) & \wedge \\
((\mathcal{L}(w_1) = T_\Sigma) & \vee & (\mathcal{L}(v_2) = T_\Sigma)) & \wedge \\
(\mathcal{L}(\{v_2, w_2\}) = T_\Sigma) & & &
\end{array}$$

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

Basic Downward Universality Algorithm

- DFS, maintain *workset* of macrostates.
- Start the algorithm from macrostate F (final states).
- Alternating structure:
  - for all clauses ...
  - exists a position such that universality holds.
- Sooner or later, the DFS either
  - reaches a leaf, or
  - reaches a macrostate which is already in *workset*.

Optimisations of Downward TA Universality Algorithm

Optimisations: Antichains

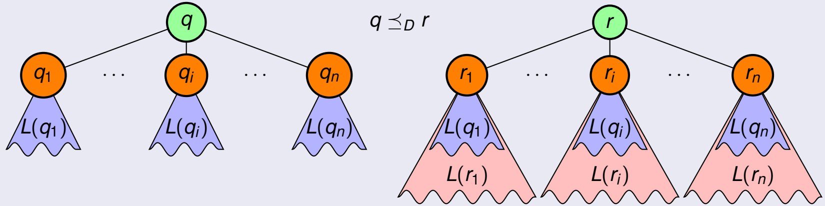
1. If a macrostate P is found to be non-universal, cache it;
   - do not expand any new macrostate $S \subseteq P$ (surely $\mathcal{L}(S) \neq T_\Sigma$).
2. For a macrostate R, check whether there is $S \subseteq R$ in ***workset***
   - in case it is, return (if S is universal, R will also be universal).
3. Some more optimisations (if interested, see our paper!)

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 $\preceq_D^{\exists\forall}$.
- further, minimise macrostates w.r.t. $\preceq_D$: $\{p, q, x\} \Rightarrow \{p, \quad x\}$

Experiments

Size	50–250	400–600
Pairs	323	64
Timeout	20 s	60 s
`Up`	31.21 %	9.38 %
`Up+s`	0.00 %	0.00 %
`Down`	53.50 %	39.06 %
`Down+s`	15.29 %	51.56 %
Avg up	1.71	0.34
Avg down	3.55	46.56

including simulation
computation time
$(T_{sim} + T_{incl})$

Size	50–250	400–600
Pairs	323	64
Timeout	20 s	60 s
`Up+s`	81.82 %	20.31 %
`Down+s`	18.18 %	79.69 %
Avg up	1.33	9.92
Avg down	3.60	2116.29

without simulation
computation time
(T_{incl})

VATA: A Tree Automata Library

VATA is a new tree automata library that

- supports non-deterministic tree automata,
- provides encodings suitable for different contexts:
  - explicit, and
  - semi-symbolic,
- is written in C++,
- is open source and free under GNU GPLv3,
  - `http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/`
  - or (shorter), `http://goo.gl/KNpMH`

Supported Operations

Supported operations:

- union,
- intersection,
- removing unreachable or useless states and transitions,
- testing language emptiness,

- computing downward and upward simulation,
- simulation-based reduction,
- testing language inclusion,

- import from file/export to file.

Simulations

Explicit:

- downward simulation $\preceq_D$,
- upward simulation $\preceq_U$.

Work by transforming automaton to labelled transition systems,

- computing simulation on the LTS, [Holík, Šimáček. MEMICS'09],
- which is an improvement of [Ranzato, Tapparo. LICS'07].

Semi-symbolic:

- downward simulation computation based on [Henzinger, Henzinger, Kopke. FOCS'95].

Reduction according to downward simulation.

Conclusion

- A new tree automata library available
  - containing various optimisations of the used algorithms,
  - particularly AFAWK state-of-the-art inclusion checking algorithms.
- Support for working with non-deterministic automata.
- Easy to extend with own encoding/operations.
- The library is open source and free under GNU GPLv3.
- Available at

  `http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/`

Future work

- Add new representations of finite word/tree automata,
  - that address particular issues, such as
    - large number of states, or
    - fast checking of language inclusion.
- Add missing operations,
  - development is demand-driven,
  - if you miss something, write to us, the feature may appear soon.

Thank you for your attention.

Questions?