VATA: A Library for Efficient Manipulation of Non-deterministic Tree Automata

Ondřej Lengál[1] Jiří Šimáček[1,2] Tomáš Vojnar[1]

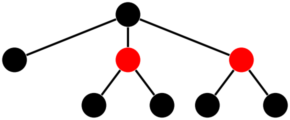
[1]Brno University of Technology, Czech Republic
[2]VERIMAG, UJF/CNRS/INPG, Gières, France

March 27, 2012

Trees

Very popular in computer science:

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

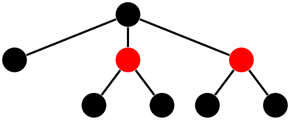


http://goo.gl/KNpMH

Trees

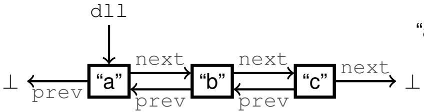
Very popular in computer science:

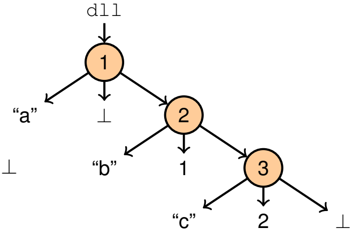
- data structures,
- computer network topologies,
- distributed protocols, . . .



In formal verification:

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





  - . . .

http://goo.gl/KNpMH

Tree Automata

Finite Non-deterministic 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, $\#a$,
  - Δ ... set of transitions in the form of $p \xrightarrow{a} (q_1, \ldots, q_n), \#a = n$,
  - F ... set of final states.

http://goo.gl/KNpMH

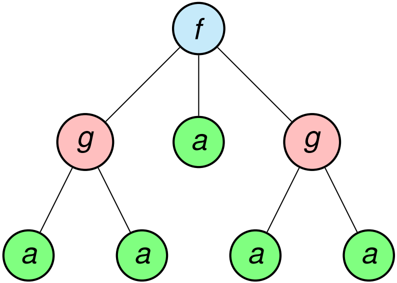
Tree Automata

Finite Non-deterministic 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, $\#a$,
  - Δ … set of transitions in the form of $p \xrightarrow{a} (q_1, \ldots, q_n), \#a = n$,
  - F … set of final states.

Example:

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



http://goo.gl/KNpMH

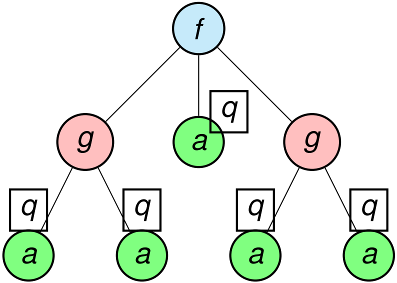
Tree Automata

Finite Non-deterministic 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, $\#a$,
  - Δ ... set of transitions in the form of $p \xrightarrow{a} (q_1, \ldots, q_n), \#a = n$,
  - F ... set of final states.

Example:

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



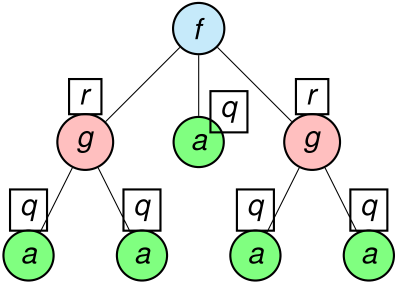
http://goo.gl/KNpMH

Finite Non-deterministic 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, $\#a$,
  - Δ ... set of transitions in the form of $p \xrightarrow{a} (q_1, \ldots, q_n), \#a = n$,
  - F ... set of final states.

Example:

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



http://goo.gl/KNpMH

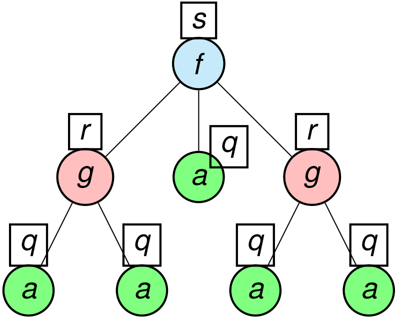
Tree Automata

Finite Non-deterministic 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, $\#a$,
  - Δ ... set of transitions in the form of $p \xrightarrow{a} (q_1, \ldots, q_n), \#a = n$,
  - F ... set of final states.

Example:

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



http://goo.gl/KNpMH

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, . . .

http://goo.gl/KNpMH

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 formal verification:

- often large due to determinisation
  - often advantageous to use non-deterministic tree automata,
  - manipulate them without determinisation,
  - even for operations such as language inclusion or size reduction,
- handling large alphabets (MSO, WSkS).

http://goo.gl/KNpMH

Available Tree Automata Libraries

- Timbuk/Taml:
  - written in OCaml,
  - explicit encoding,
  - basic support for operations on non-deterministic automata.

- MONA TA library:
  - written in C,
  - semi-symbolic encoding using MTBDDs,
    - *multi-terminal binary decision diagrams*,
  - supports deterministic binary automata only.

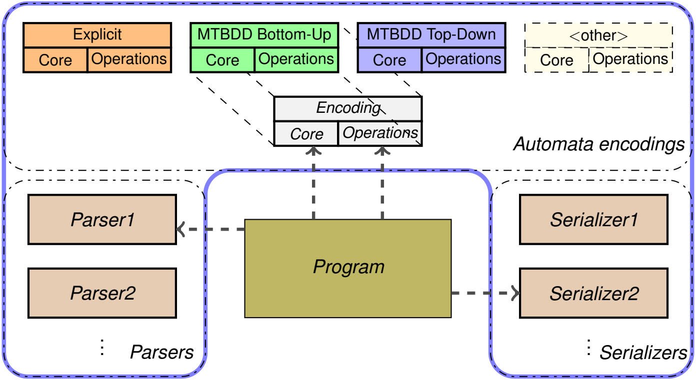
http://goo.gl/KNpMH

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/

http://goo.gl/KNpMH

Architecture of VATA



VATA is a framework that can be easily extended:

- the whole infrastructure can be used even for an own TA encoding,
- easy to be extended with word automata, ω-automata,
- word automata currently supported as unary TA. `http://goo.gl/KNpMH`

VATA: Explicit Encoding

- Transitions stored in the top-down manner,
  - advantageous in some cases.
- Transitions maintained in shared structures,
  - modifications using copy-on-write.

A

$q_1 \xrightarrow{c} (q_1, q_2),$

$q_1 \xrightarrow{e} \; ,$

$q_2 \xrightarrow{a} (q_1, q_1),$

$q_2 \xrightarrow{b} (q_1, q_1),$

$q_2 \xrightarrow{b} (q_1, q_2)$

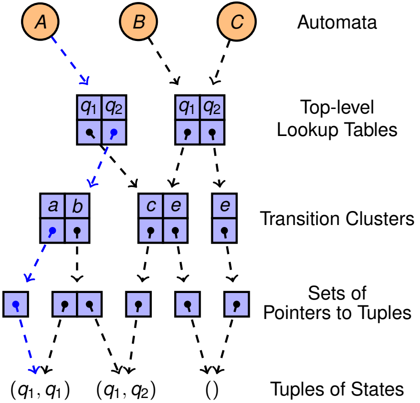
B

$q_1 \xrightarrow{c} (q_1, q_2),$

$q_1 \xrightarrow{e} \; ,$

$q_2 \xrightarrow{e}$

$C = B$

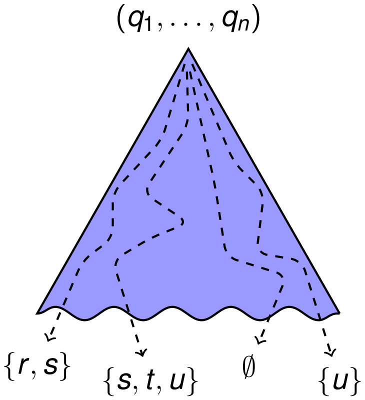


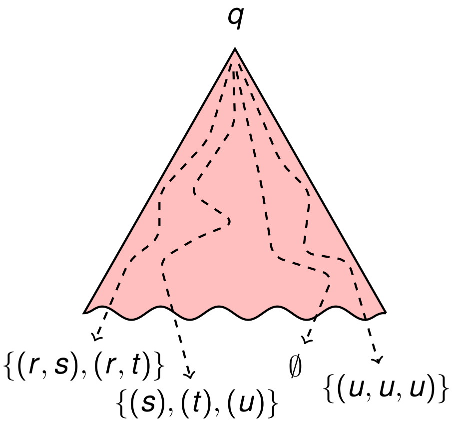
http://goo.gl/KNpMH

VATA: Semi-symbolic Encoding

Dual representation using our own MTBDD library:

- Bottom-up:
- Top-down:





Bottom-up : inspired by MONA, but has sets of states in leaves.
Top-down : sets of state tuples in leaves.

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.

http://goo.gl/KNpMH

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].

http://goo.gl/KNpMH

Tree Automata Reduction

Simulation-based reduction of TA:

1. Compute the downward simulation relation $\preceq_D$ on states of TA.
2. Take the symmetric fragment $\sim_D$ of $\preceq_D$, $\quad \sim_D = \preceq_D \cap \preceq_D^{-1}$
   - $\sim_D$ is a language compatible equivalence relation.
3. Merge states in all equivalence classes of $\sim_D$.

http://goo.gl/KNpMH

Language Inclusion Checking

Textbook approach for checking $\mathcal{L}(\mathcal{A}_S) \subseteq \mathcal{L}(\mathcal{A}_B)$ on TA:

- Check $\mathcal{A}_S \cap \overline{\mathcal{A}_B} = \emptyset$.

Two methods in VATA:

- upward (optimised version of the textbook approach),
- downward.

http://goo.gl/KNpMH

Upward Language Inclusion Checking

- [Abdulla, Chen, Holík, Mayr, Vojnar. TACAS'10]

The idea will be presented on testing universality of $\mathcal{A} = (Q, \Sigma, \Delta, F)$.

- the extension to checking TA inclusion is straightforward.

On-the-fly approach:

1. Traverse $\mathcal{A}$ bottom-up.
2. Maintain a workset W of sets $P \subseteq Q$.
3. Generate tuples $(P_1, \ldots, P_n)$ where $P_1, \ldots, P_n \in W$.
4. $\forall f \in \Sigma$ generate T s.t. $(P_1, \ldots, P_n) \xrightarrow{f} T$.
5. If you encounter R where $R \cap F = \emptyset$, return false.
6. If no new sets are found, return true.

Optimisations:

- use antichains and upward simulation.

http://goo.gl/KNpMH

Downward Language Inclusion Checking

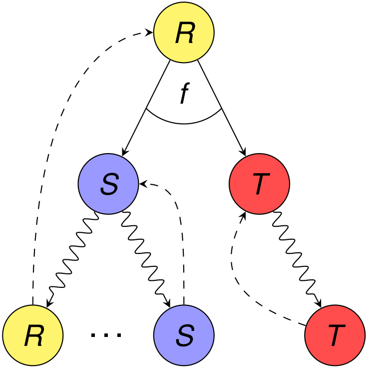
- [Holík, Lengál, Šimáček, Vojnar. ATVA'11]
- The main idea will also be explained on checking TA universality.
- A set of states R is universal, $U(R)$, iff for all symbols $f \in \Sigma$:
  - if $\#f = 0$, then there is a state $q \in R$ s.t. $q \xrightarrow{f}$,
  - if $\#f = n > 0$,
    - given the set U of all tuples accessible from R over f,
    - for all choice functions $c : U \to \{1, \ldots, n\}$,
    - there exists $i \in \{1, \ldots, n\}$ s.t. $U(c^{-1}(i))$ (recursively).

http://goo.gl/KNpMH

Downward Language Inclusion Checking

Idea of the algorithm:

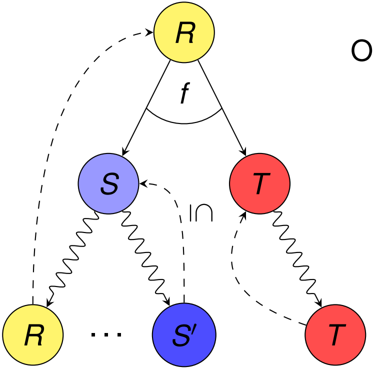
1. Start from the set of accepting states.
2. Perform a DFS while checking the universality condition.
3. Cut the DFS when
   - the condition is falsified, or
   - the DFS finds a set already on the stack.



http://goo.gl/KNpMH

Idea of the algorithm:

1. Start from the set of accepting states.
2. Perform a DFS while checking the universality condition.
3. Cut the DFS when
   - the condition is falsified, or
   - the DFS finds a set already on the stack.



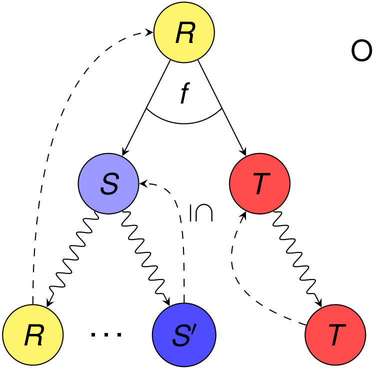
Optimisation 1:

- compare sets of states w.r.t. inclusion rather than equality:
- if S is universal, $U(S)$, and $S' \supseteq S$, then S' will also be universal, $U(S')$,

http://goo.gl/KNpMH

Idea of the algorithm:

1. Start from the set of accepting states.
2. Perform a DFS while checking the universality condition.
3. Cut the DFS when
   - the condition is falsified, or
   - the DFS finds a set already on the stack.



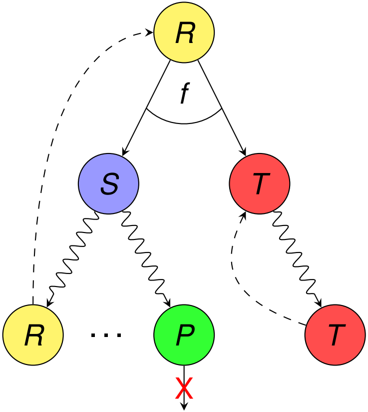
Optimisation 1:

- compare sets of states w.r.t. inclusion rather than equality:
- if S is universal, $U(S)$, and $S' \supseteq S$, then S' will also be universal, $U(S')$,
- instead of inclusion, a weaker language compatible relation, such as downward simulation, can be used.

http://goo.gl/KNpMH

Optimisation 2: (antichains)

- if we find a set P which is not universal, $\neg U(P)$, we cache it and never expand a set P' s.t. $P' \subseteq P$, because $\neg U(P) \implies \neg U(P')$,

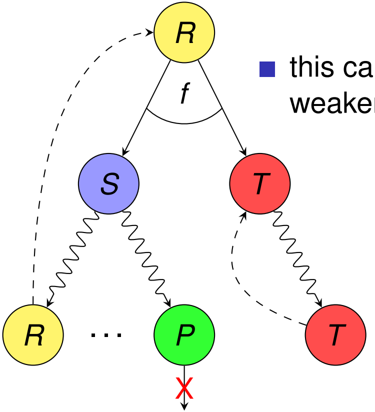


http://goo.gl/KNpMH

Downward Language Inclusion Checking

Optimisation 2: (antichains)

- if we find a set P which is not universal, $\neg U(P)$, we cache it and never expand a set P' s.t. $P' \subseteq P$, because $\neg U(P) \implies \neg U(P')$,



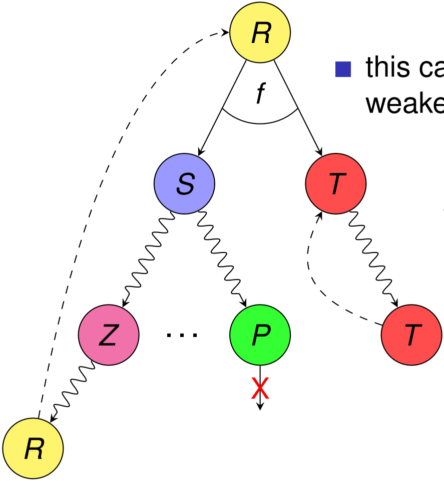
- this can again be generalised to a weaker language compatible relation.

http://goo.gl/KNpMH

Downward Language Inclusion Checking

Optimisation 2: (antichains)

- if we find a set P which is not universal, $\neg U(P)$, we cache it and never expand a set P' s.t. $P' \subseteq P$, because $\neg U(P) \implies \neg U(P')$,



- this can again be generalised to a weaker language compatible relation.

A similar optimisation for the case when for Z it is found out that the universality condition holds cannot be done in the same manner.

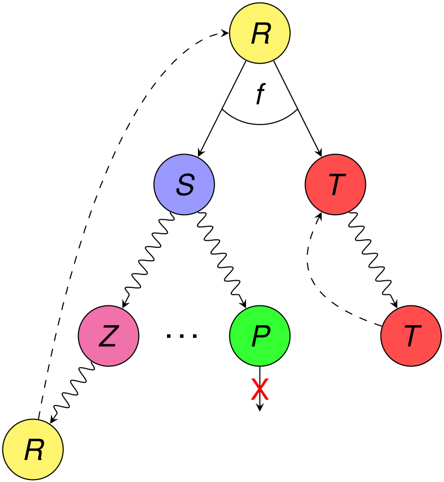
- The reason is that universality of R may be falsified on other branches.

http://goo.gl/KNpMH

Downward Language Inclusion Checking

Optimisation 3: (further improving [ATVA'11])

- cache the set Z for which the universality condition holds, but together with the precondition why it holds:

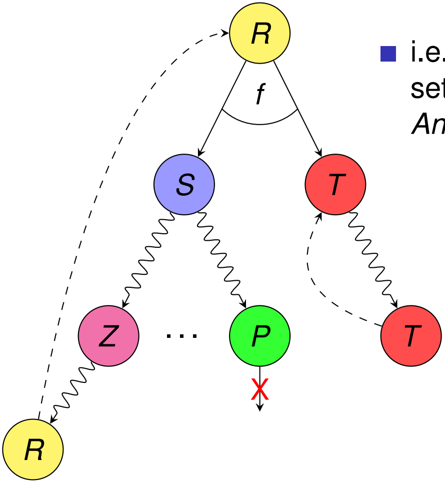


http://goo.gl/KNpMH

Downward Language Inclusion Checking

Optimisation 3: (further improving [ATVA'11])

- cache the set Z for which the universality condition holds, but together with the precondition why it holds:



- i.e. we maintain a pair (Ant, Con) of sets of sets of states meaning that $Ant \implies Con$, i.e.

$$\bigwedge_{A \in Ant} U(A) \implies \bigwedge_{C \in Con} U(C),$$

- when the DFS is returning via G, it removes G from Ant and adds G to Con.
- when Ant becomes empty, all sets from Con are cached.

http://goo.gl/KNpMH

Experiments

Explicit encoding:

- Comparison to Timbuk/Taml (tested on $\sim 3,000$ pairs of TA):
  - $20,000\times$ faster on union,
  - $100,000\times$ faster on intersection.

- Comparison of different inclusion checking algorithms
  - `down` — downward, `up` — upward,
  - `+s` — using upward/downward simulation,
  - `-o` — with optimisation 3 (*Ant*, *Con*).

	down	down+s	down-o	down-o+s	up	up+s
Winner	36.35 %	4.15 %	32.20 %	3.15 %	24.14 %	0.00 %
Timeouts	32.51 %	18.27 %	32.51 %	18.27 %	0.00 %	0.00 %

http://goo.gl/KNpMH

Experiments

Semi-symbolic encoding:

- Comparison to our previous version that used CUDD:
  - being over 300 times faster on inclusion checking on average,
- Comparison of different inclusion checking algorithms
  - `down` — downward, `up` — upward,
  - `+s` — using downward simulation,
  - `-o` — with optimisation 3 (*Ant*, *Con*).

	down	down+s	down-o	down-o+s	up
Winner	44.02 %	0.00 %	31.73 %	0.00 %	24.25 %
Timeouts	5.87 %	77.93 %	5.87 %	78.00 %	22.26 %

http://goo.gl/KNpMH

Conclusion

- We developed a new tree automata library,
  - containing various optimisations of the used 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/

http://goo.gl/KNpMH

Future work

- Improve the semi-symbolic downward simulation algorithm.
- 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.

http://goo.gl/KNpMH

Questions?