Tree automata techniques for the verification of infinite state-systems

Summer School VTSA 2011

Florent Jacquemard

INRIA Saclay & LSV (UMR CNRS/ENS Cachan)

florent.jacquemard@inria.fr

http://www.lsv.ens-cachan.fr/~jacquema

TATA book http://tata.gforge.inria.fr

(chapters 1, 3, 7, 8)

Tree Automata Techniques and Applications

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, Marc Tommasi

Finite tree automata

- tree recognizers
- generalize NFA from words to trees

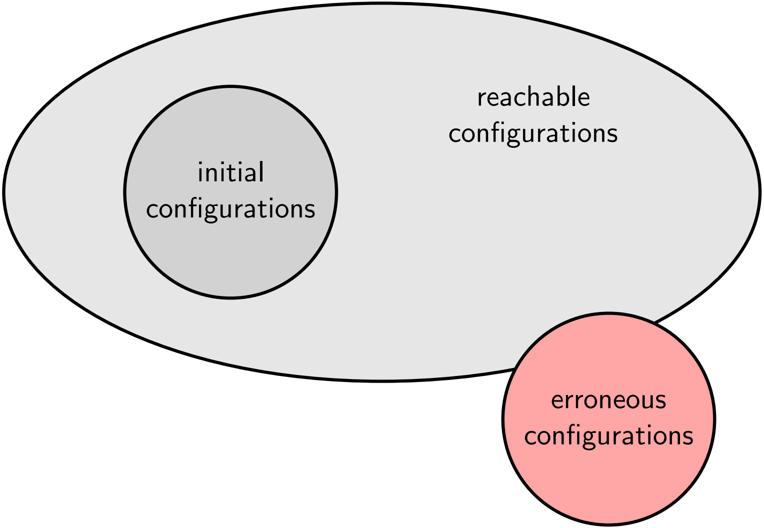
= finite representations of infinite set of labeled trees

are a useful tool for verification procedures

- composition results
  - closure under Boolean operations
  - closure under transformations
- decision results, efficient algorithms
- expressiveness, close relationship with logic

Verification of infinite state systems

regular model checking : static analysis of safety properties for infinite state systems, using symbolic reachability verification techniques.



Concurrent readers/writers

Example from [Clavel et al. LNCS 4350 2007]

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

- writers can access the file if nobody else is accessing it (1)
- readers can access the file if no writer is accessing it (2)
- readers and writers can leave the file at any time (3,4)

Properties expected:

- mutual exclusion between readers and writers
- mutual exclusion between writers

Concurrent readers/writers: reachable configurations

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

Initial configuration: $\mathsf{state}(0,0)$

Concurrent readers/writers: reachable configurations

$$\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}$$

Reachable configurations: $\mathsf{state}(0,0)$

Concurrent readers/writers: reachable configurations

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

Reachable configurations:

$$
\begin{tikzcd}
\mathsf{state}(0,0) \arrow[r, bend left, "1"] & \mathsf{state}\big(0,s(0)\big) \arrow[l, bend left, "3"]
\end{tikzcd}
$$

Concurrent readers/writers: reachable configurations

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

Reachable configurations:

$$
\begin{tikzcd}
\mathsf{state}(0,0) \arrow[r, bend left, "1"] \arrow[d, bend right, "2"'] & \mathsf{state}\big(0,s(0)\big) \arrow[l, bend left, "3"] \\
\mathsf{state}\big(s(0),0\big) \arrow[u, bend right, "4"'] \arrow[d, bend right, "2"'] & \\
\mathsf{state}\big(s(s(0)),0\big) \arrow[u, bend right, "4"'] & \\
\vdots &
\end{tikzcd}
$$

Concurrent readers/writers: finite representation

$$
\begin{tikzcd}
\mathsf{state}(0,0) \arrow[rr, bend left, "1"] \arrow[d, bend right, "2"'] & & \mathsf{state}\big(0,s(0)\big) \arrow[ll, bend left, "3"] \\
\mathsf{state}\big(s(0),0\big) \arrow[u, bend right, "4"'] \arrow[d, bend right, "2"'] & & \\
\mathsf{state}\big(s(s(0)),0\big) \arrow[u, bend right, "4"'] & & \\
\vdots & &
\end{tikzcd}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0, q_0) \mid \mathsf{state}(q_0, q_1) \mid \mathsf{state}(q_1, q_0) \mid \mathsf{state}(q_2, q_0) \\
q_1 & := & s(q_0) \\
q_2 & := & s(q_1) \mid s(q_2)
\end{array}
$$

Concurrent readers/writers: automata construction

$$\begin{array}{llcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}$$

$$\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0)
\end{array}$$

Concurrent readers/writers: automata construction

$$\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
 & \multicolumn{3}{l}{\mathsf{state}(0,0) \in q \Rightarrow \mathsf{state}(0,s(0)) \in q} \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
\\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
\\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}$$

$$\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0)
\end{array}$$

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
 & \multicolumn{3}{l}{\mathsf{state}(0,0) \in q \Rightarrow \mathsf{state}(0,s(0)) \in q} \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \\
q_1 & := & s(q_0)
\end{array}
$$

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
 & & & \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
 & \mathsf{state}(q_0,0) \in q & \Rightarrow & \mathsf{state}(s(q_0),0) \in q \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
 & & & \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \\
q_1 & := & s(q_0)
\end{array}
$$

Concurrent readers/writers: automata construction

$$\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
\\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
 & \mathsf{state}(q_0,0) \in q & \Rightarrow & \mathsf{state}(s(q_0),0) \in q \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
\\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}$$

$$\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \\
q_1 & := & s(q_0)
\end{array}$$

Concurrent readers/writers: automata construction

$$
\begin{array}{lllll}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
\\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
 & \multicolumn{3}{l}{\mathsf{state}(q_1,0) \in q \Rightarrow \mathsf{state}(s(q_1),0) \in q} \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
\\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \\
q_1 & := & s(q_0)
\end{array}
$$

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
 & & & \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
 & \multicolumn{3}{l}{\mathsf{state}(q_1,0) \in q \Rightarrow \mathsf{state}(s(q_1),0) \in q} \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
 & & & \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \mid \mathsf{state}(q_2,q_0) \\
q_1 & := & s(q_0)
\end{array}
$$

System Timbuk [Thomas Genet]. Automated construction, with guess of *accelaration* $q_2 := s(q_2)$ by user assistance.

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\[1em]
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
 & \multicolumn{3}{l}{\mathsf{state}(q_2,0) \in q \Rightarrow \mathsf{state}(s(q_2),0) \in q} \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\[1em]
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \mid \mathsf{state}(q_2,q_0) \\
q_1 & := & s(q_0)
\end{array}
$$

System Timbuk [Thomas Genet]. Automated construction, with guess of *accelaration* $q_2 := s(q_2)$ by user assistance.

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
\\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
\\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
 & \multicolumn{3}{l}{\mathsf{state}(q_0,s(q_0)) \in q \Rightarrow \mathsf{state}(q_0,q_0) \in q} \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \mid \mathsf{state}(q_2,q_0) \\
q_1 & := & s(q_0) \\
q_2 & := & s(q_1) \mid s(q_2)
\end{array}
$$

System Timbuk [Thomas Genet]. Automated construction, with guess of *accelaration* $q_2 := s(q_2)$ by user assistance.

Concurrent readers/writers: automata construction

$$
\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
\\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
\\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
\\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w) \\
 & \multicolumn{3}{l}{\mathsf{state}(s(q_0 \mid q_1 \mid q_2), q_0) \in q \Rightarrow \mathsf{state}(q_0 \mid q_1 \mid q_2, q_0) \in q}
\end{array}
$$

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q & := & \mathsf{state}(q_0,q_0) \mid \mathsf{state}(q_0,q_1) \mid \mathsf{state}(q_1,q_0) \mid \mathsf{state}(q_2,q_0) \\
q_1 & := & s(q_0) \\
q_2 & := & s(q_1) \mid s(q_2)
\end{array}
$$

System Timbuk [Thomas Genet]. Automated construction, with guess of *accelaration* $q_2 := s(q_2)$ by user assistance.

Concurrent readers/writers: verification

Properties expected:

1. mutual exclusion between readers and writers
   forbidden pattern: $\mathsf{state}(s(x), s(y))$
2. mutual exclusion between writers
   forbidden pattern: $\mathsf{state}(x, s(s(y)))$

The red set: union of

1. $\mathsf{state}\big((q_1 \mid q_2), (q_1 \mid q_2)\big)$
2. $\mathsf{state}\big((q_0 \mid q_1 \mid q_2), (q_1 \mid q_2)\big)$

with $q_0 := 0$, $q_1 := s(q_0)$, $q_2 := s(q_1) \mid s(q_2)$

Verification: The intersection between the set of reachable configurations and the red set is empty.

Functional program

Lists built with *constructor* symbols cons and nil.

$$\begin{array}{rcl} \mathsf{app}(\mathsf{nil}, y) & = & y \\ \mathsf{app}\big(\mathsf{cons}(x, y), z\big) & = & \mathsf{cons}\big(x, \mathsf{app}(y, z)\big) \end{array}$$

Functional program analysis

set of initial configurations q_{app}: terms of the form $\mathsf{app}(\ell_1, \ell_2)$ where ℓ_1, ℓ_2 are lists of 0 and 1, defined by

$$\begin{array}{rcl} q & := & 0 \mid 1 \\ q_\ell & := & \mathsf{nil} \mid \mathsf{cons}(q, q_\ell) \\ q_{\mathsf{app}} & := & \mathsf{app}(q_\ell, q_\ell) \end{array}$$

set of reachable configurations = the closure according to

$$\begin{array}{rcl} \mathsf{app}(\mathsf{nil}, y) & = & y \\ \mathsf{app}\big(\mathsf{cons}(x, y), z\big) & = & \mathsf{cons}\big(x, \mathsf{app}(y, z)\big) \end{array}$$

it is

$$\begin{array}{rcl} q & := & 0 \mid 1 \\ q_\ell & := & \mathsf{nil} \mid \mathsf{cons}(q, q_\ell) \\ q_{\mathsf{app}} & := & \mathsf{app}(q_\ell, q_\ell) \mid \mathsf{cons}(q, q_{\mathsf{app}}) \end{array}$$

Functional program : rev

[Thomas Genet, Valérie Viet Triem Tong, LPAR 01]. Timbuk.

$$\begin{array}{rcl} \mathsf{app}(\mathsf{nil}, y) & = & y \\ \mathsf{app}\big(\mathsf{cons}(x, y), z\big) & = & \mathsf{cons}\big(x, \mathsf{app}(y, z)\big) \\ \mathsf{rev}(\mathsf{nil}) & = & \mathsf{nil} \\ \mathsf{rev}\big(\mathsf{cons}(x, y)\big) & = & \mathsf{app}\big(\mathsf{rev}(y), \mathsf{cons}(x, \mathsf{nil})\big) \end{array}$$

set of initial config.:

$$\begin{array}{rcl} q_0 & := & 0 \\ q_1 & := & 1 \\ q_{\ell_1} & := & \mathsf{nil} \mid \mathsf{cons}(q_1, q_{\ell_1}) \\ q_{\ell_{01}} & := & \mathsf{nil} \mid \mathsf{cons}(q_0, q_{\ell_1}) \mid \mathsf{cons}(q_0, q_{\ell_{01}}) \\ q_{\mathsf{rev}} & := & \mathsf{rev}(q_{\ell_{01}}) \end{array}$$

Functional program : rev

[Thomas Genet, Valérie Viet Triem Tong, LPAR 01]. Timbuk.

$$\begin{array}{rcl}
\mathsf{app}(\mathsf{nil}, y) & = & y \\
\mathsf{app}\big(\mathsf{cons}(x, y), z\big) & = & \mathsf{cons}\big(x, \mathsf{app}(y, z)\big) \\
\mathsf{rev}(\mathsf{nil}) & = & \mathsf{nil} \\
\mathsf{rev}\big(\mathsf{cons}(x, y)\big) & = & \mathsf{app}\big(\mathsf{rev}(y), \mathsf{cons}(x, \mathsf{nil})\big)
\end{array}$$

set of initial config.: $\mathsf{rev}(\ell)$ where $\ell \in q_{\ell_{01}}$, list of 0's followed by 1's

$$\begin{array}{rcl}
q_0 & := & 0 \\
q_1 & := & 1 \\
q_{\ell_1} & := & \mathsf{nil} \mid \mathsf{cons}(q_1, q_{\ell_1}) \\
q_{\ell_{01}} & := & \mathsf{nil} \mid \mathsf{cons}(q_0, q_{\ell_1}) \mid \mathsf{cons}(q_0, q_{\ell_{01}}) \\
q_{\mathsf{rev}} & := & \mathsf{rev}(q_{\ell_{01}})
\end{array}$$

Functional program cntd

set of reachable configurations: by completion of equations for initial configurations

$$
\begin{array}{rcl}
q_0 & := & 0 \\
q_1 & := & 1 \\
q_{\ell_1} & := & \mathsf{nil} \mid \mathsf{cons}(q_1, q_{\ell_1}) \mid \mathsf{cons}(q_1, q_{\mathsf{nil}}) \mid \mathsf{app}(q_{\mathsf{nil}}, q_{\ell_1}) \\
q_{\ell_{01}} & := & \mathsf{nil} \mid \mathsf{cons}(q_0, q_{\ell_1}) \mid \mathsf{cons}(q_0, q_{\ell_{01}}) \\
q_{\mathsf{rev}} & := & \mathsf{rev}(q_{\ell_{01}}) \mid \mathsf{nil} \mid \mathsf{app}(q_{\ell_{10}}, q_{\mathsf{nil}}) \\
q_{\ell_{10}} & := & \mathsf{rev}(q_{\ell_{01}}) \mid \mathsf{app}(q_{\ell_1}, q_{\ell_0}) \\
q_{\mathsf{nil}} & := & \mathsf{nil} \mid \mathsf{rev}(q_{\mathsf{nil}}) \\
q_{\ell_0} & := & \mathsf{cons}(q_0, q_{\mathsf{nil}}) \mid \mathsf{app}(q_{\mathsf{nil}}, q_{\ell_0}) \mid \mathsf{app}(q_{\ell_0}, q_{\ell_0})
\end{array}
$$

property expected: $\mathsf{rev}(\ell)$ not reachable when $\ell \models \exists x, y\ x < y \wedge 0(x) \wedge 1(y)$.

verification The intersection of q_{rev} and the above set is empty.

Imperative programs

$$p ::= 0 \mid X \mid p \cdot p \mid p \parallel p$$

- 0: null process (termination)
- X: program point
- $p \cdot p$: sequential composition
- $p \parallel p$: parallel composition

Transition rules

- procedure call: $X \to Y \cdot Z \quad (Z =$ return point$)$
- procedure call with global state: $Q \cdot X \to Q' \cdot Y \cdot Z$
- procedure return: $Q \cdot Y \to Q'$
- global state change: $Q \cdot X \to Q' \cdot X$
- dynamic thread creation: $X \to Y \| Z$
- handshake : $X \| Y \to X' \| Y'$

Imperative program

[Bouajjani Touili CAV 02]

```
void X() {
  while(true) {
    if Y() {
    thread_create(&t1,Z)
    } else { return }
    }
  }
```

$$
\begin{array}{llll}
\mathtt{X} & \rightarrow & \mathtt{Y} \cdot \mathtt{X} & (r_1) \\
\mathtt{Y} & \rightarrow & \mathtt{t} & (r_2) \\
\mathtt{Y} & \rightarrow & \mathtt{f} & (r_3) \\
\mathtt{t} \cdot \mathtt{X} & \rightarrow & \mathtt{X} \parallel \mathtt{Z} & (r_4) \\
\mathtt{f} & \rightarrow & 0 & (r_5)
\end{array}
$$

The set of reachable configurations is infinite but regular.

Related models of imperative programs

- Pushdown systems (sequential programs with procedure calls)

$$X_1 \cdot \ldots \cdot X_n \to Y_1 \cdot \ldots \cdot Y_m$$

- Petri nets (multi-threaded programs)

$$X_1 \parallel \ldots \parallel X_n \to Y_1 \parallel \ldots \parallel Y_m$$

- PA processes [Lugiez Schnoebelen TCS 02]

$$X_1 \to Y_1 \cdot \ldots \cdot Y_m, \quad X_1 \to Y_1 \parallel \ldots \parallel Y_m$$

- Process rewrite systems (PRS) [Mayr Rusinowitch IC 99], [Bouajjani Touili RTA 05]

$$X_1 \cdot \ldots \cdot X_n \to Y_1 \cdot \ldots \cdot Y_m, \quad X_1 \parallel \ldots \parallel X_n \to Y_1 \parallel \ldots \parallel Y_m$$

- Dynamic pushdown networks [Seidl CIAA 09]

Tree languages modulo

In the above model,

- $\cdot$ is associative,
- $\|$ is associative and commutative.

The terms of the above algebra correspond to unranked trees,

- ordered (modulo A) and
- unordered (modulo AC).

(models for XML processing)

Overview

Verification of other infinite-states systems.

- configuration $=$ tree (ranked or unranked)
  - process,
  - message exchanged in a protocol,
  - local network with a tree shape,
  - tree data structure in memory, with pointers (e.g. binary search trees)...
- (infinite) set of configurations $=$ tree language L
- transition relation between configurations
- safety: $\mathit{transitive\ closure}(L_{\mathsf{init}}) \cap L_{\mathsf{error}} = \emptyset$.

Different kinds of trees

- finite ranked trees (terms in first order logic)
- finite unranked ordered trees
- finite unranked unordered trees
- infinite trees...

$\Rightarrow$ several classes of tree automata.

Overview: properties of automata

- determinism,
- Boolean closures,
- closures under transformations
  (homomorphismes, transducers, rewrite systems...)
- minimization,
- decision problems, complexity,
  - membership,
  - emptiness,
  - universality,
  - inclusion, equivalence,
  - emptiness of intersection,
  - finiteness...
- pumping and star lemma,
- expressiveness, correspondence with logics.

Organization of the tutorial

1. finite ranked tree automata
   - properties
   - algorithms
   - closure under transformation,
     applications to program verification
2. correspondence with the monadic second order logic of the tree (Thatcher and Wright's theorem).
3. finite unranked tree automata
   - ordered = Hedge Automata
   - unordered = Presburger automata
   - closure modulo A and AC
   - XML typing and analysis of transformations
4. tree automata as Horn clause sets

Part I

Automata on Finite Ranked Trees

Terms in first order logic

Plan

Terms

TA: Definitions and Expressiveness

Determinism and Boolean Closures

Decision Problems

Minimization

Closure under Tree Transformations, Program Verification

Signature

Definition : Signature

A signature Σ is a finite set of function symbols each of them with an arity greater or equal to 0.

We denote Σ_i the set of symbols of arity i.

Example :

$\{+ : 2, s : 1, 0 : 0\}$, $\{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$.

We also consider a countable set $\mathcal{X}$ of variable symbols.

Terms

Definition : Term

The set of terms over the signature Σ and $\mathcal{X}$ is the smallest set $\mathcal{T}(\Sigma, \mathcal{X})$ such that:

- $\Sigma_0 \subseteq \mathcal{T}(\Sigma, \mathcal{X})$,
- $\mathcal{X} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$,
- if $f \in \Sigma_n$ and if $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})$.

The set of ground terms (terms without variables, i.e. $\mathcal{T}(\Sigma, \emptyset)$) is denoted $\mathcal{T}(\Sigma)$.

Example :

x, $\neg(x)$, $\wedge\big(\vee(x, \neg(y)), \neg(x)\big)$.

Terms (2)

A term where each variable appears at most once is called linear.
A term without variable is called ground.

Depth $\mathsf{h}(t)$:

- $\mathsf{h}(a) = \mathsf{h}(x) = 0$ if $a \in \Sigma_0$, $x \in \mathcal{X}$,
- $\mathsf{h}\big(f(t_1, \ldots, t_n)\big) = \max\{\mathsf{h}(t_1), \ldots, \mathsf{h}(t_n)\} + 1$.

Positions

A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ can also be seen as a function from the set of its positions $\mathcal{P}os(t)$ into $\Sigma \cup \mathcal{X}$.
The empty position (root) is denoted ε.

$\mathcal{P}os(t)$ is a subset of $\mathbb{N}^*$ satisfying the following properties:

- $\mathcal{P}os(t)$ is closed under prefix,
- for all $p \in \mathcal{P}os(t)$ such that $t(p) \in \Sigma_n$ $(n \geq 1)$, $\{pj \in \mathcal{P}os(t) \mid j \in \mathbb{N}\} = \{p1, ..., pn\}$,
- every $p \in \mathcal{P}os(t)$ such that $t(p) \in \Sigma_0 \cup \mathcal{X}$ is maximal in $\mathcal{P}os(t)$ for the prefix ordering.

The size of t is defined by $\|t\| = |\mathcal{P}os(t)|$.

Subterm $t|_p$ at position $p \in \mathcal{P}os(t)$:

- $t|_\varepsilon = t$,
- $f(t_1, \ldots, t_n)|_{ip} = t_i|_p$.

The replacement in t of $t|_p$ by s is denoted $t[s]_p$.

Positions (Example)

Example :

$t = \wedge(\wedge(x, \vee(x, \neg(y))), \neg(x))$,
$t|_{11} = x$, $t|_{12} = \vee(x, \neg(y))$, $t|_2 = \neg(x)$,
$t[\neg(y)]_{11} = \wedge(\wedge(\neg(y), \vee(x, \neg(y))), \neg(x))$.

Contexts

Definition : Contexte

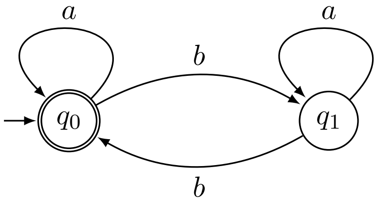
A *context* is a linear term.

The application of a context $C \in \mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$ to n terms $t_1, \ldots, t_n$, denoted $C[t_1, \ldots, t_n]$, is obtained by the replacement of each x_i by t_i, for $1 \leq i \leq n$.

Plan

Bottom-up Finite Tree Automata

$(a + b\, a^* b)^*$



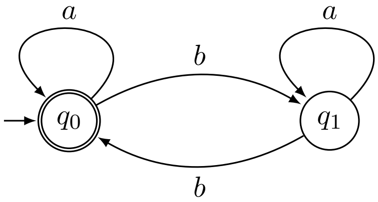
word. run on $aabba$: $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_0$.

tree. run on $a(a(b(b(a(\varepsilon)))))$:
$q_0 \rightarrow a(q_0) \rightarrow a(a(q_0)) \rightarrow a(a(b(q_1))) \rightarrow a(a(b(b(q_0)))) \rightarrow a(a(b(b(a(q_0))))) \rightarrow a(a(b(b(a(\varepsilon)))))$

with $q_0 := \varepsilon$, $q_0 := a(q_0)$, $q_1 := a(q_1)$, $q_1 := b(q_0)$, $q_0 := b(q_1)$.

Bottom-up Finite Tree Automata

$(a + b\,a^*b)^*$



word. run on $aabba$: $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_0$.

tree. run on $a(a(b(b(a(\varepsilon)))))$:
$a(a(b(b(a(\varepsilon))))) \to a(a(b(b(a(q_0))))) \to a(a(b(b(q_0)))) \to$
$a(a(b(q_1))) \to a(a(q_0)) \to a(q_0) \to q_0$

with $\varepsilon \to q_0$, $a(q_0) \to q_0$, $a(q_1) \to q_1$, $b(q_0) \to q_1$, $b(q_1) \to q_0$.

Bottom-up Finite Tree Automata

Definition : Tree Automata

A *tree automaton* (TA) over a signature Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where Q is a finite set of *states*, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $f(q_1, \ldots, q_n) \to q$ with $f \in \Sigma_n$ $(n \geq 0)$ and $q_1, \ldots, q_n, q \in Q$.

The state q is called the head of the rule.
The language of $\mathcal{A}$ in state q is recursively defined by

$$\begin{array}{rcl} L(\mathcal{A}, q) & = & \left\{a \in \Sigma_0 \;\middle|\; a \to q \in \Delta\right\} \\ & \cup & \displaystyle\bigcup_{f(q_1,\ldots,q_n) \to q \in \Delta} f\big(L(\mathcal{A}, q_1), \ldots, L(\mathcal{A}, q_n)\big) \end{array}$$

with $f(L_1, \ldots, L_n) := \left\{f(t_1, \ldots, t_n) \;\middle|\; t_1 \in L_1, \ldots, t_n \in L_n\right\}$.

We say that $t \in L(\mathcal{A}, q)$ is accepted, or recognized, by $\mathcal{A}$ in state q.

The language of $\mathcal{A}$ is $L(\mathcal{A}) := \displaystyle\bigcup_{q^{\mathsf{f}} \in Q^{\mathsf{f}}} L(\mathcal{A}, q^{\mathsf{f}})$ (regular language).

Recognized Languages: Operational Definition

Rewrite Relation

The rewrite relation associated to Δ is the smallest binary relation, denoted $\xrightarrow[\Delta]{}$, containing Δ and closed under application of contexts.

The reflexive and transitive closure of $\xrightarrow[\Delta]{}$ is denoted $\xrightarrow[\Delta]{*}$.

For $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$, it holds that

$$L(\mathcal{A}, q) = \left\{ t \in \mathcal{T}(\Sigma) \;\middle|\; t \xrightarrow[\Delta]{*} q \right\}$$

and hence

$$L(\mathcal{A}) = \left\{ t \in \mathcal{T}(\Sigma) \;\middle|\; t \xrightarrow[\Delta]{*} q \in Q^{\mathsf{f}} \right\}$$

Tree Automata: Example 1

Example :

$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$,

$$\mathcal{A} = \left(\Sigma, \{q_0, q_1\}, \{q_1\}, \left\{ \begin{array}{rclrcl} \bot & \rightarrow & q_0 & \top & \rightarrow & q_1 \\ \neg(q_0) & \rightarrow & q_1 & \neg(q_1) & \rightarrow & q_0 \\ \vee(q_0, q_0) & \rightarrow & q_0 & \vee(q_0, q_1) & \rightarrow & q_1 \\ \vee(q_1, q_0) & \rightarrow & q_1 & \vee(q_1, q_1) & \rightarrow & q_1 \\ \wedge(q_0, q_0) & \rightarrow & q_0 & \wedge(q_0, q_1) & \rightarrow & q_0 \\ \wedge(q_1, q_0) & \rightarrow & q_0 & \wedge(q_1, q_1) & \rightarrow & q_1 \end{array} \right\} \right)$$

$$\begin{array}{rl} & \wedge(\wedge(\top, \vee(\top, \neg(\bot))), \neg(\top)) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(\top, \vee(\top, \neg(\bot))), \neg(q_1)) \\ \xrightarrow[\mathcal{A}]{} & \wedge(\wedge(q_1, \vee(q_1, \neg(q_0))), \neg(q_1)) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(q_1, \vee(q_1, \neg(q_0))), q_0) \\ \xrightarrow[\mathcal{A}]{} & \wedge(\wedge(q_1, \vee(q_1, q_1)), q_0) \xrightarrow[\mathcal{A}]{} \wedge(\wedge(q_1, q_1), q_0) \xrightarrow[\mathcal{A}]{} \wedge(q_1, q_0) \xrightarrow[\mathcal{A}]{} q_0 \end{array}$$

Tree Automata: Example 2

Example :

$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \perp : 0\}$,
TA recognizing the ground instances of $\neg(\neg(x))$:

$$\mathcal{A} = \left(\Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \left\{ \begin{array}{rcl rcl} \perp & \to & q & \top & \to & q \\ \neg(q) & \to & q & \neg(q) & \to & q_\neg \\ \neg(q_\neg) & \to & q_\mathsf{f} & & & \\ \vee(q,q) & \to & q & \wedge(q,q) & \to & q \end{array} \right\} \right)$$

Example :

Ground terms embedding the pattern $\neg(\neg(x))$: $\mathcal{A} \cup \{\neg(q_\mathsf{f}) \to q_\mathsf{f}, \vee(q_\mathsf{f}, q_*) \to q_\mathsf{f}, \vee(q_*, q_\mathsf{f}) \to q_\mathsf{f}, \ldots\}$ (propagation of q_f).

Linear Pattern Matching

Proposition :

Given a linear term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, there exists a TA $\mathcal{A}$ recognizing the set of ground instances of t: $L(\mathcal{A}) = \left\{ t\sigma \;\middle|\; \sigma : \mathcal{X} \to \mathcal{T}(\Sigma) \right\}$.

e.g. in regular tree model checking, definition of error configurations by forbidden patterns.

Runs

Definition : Run

A *run* of a TA $(\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ on a term $t \in \mathcal{T}(\Sigma)$ is a function $r : \mathcal{P}os(t) \to Q$ such that for all $p \in \mathcal{P}os(t)$,
if $t(p) = f \in \Sigma_n$, $r(p) = q$ and $r(pi) = q_i$ for all $1 \leq i \leq n$,
then $f(q_1, \ldots, q_n) \to q \in \Delta$.

The run r is *accepting* if $r(\varepsilon) \in Q^{\mathsf{f}}$.
$L(\mathcal{A})$ is the set of ground terms of $\mathcal{T}(\Sigma)$ for which there exists an accepting run.

Pumping Lemma

Lemma : Pumping Lemma

Let $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$.
$L(\mathcal{A}) \neq \emptyset$ iff there exists $t \in L(\mathcal{A})$ such that $h(t) \leq |Q|$.

Lemma : Iteration Lemma

For all TA $\mathcal{A}$, there exists $k > 0$ such that for all term $t \in L(\mathcal{A})$ with $h(t) > k$, there exists 2 contexts $C, D \in \mathcal{T}(\Sigma, \{x_1\})$ with $D \neq x_1$ and a term $u \in \mathcal{T}(\Sigma)$ such that $t = C\big[D[u]\big]$ and for all $n \geq 0$, $C\big[D^n[u]\big] \in L(\mathcal{A})$.

<u>usage</u>: to show that a language is not regular.

Non Regular Languages

We show with the pumping and iteration lemmatas that the following tree languages are not regular:

- $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$,
- $\{f(g^n(a), h^n(a)) \mid n \geq 0\}$,
- $\{t \in \mathcal{T}(\Sigma) \mid |\mathcal{P}os(t)| \text{ is prime}\}$.

Epsilon-Transitions

We extend the class TA into TAε with the addition of another type of transition rules of the form $q \xrightarrow{\varepsilon} q'$ (ε-transition).
with the same expressiveness as TA.

Proposition : Suppression of ε-transitions

For all TAε $\mathcal{A}_\varepsilon$, there exists a TA (without ε-transition) $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}_\varepsilon)$. The size of $\mathcal{A}$ is polynomial in the size of $\mathcal{A}_\varepsilon$.

pr.: We start with $\mathcal{A}_\varepsilon$ and we add $f(q_1, \ldots, q_n) \to q'$ if there exists $f(q_1, \ldots, q_n) \to q$ and $q \xrightarrow{\varepsilon} q'$.

Top-Down Tree Automata

Definition : Top-Down Tree Automata

A top-down tree automaton over a signature Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{init}}, \Delta)$ where Q is a finite set of *states*, $Q^{\mathsf{init}} \subseteq Q$ is the subset of initial states and Δ is a set of transition rules of the form: $q \to f(q_1, \ldots, q_n)$ with $f \in \Sigma_n$ $(n \geq 0)$ and $q_1, \ldots, q_n, q \in Q$.

A ground term $t \in \mathcal{T}(\Sigma)$ is accepted by $\mathcal{A}$ in the state q iff $q \xrightarrow[\Delta]{*} t$.

The language of $\mathcal{A}$ starting from the state q is
$L(\mathcal{A}, q) := \left\{ t \in \mathcal{T}(\Sigma) \;\middle|\; q \xrightarrow[\Delta]{*} t \right\}$.

The language of $\mathcal{A}$ is $L(\mathcal{A}) := \bigcup_{q^{\mathsf{i}} \in Q^{\mathsf{init}}} L(Q, q^{\mathsf{i}})$.

Top-Down Tree Automata (Expressiveness)

Proposition : Expressiveness

The set of top-down tree automata languages is exactly the set of regular tree languages.

Remark: Notations

In the next slides

$$\text{TA} = \text{Bottom-Up Tree Automata}$$

Plan

Determinism

Definition : Determinism

A TA $\mathcal{A}$ is *deterministic* if for all $f \in \Sigma_n$, for all states $q_1, \ldots, q_n$ of $\mathcal{A}$, there is at most one state q of $\mathcal{A}$ such that $\mathcal{A}$ contains a transition $f(q_1, \ldots, q_n) \to q$.

If $\mathcal{A}$ is deterministic, then for all $t \in \mathcal{T}(\Sigma)$, there exists at most one state q of $\mathcal{A}$ such that $t \in L(\mathcal{A}, q)$. It is denoted $\mathcal{A}(t)$ or $\Delta(t)$.

Completeness

Definition : Completeness

A TA $\mathcal{A}$ is *complete* if for all $f \in \Sigma_n$, for all states $q_1, \ldots, q_n$ of $\mathcal{A}$, there is at least one state q of $\mathcal{A}$ such that $\mathcal{A}$ contains a transition $f(q_1, \ldots, q_n) \to q$.

If $\mathcal{A}$ is complete, then for all $t \in \mathcal{T}(\Sigma)$, there exists at least one state q of $\mathcal{A}$ such that $t \in L(\mathcal{A}, q)$.

Completion

Proposition : Completion

For all TA $\mathcal{A}$, there exists a complete TA $\mathcal{A}_c$ such that $L(\mathcal{A}_c) = L(\mathcal{A})$. Moreover, if $\mathcal{A}$ is deterministic, then $\mathcal{A}_c$ is deterministic. The size of $\mathcal{A}_c$ is polynomial in the size of $\mathcal{A}$, its construction is PTIME.

Completion

Proposition : Completion

For all TA $\mathcal{A}$, there exists a complete TA $\mathcal{A}_c$ such that $L(\mathcal{A}_c) = L(\mathcal{A})$. Moreover, if $\mathcal{A}$ is deterministic, then $\mathcal{A}_c$ is deterministic. The size of $\mathcal{A}_c$ is polynomial in the size of $\mathcal{A}$, its construction is PTIME.

pr.: add a trash state $q_\perp$.

Determinization

Proposition : Determinization

For all TA $\mathcal{A}$, there exists a deterministic TA $\mathcal{A}_{det}$ such that $L(\mathcal{A}_{det}) = L(\mathcal{A})$. Moreover, if $\mathcal{A}$ is complete, then $\mathcal{A}_{det}$ is complete. The size of $\mathcal{A}_{det}$ is exponential in the size of $\mathcal{A}$, its construction is EXPTIME.

pr.: subset construction. Transitions:

$$f(S_1, \ldots, S_n) \to \{q \mid \exists q_1 \in S_1 \ldots \exists q_n \in S_n\ f(q_1, \ldots, q_n \to q \in \Delta\}$$

for all $S_1, \ldots, S_n \subseteq Q$.

Determinization (Example)

Exercice :

Determinise and complete the previous TA (pattern matching of $\neg(\neg(x))$):

$$\mathcal{A} = \left(\Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \left\{ \begin{array}{rclrcl} \bot & \rightarrow & q & \top & \rightarrow & q \\ \neg(q) & \rightarrow & q & \neg(q) & \rightarrow & q_\neg \\ \neg(q_\neg) & \rightarrow & q_\mathsf{f} & \neg(q_\mathsf{f}) & \rightarrow & q_\mathsf{f} \\ \vee(q, q) & \rightarrow & q & \wedge(q, q) & \rightarrow & q \\ \vee(q_\mathsf{f}, q_*) & \rightarrow & q_\mathsf{f} & \vee(q_*, q_\mathsf{f}) & \rightarrow & q_\mathsf{f} \end{array} \right\} \right)$$

Top-Down Tree Automata and Determinism

Definition : Determinism

A top-down tree automaton $(\Sigma, Q, Q^{\mathsf{init}}, \Delta)$ is *deterministic* if $|Q^{\mathsf{init}}| = 1$ and for all state $q \in Q$ and $f \in \Sigma$, Δ contains at most one rule with left member q and symbol f.

The top-down tree automata are in general not determinizable .

Proposition :

There exists a regular tree language which is not recognizable by a deterministic top-down tree automaton.

Top-Down Tree Automata and Determinism

Definition : Determinism

A top-down tree automaton $(\Sigma, Q, Q^{\mathsf{init}}, \Delta)$ is *deterministic* if $|Q^{\mathsf{init}}| = 1$ and for all state $q \in Q$ and $f \in \Sigma$, Δ contains at most one rule with left member q and symbol f.

The top-down tree automata are in general not determinizable .

Proposition :

There exists a regular tree language which is not recognizable by a deterministic top-down tree automaton.

pr.: $L = \{f(a, b), f(b, a)\}$.

Boolean Closure of Regular Tree Languages

Proposition : Closure

The class of regular tree languages is closed under union, intersection and complementation.

op.	technique	computation time and size of automata
$\cup$	disjoint $\cup$	
$\cap$	Cartesian product	
$\neg$	determinization, completion, invert final / non-final states	(lower bound)

Remark :

For the deterministic TA, the construction for the complementation is polynomial.

Boolean Closure of Regular Tree Languages

Proposition : Closure

The class of regular tree languages is closed under union, intersection and complementation.

op.	technique	computation time and size of automata
$\cup$	disjoint $\cup$	linear
$\cap$	Cartesian product	
$\neg$	determinization, completion, invert final / non-final states	(lower bound)

Remark :

For the deterministic TA, the construction for the complementation is polynomial.

Boolean Closure of Regular Tree Languages

Proposition : Closure

The class of regular tree languages is closed under union, intersection and complementation.

op.	technique	computation time and size of automata
$\cup$	disjoint $\cup$	linear
$\cap$	Cartesian product	quadratic
$\neg$	determinization, completion, invert final / non-final states	(lower bound)

Remark :

For the deterministic TA, the construction for the complementation is polynomial.

Boolean Closure of Regular Tree Languages

Proposition : Closure

The class of regular tree languages is closed under union, intersection and complementation.

op.	technique	computation time and size of automata
$\cup$	disjoint $\cup$	linear
$\cap$	Cartesian product	quadratic
$\neg$	determinization, completion, invert final / non-final states	exponential (lower bound)

Remark :

For the deterministic TA, the construction for the complementation is polynomial.

Plan

Terms

TA: Definitions and Expressiveness

Determinism and Boolean Closures

Decision Problems

Minimization

Closure under Tree Transformations, Program Verification

Cleaning

Definition : Clean

A state q of a TA $\mathcal{A}$ is called *inhabited* if there exists at least one $t \in L(\mathcal{A}, q)$. A TA is called *clean* if all its states are inhabited.

Proposition : Cleaning

For all TA $\mathcal{A}$, there exists a clean TA $\mathcal{A}_{clean}$ such that $L(\mathcal{A}_{clean}) = L(\mathcal{A})$. The size of $\mathcal{A}_{clean}$ is smaller than the size of $\mathcal{A}$, its construction is PTIME.

pr.: state marking algorithm, running time $O\big(|Q| \times \|\Delta\|\big)$.

State Marking Algorithm

We construct $M \subseteq Q$ containing all the inhabited states.

- start with $M = \emptyset$
- for all $f \in \Sigma$, of arity $n \geq 0$, and all $q_1, \ldots, q_n \in M$ st there exists $f(q_1, \ldots, q_n) \to q$ in Δ, add q to M (if it was not already).

We iterate the last step until a fixpoint M_* is reached.

Lemma :

$q \in M_*$ iff $\exists t \in L(\mathcal{A}, q)$.

Membership Problem

Definition : Membership

INPUT: a TA $\mathcal{A}$ over Σ, a term $t \in \mathcal{T}(\Sigma)$.
QUESTION: $t \in L(\mathcal{A})$?

Proposition : Membership

The membership problem is decidable in polynomial time.

Exact complexity:

- non-deterministic bottom-up: LOGCFL-complete
- deterministic bottom-up: unknown (LOGDCFL)
- deterministic top-down: LOGSPACE-complete.

Emptiness Problem

Definition : Emptiness

INPUT: a TA $\mathcal{A}$ over Σ.
QUESTION: $L(\mathcal{A}) = \emptyset$?

Proposition : Emptiness

The emptiness problem is decidable in linear time.

Emptiness Problem

Definition : Emptiness

INPUT: a TA $\mathcal{A}$ over Σ.
QUESTION: $L(\mathcal{A}) = \emptyset$?

Proposition : Emptiness

The emptiness problem is decidable in linear time.

pr.:
quadratic: clean, check if the clean automaton contains a final state.
linear: reduction to propositional HORN-SAT.
linear bis: optimization of the data structures for the cleaning (exo).

Remark :

The problem of the emptiness is PTIME-complete.

Instance-Membership Problem

Definition : Instance-Membership (IM)

INPUT: a TA $\mathcal{A}$ over Σ, a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$.
QUESTION: does there exists $\sigma : vars(t) \to \mathcal{T}(\Sigma)$ s.t. $t\sigma \in L(\mathcal{A})$?

Proposition : Instance-Membership

1. The problem IM is decidable in polynomial time when t is linear.
2. The problem IM is NP-complet when $\mathcal{A}$ is deterministic.
3. The problem IM is EXPTIME-complete in general.

Problem of the Emptiness of Intersection

Definition : Emptiness of Intersection

INPUT: n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over Σ.
QUESTION: $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$?

Proposition : Emptiness of Intersection

The problem of the emptiness of intersection is EXPTIME-complete.

Problem of the Emptiness of Intersection

Definition : Emptiness of Intersection

INPUT: n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over Σ.
QUESTION: $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$?

Proposition : Emptiness of Intersection

The problem of the emptiness of intersection is EXPTIME-complete.

pr.: EXPTIME: n applications of the closure under $\cap$ and emptiness decision.

EXPTIME-hardness: APSPACE $=$ EXPTIME
reduction of the problem of the existence of a successful run (starting from an initial configuration) of an alternating Turing machine (ATM) $M = (\Gamma, S, s_0, S_\mathsf{f}, \delta)$.
[Seidl 94], [Veanes 97]

Let $M = (\Gamma, S, s_0, S_\mathsf{f}, \delta)$ be a Turing Machine (Γ: input alphabet, S: state set, s_0 initial state, S_f final states, δ: transition relation). First some notations.

- a *configuration* of M is a word of $\Gamma^*\Gamma_S\Gamma^*$ where $\Gamma_S = \{a^s \mid a \in \Gamma, s \in S\}$. In this word, the letter of Γ_S indicates both the current state and the current position of the head of M.
- a *final configuration* of M is a word of $\Gamma^*\Gamma_{S_\mathsf{f}}\Gamma^*$.
- an *initial configuration* of M is a word of $\Gamma_{s_0}\Gamma^*$.
- a *transition* of M (following δ) between two configurations v and v' is denoted $v \rhd v'$

The initial configuration v_0 is accepting iff there exists a final configuration v_f and a finite sequence of transitions $v_0 \rhd \ldots \rhd v_\mathsf{f}$?
This problem whether v_0 is accepting is undecidable in general.
If the tape is polynomially bounded (we are restricted to configurations of length $n = |v_0|^c$, for some fixed $c \in \mathbb{N}$), the problem is PSPACE complete.
M alternating: $S = S_\exists \uplus S_\forall$.
Definition accepting configurations:

- every final configuration (whose state is in S_f) is accepting
- a configuration c whose state is in $S_\exists$ is accepting if it has at least one successor accepting
- a configuration c whose state is in $S_\forall$ is accepting if all its successors are accepting

Theorem (Chandra, Kozen, Stockmeyer 81)

APSPACE = *EXPTIME*

In order to show EXPTIME-hardness, we reduce the problem of deciding whether v_0 is accepting for M alternating and polynomially bounded.
Hypotheses (non restrictive):

- $s_0 \in S_\exists$ or $s_0 \in S_\forall \cap S_\mathsf{f}$
- s_0 is non reentering (it only occurs in v_0)
- every configuration with state in $S_\forall$ has 0 or 2 successors
- final configurations are restricted to $\flat_{S_\mathsf{f}} \flat^*$ where $\flat \in \Gamma$ is the blank symbol.
- S_f is a singleton.

2 technical definitions: for $k \leq n$,

$$\text{view}(v,k) = \begin{array}{ll} v[k]v[k+1] & \text{if } k=1 \\ v[k-1]v[k] & \text{if } k=n \\ v[k-1]v[k]v[k+1] & \text{otherwise} \end{array}$$

$$\text{view}(v,v_1,v_2,k) = \langle \text{view}(v,k), \text{view}(v_1,k), \text{view}(v_2,k) \rangle$$

$v \rhd_k \langle v_1, v_2 \rangle$ iff

1. if $v[k] \in \Gamma_S$, then $\exists w \rhd w_1, w_2$ s.t. $\text{view}(v,v_1,v_2,k) = \text{view}(w,w_1,w_2,k)$
2. if $v[k] = a \in \Gamma$, then $v_1[k] \in \{a\} \cup a_S$ and $v_2 = \varepsilon$ or $v_2[k] \in \{a\} \cup a_S$.

first item: around position k, we have two correct transitions of M. This can be tested by the membership of $\text{view}(v,v_1,v_2,k)$ to a given set which only depends on M.

Lemma

$v \rhd v_1, v_2$ *iff* $\forall k \leq n \ v \rhd_k \langle v_1, v_2 \rangle$.

Term representations of runs:
rem. a run of M is not a sequence of configurations but a tree of configurations (because of alternation).
Signature Σ: $\emptyset$: constant, Γ: unary, S: unaires, p binary.
Notation: if $v = a_1 \dots a_n$, $v(x)$ denotes $a_n(a_{n-1}(\dots a_1(x)))$.
Term representations of runs:

- $v_{\mathsf{f}}(p(\emptyset, \emptyset))$ with v_{f} final configuration,
- $v(p(t_1, t_2))$ with v $\forall$-configuration, $t_1 = v_1'(p(t_{1,1}, t_{1,2}))$, $t_2 = v_2'(p(t_{2,1}, t_{2,2}))$ are two term representations of runs, and $v_1 \rhd v_1'$, $v_2 \rhd v_2'$
- $v(p(t_1, \emptyset))$ with v $\exists$-configuration, $t_1 = v_1'(p(t_{1,1}, t_{1,2}))$ term representations of run, and $v_1 \rhd v_1'$.

notations for $t_1 = v_1'(p(t_{1,1}, t_{1,2}))$:

- $\mathsf{head}(t_1) = v_1$
- $\mathsf{left}(t_1) = t_{1,1}$
- $\mathsf{right}(t_1) = t_{1,2}$.

This recursive definition suggest the construction of a TA recognizing term representations of successful runs. The difficulty

is the conditions $v_1 \rhd v'_1$, $v_2 \rhd v'_2$, for which we use the above lemma.
We build $2n$ deterministic automata :
for all $1 < k < n$, $\mathcal{A}_k$ recognizes

- $v_{\mathsf{f}}(p(\emptyset, \emptyset))$ (recall there is only 1 final configuration by hyp.)
- $v(p(t_1, t_2))$ such that $t_1 \neq \emptyset$ and
  - $v \rhd_k \langle \mathsf{head}(t_1), \mathsf{head}(t_2) \rangle$
  - $\mathsf{left}(t_1) \in L(\mathcal{A}_k)$, $\mathsf{right}(t_1) \in L(\mathcal{A}_k) \cup \{\emptyset\}$,
  - $t_2 = \emptyset$ or $\mathsf{left}(t_2) \in L(\mathcal{A}_k)$, $\mathsf{right}(t_2) \in L(\mathcal{A}_k) \cup \{\emptyset\}$

idea: $\mathcal{A}_k$ memorizes $\mathsf{view}(\mathsf{head}(t_1), k)$ and $\mathsf{view}(\mathsf{head}(t_2), k)$ and compare with $\mathsf{view}(v, k)$.
for all $1 < k < n$, $\mathcal{A}'_k$ recognizes the terms $v_0(p(t_1, t_2))$ with $t_1 = t_2 = \emptyset$ (if s_0 universal and final) or $t_2 = \emptyset$ (if s_0 existential, not final) and $t_1, t_2 \in T$, minimal set of terms without s_0 containing

- $\emptyset$
- $v(p(t_1, t_2))$ such that $t_1 \neq \emptyset$ and
  - $v \rhd_k \langle \mathsf{head}(t_1), \mathsf{head}(t_2) \rangle$
  - $\mathsf{left}(t_1) \in T$, $\mathsf{right}(t_1) \in T$,

- $t_2 = \emptyset$ or $\mathsf{left}(t_2) \in T$, $\mathsf{right}(t_2) \in T$

$$\text{representations of successful runs} = \bigcap_{k=1}^{n} L(\mathcal{A}_k) \cap L(\mathcal{A}'_k).$$

Problem of Universality

Definition : Universality

INPUT: a TA $\mathcal{A}$ over Σ.
QUESTION: $L(\mathcal{A}) = \mathcal{T}(\Sigma)$

Proposition : Universality

The problem of universality is EXPTIME-complete.

Problem of Universality

Definition : Universality

INPUT: a TA $\mathcal{A}$ over Σ.
QUESTION: $L(\mathcal{A}) = \mathcal{T}(\Sigma)$

Proposition : Universality

The problem of universality is EXPTIME-complete.

pr.: EXPTIME: Boolean closure and emptiness decision.

EXPTIME-hardness: again APSPACE = EXPTIME.

Remark :

The problem of universality is decidable in polynomial time for the deterministic (bottom-up) TA.

pr.: completion and cleaning.

Problems of Inclusion an Equivalence

Definition : Inclusion

INPUT: two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Definition : Equivalence

INPUT: two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION: $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Equivalence

The problems of inclusion and equivalence are EXPTIME-complete.

Problems of Inclusion an Equivalence

Definition : Inclusion

INPUT: two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Definition : Equivalence

INPUT: two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION: $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Equivalence

The problems of inclusion and equivalence are EXPTIME-complete.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)} = \emptyset$.

Problems of Inclusion an Equivalence

Definition : Inclusion

INPUT:	two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION:	$L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$

Definition : Equivalence

INPUT:	two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ.
QUESTION:	$L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Proposition : Inclusion, Equivalence

The problems of inclusion and equivalence are EXPTIME-complete.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)} = \emptyset$.
EXPTIME-hardness: universality is $\mathcal{T}(\Sigma) = L(\mathcal{A}_2)$?

Remark :

If $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, it is $O\big(\|\mathcal{A}_1\| \times \|\mathcal{A}_2\|\big)$.

Problem of Finiteness

Definition : Finiteness

INPUT:	a TA $\mathcal{A}$
QUESTION:	is $L(\mathcal{A})$ finite?

Proposition : Finiteness

The problem of finiteness is decidable in polynomial time.

Plan

Terms

TA: Definitions and Expressiveness

Determinism and Boolean Closures

Decision Problems

Minimization

Closure under Tree Transformations, Program Verification

Theorem of Myhill-Nerode

Definition :

A *congruence* $\equiv$ on $\mathcal{T}(\Sigma)$ is an equivalence relation such that for all $f \in \Sigma_n$, if $s_1 \equiv t_1, \ldots,\ s_n \equiv t_n$, then $f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n)$.

Given $L \subseteq \mathcal{T}(\Sigma)$, the congruence $\equiv_L$ is defined by:

$$s \equiv_L t \text{ if for all context } C \in \mathcal{T}\big(\Sigma, \{x\}\big),\ C[s] \in L \text{ iff } C[t] \in L.$$

Theorem : Myhill-Nerode

The three following propositions are equivalent:

1. L is regular
2. L is a union of equivalence classes for a congruence $\equiv$ of finite index
3. $\equiv_L$ is a congruence of finite index

Proof Theorem of Myhill-Nerode

$1 \Rightarrow 2.$ $\mathcal{A}$ deterministic, def. $s \equiv_{\mathcal{A}} t$ iff $\mathcal{A}(s) = \mathcal{A}(t)$.

$2 \Rightarrow 3.$ we show that if $s \equiv t$ then $s \equiv_L t$, hence the index of $\equiv_L \leq$ index of $\equiv$ (since we have $\equiv \subseteq \equiv_L$). If $s \equiv t$ then $C[s] \equiv C[t]$ for all $C[\,]$ (induction on C), hence $C[s] \in L$ iff $C[t] \in L$, i.e. $s \equiv_L t$.

$3 \Rightarrow 1.$ we construct $\mathcal{A}_{\mathsf{min}} = (Q_{\mathsf{min}}, Q^{\mathsf{f}}_{\mathsf{min}}, \Delta_{\mathsf{min}})$,

- Q_{min} = equivalence classes of $\equiv_L$,
- $Q^{\mathsf{f}}_{\mathsf{min}} = \{[s] \mid s \in L\}$,
- $\Delta_{\mathsf{min}} = \{f\big([s_1], \ldots, [s_n]\big) \to \big[f(s_1, \ldots, s_n)\big]\}$

Clearly, $\mathcal{A}_{\mathsf{min}}$ is deterministic, and for all $s \in \mathcal{T}(\Sigma)$, $\mathcal{A}_{\mathsf{min}}(s) = [s]_L$, i.e. $s \in L(\mathcal{A}_{\mathsf{min}})$ iff $s \in L$.

Minimization

Corollary :

For all DTA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$, there exists a unique DTA $\mathcal{A}_{\mathsf{min}}$ whose number of states is the index of $\equiv_{L(\mathcal{A})}$ and such that $L(\mathcal{A}_{\mathsf{min}}) = L(\mathcal{A})$.

Minimization

Let $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ be a DTA, we build a deterministic minimal automaton $\mathcal{A}_{\mathsf{min}}$ as in the proof of $3 \Rightarrow 1$ of the previous theorem for $L(\mathcal{A})$ (i.e. Q_{min} is the set of equivalence classes for $\equiv_{L(\mathcal{A})}$).

We build first an equivalence $\approx$ on the states of Q:

- $q \approx_0 q'$ iff $q, q' \in Q^{\mathsf{f}}$ ou $q, q' \in Q \setminus Q^{\mathsf{f}}$.
- $q \approx_{k+1} q'$ iff $q \approx_k q'$ et $\forall f \in \Sigma_n$,
  $\forall q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in Q$ $(1 \leq i \leq n)$,

$$\Delta\big(f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)\big) \approx_k \Delta\big(f(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots,$$

Let $\approx$ be the fixpoint of this construction, $\approx$ is $\equiv_{L(\mathcal{A})}$, hence $\mathcal{A}_{\mathsf{min}} = (\Sigma, Q_{\mathsf{min}}, Q^{\mathsf{f}}_{\mathsf{min}}, \Delta_{\mathsf{min}})$ with :

- $Q_{\mathsf{min}} = \{[q]_\approx \mid q \in Q\}$,
- $Q^{\mathsf{f}}_{\mathsf{min}} = \{[q^{\mathsf{f}}]_\approx \mid q^{\mathsf{f}} \in Q^{\mathsf{f}}\}$,
- $\Delta_{\mathsf{min}} = \big\{f\big([q_1]_\approx, \ldots, [q_n]_\approx\big) \to \big[f(q_1, \ldots, q_n)\big]_\approx\big\}$.

recognizes $L(\mathcal{A})$. and it is smaller than $\mathcal{A}$.

Algebraic Characterization of Regular Languages

Corollary :

A set $L \subseteq \mathcal{T}(\Sigma)$ is regular iff there exists

- a Σ-algebra $\mathcal{Q}$ of finite domain Q,
- an homomorphism $h : \mathcal{T}(\Sigma) \to \mathcal{A}$,
- a subset $Q^{\mathsf{f}} \subseteq Q$ such that $L = h^{-1}(Q^{\mathsf{f}})$.

operations of $\mathcal{Q}$:
for each $f \in \Sigma_n$, there is a function $f^{\mathcal{Q}} : Q^n \to Q$.

Plan

Tree Transformations, Verification

- formalisms for the transformation of terms (languages): rewrite systems, tree homomorphisms, transducers...
  = transitions in an infinite states system,
  = evaluation of programs,
  = transformation of XML documents, updates...
- problem of the type checking:

  given:
  - $L_{\text{in}} \subseteq \mathcal{T}(\Sigma)$, (regular) input language
  - h transformation $\mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$
  - $L_{\text{out}} \subseteq \mathcal{T}(\Sigma')$ (regular) output language

  question: do we have $h(L_{\text{in}}) \subseteq L_{\text{out}}$?

Tree Homomorphisms

Tree Homomorphisms

Definition :

$$h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$$
$$h\big(f(t_1, \dots, t_n)\big) := t_f\{x_1 \leftarrow h(t_1), \dots, x_n \leftarrow h(t_n)\}$$
for $f \in \Sigma_n$, with $t_f \in \mathcal{T}\big(\Sigma', \{x_1, \dots, x_n\}\big)$.

h is called

- *linear* if for all $f \in \Sigma$, t_f is linear,
- *complete* if for all $f \in \Sigma_n$, $vars(t_f) = \{x_1, \dots, x_n\}$,
- *symbol-to-symbol* if for all $f \in \Sigma_n$, $height(t_f) = 1$.

Homomorphisms: Examples

Example : ternary trees $\rightarrow$ binary trees

Let $\Sigma = \{a : 0, b : 0, g : 3\}$, $\Sigma' = \{a : 0, b : 0, f : 2\}$ and $h : \mathcal{T}(\Sigma) \rightarrow \mathcal{T}(\Sigma')$ defined by

- $t_a = a$,
- $t_b = b$,
- $t_g = f(x_1, f(x_2, x_3))$.

$h\big(g(a, g(b, b, b), a)\big) = f(a, f(f(f(b, f(b, b))), a))$

Example : Elimination of the $\wedge$

Let $\Sigma = \{0 : 0, 1 : 0, \neg : 1, \vee : 2, \wedge : 2\}$, $\Sigma' = \{0 : 0, 1 : 0, \neg : 1, \vee : 2\}$ and $h : \mathcal{T}(\Sigma) \rightarrow \mathcal{T}(\Sigma')$ with $t_\wedge = \neg(\vee(\neg(x_1), \neg(x_2)))$.

Closure of Regular Languages under Linear Homomorphisms

Theorem :

If L is regular and h is a linear homomorphism, then $h(L)$ is regular.

Closure of Regular Languages under Linear Homomorphisms

Theorem :

If L is regular and h is a linear homomorphism, then $h(L)$ is regular.

let $\mathcal{A} = (Q, Q^{\mathsf{f}}, \Delta)$ be clean, we build $\mathcal{A}' = (Q', Q'_{\mathsf{f}}, \Delta')$.
For each $r = f(q_1, \ldots, q_n) \to q \in \Delta$, with $t_f \in \mathcal{T}(\Sigma', \mathcal{X}_n)$ (linear),
let $Q^r = \{q^r_p \mid p \in \mathcal{P}os(t_f)\}$, and Δ_r defined as follows:
for all $p \in \mathcal{P}os(t_f)$:

- if $t_f(p) = g \in \Sigma'_m$, then $g(q^r_{p_1}, \ldots, q^r_{p_m}) \to q^r_p \in \Delta_r$,
- if $t_f(p) = x_i$, then $q_i \xrightarrow{\varepsilon} q^r_p \in \Delta_r$,
- $q^r_\varepsilon \xrightarrow{\varepsilon} q \in \Delta_r$.

$$Q' = Q \cup \bigcup_{r \in \Delta} Q^r,$$
$$Q'_{\mathsf{f}} = Q_{\mathsf{f}},$$
$$\Delta' = \bigcup_{r \in \Delta} \Delta_r.$$

It holds that $h\big(L(\mathcal{A})\big) = L(\mathcal{A}')$.

Closure of Regular Languages under Linear Homomorphisms

This is not true in general for the non-linear homomorphisms.

Closure of Regular Languages under Linear Homomorphisms

This is not true in general for the non-linear homomorphisms.

Example : Non-linear homomorphisms

$\Sigma = \{a : 0, g : 1, f : 1\}$, $\Sigma' = \{a : 0, g : 1, f' : 2\}$,
$h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$ with $t_a = a$, $t_g = g(x_1)$, $t_f = f'(x_1, x_1)$.
Let $L = \left\{ f\big(g^n(a)\big) \;\middle|\; n \geq 0 \right\}$,
$h(L) = \left\{ f'\big(g^n(a), g^n(a)\big) \;\middle|\; n \geq 0 \right\}$ is not regular.

Closure of Regular Languages under Inverse Homomorphisms

Theorem :

For all regular languages L and all homomorphisms h, $h^{-1}(L)$ is regular.

$\mathcal{A}' = (Q', Q'_{\mathsf{f}}, \Delta')$ complete deterministic such that $L(\mathcal{A}') = L$.
We construct $\mathcal{A} = (Q, Q_{\mathsf{f}}, \Delta)$ with $Q = Q' \uplus \{q_\forall\}$ $Q_f = Q'_{\mathsf{f}}$ and Δ is defined by:

- for $a \in \Sigma_0$, if $t_a \xrightarrow[\mathcal{A}']{*} q$ then $a \to q \in \Delta$;
- for all $f \in \Sigma_n$ with $n > 0$, for $p_1, \ldots, p_n \in Q$, if $t_f\{x_1 \mapsto p_1, \ldots, x_n \mapsto p_n\} \xrightarrow[\mathcal{A}']{*} q$ then $f(q_1, \ldots, q_n) \to q \in \Delta$ where $q_i = p_i$ if x_i occurs in t_f and $q_i = q_\forall$ otherwise;
- for $a \in \Sigma_0$, $a \to q_\forall \in \Delta$;
- for $f \in \Sigma_n$ where $n > 0$, $f(q_\forall, \ldots, q_\forall) \to q_\forall \in \Delta$.

It holds that $t \xrightarrow[\mathcal{A}]{*} q$ iff $h(t) \xrightarrow[\mathcal{A}']{*} q$ for all $q \in Q'$.

Closure under Homomorphisms

Theorem :

The class of regular tree languages is the smallest non trivial class of sets of trees closed under linear homomorphisms and inverse homomorphisms.

A problem whose decidability has been open for 35 years:

INPUT:	a TA $\mathcal{A}$, an homomorphism h
QUESTION:	is $h(L(\mathcal{A}))$ regular?

Tree Transducers

Tree Transducers

Definition : Bottom-up Tree Transducers

A *bottom-up tree transducer* (TT) is a tuple $U = (\Sigma, \Sigma', Q, Q^{\mathsf{f}}, \Delta)$ where

- Σ, Σ' are the input, resp. output, signatures,
- Q is a finite set of *states*,
- $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states
- Δ is a set of transduction (rewrite) rules of the form:
  - $f(p_1(x_1), \ldots, p_n(x_n)) \to p(u)$ with $f \in \Sigma_n$ $(n \geq 0)$, $p_1, \ldots, p_n, p \in Q$, $x_1, \ldots, x_n$ pairwise distinct and $u \in \mathcal{T}(\Sigma', \{x_1, \ldots, x_n\})$, or
  - $p(x_1) \to p'(u)$ with $q, q' \in Q$, $u \in \mathcal{T}(\Sigma', \{x_1\})$.

A TT is *linear* if all the u in transduction rules are linear.

The transduction relation of U is the binary relation:

$$L(U) = \left\{ \langle t, t' \rangle \;\middle|\; t \xrightarrow[U]{*} q(t'), t \in \mathcal{T}(\Sigma), t' \in \mathcal{T}(\Sigma'), q \in Q^{\mathsf{f}} \right\}$$

Example 1

$$U_1 = \big(\{f:1, a:0\}, \{g:2, f, f':1, a:0\}, \{q, q'\}, \{q'\}, \Delta_1\big),$$

$$\Delta_1 = \left\{ \begin{array}{rcl} a & \to & q(a) \\ f(q(x_1)) & \to & q(f(x_1)) \mid q(f'(x_1)) \mid q'(g(x_1, x_1)) \end{array} \right\}$$

Example 2

$$\Sigma_{in} = \{f:2, g:1, a:0\},$$
$$U_2 = \big(\Sigma_{in}, \Sigma_{in} \cup \{f':1\}, \{q, q', q_{\mathsf{f}}\}, \{q_{\mathsf{f}}\}, \Delta_2\big),$$

$$\Delta_2 = \left\{ \begin{array}{rcl} a & \to & q(a) \bigm| q'(a) \\ g(q(x_1)) & \to & q(g(x_1)) \\ g(q'(x_1)) & \to & q'(g(x_1)) \\ f(q'(x_1), q'(x_2)) & \to & q'(f(x_1, x_2)) \\ f(q'(x_1), q'(x_2)) & \to & q_{\mathsf{f}}(f'(x_1)) \end{array} \right\}$$

$$L(U_2) = \big\{ \langle f(t_1, t_2), f'(t_1) \bigm| t_2 = g^m(a), m \geq 0 \big\}$$

Tree Transducers, Example

Token tree protocol [Abdulla et al CAV02]

$$\begin{array}{rcl}
\underline{\mathsf{n}} & \to & q_0(\underline{\mathsf{n}'}) \\
\underline{\mathsf{t}} & \to & q_1(\underline{\mathsf{n}'}) \\
\mathsf{n}\big(q_0(x_1), q_0(x_2)\big) & \to & q_0\big(\mathsf{n}(x_1, x_2)\big) \\
\mathsf{t}\big(q_0(x_1), q_0(x_2)\big) & \to & q_1\big(\mathsf{n}(x_1, x_2)\big) \\
\mathsf{n}\big(q_1(x_1), q_0(x_2)\big) & \to & q_2\big(\mathsf{t}(x_1, x_2)\big) \\
\mathsf{n}\big(q_0(x_1), q_1(x_2)\big) & \to & q_2\big(\mathsf{t}(x_1, x_2)\big) \\
\mathsf{n}\big(q_2(x_1), q_0(x_2)\big) & \to & q_2\big(\mathsf{n}(x_1, x_2)\big) \\
\mathsf{n}\big(q_0(x_1), q_2(x_2)\big) & \to & q_2\big(\mathsf{n}(x_1, x_2)\big)
\end{array}$$

property: mutual exclusion (for every network)
initial: terms of $\mathcal{T}\big(\{\mathsf{t}, \mathsf{n}, \underline{\mathsf{t}}, \underline{\mathsf{n}}\}\big)$, containing exactly one token.
verification: the intersection of his closure with the set
$\{q_2(t) \mid t \in \mathcal{T}\big(\{\mathsf{t}, \mathsf{n}, \underline{\mathsf{t}}, \underline{\mathsf{n}}\}\big),\ t$ contains at least 2 tokens$\}$ (regular) is empty.

Languages

- Linear bottom-up TT are closed under composition.
- Deterministic bottom-up TT are closed under composition.

Theorem :

- The domain of a TT is a regular tree language.
- The image of a regular tree language by a linear TT is a regular tree language.

Transducers and Homomorphisms

An homomorphism is called *delabeling* if it is linear, complete, symbol-to-symbol.

Definition : Bimorphisms

A *bimorphism* is a triple $B = (h, h', L)$ where h, h' are homomorphisms and L is a regular tree language.

$$L(B) = \left\{ \langle h(t), h'(t) \rangle \;\middle|\; t \in L \right\}$$

Theorem :

TT $\equiv$ bimorphisms (h, h', L) where h delabeling.

Term Rewriting Systems

Definition : Substitution

A *substitution* is a function of finite domain from $\mathcal{X}$ into $\mathcal{T}(\Sigma, \mathcal{X})$.
We extend the definition to $\mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\Sigma, \mathcal{X})$ by:

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma) \quad (n \geq 0)$$

The application $C[t_1, \ldots, t_n]$ of a context $C \in \mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$ to n terms $t_1, \ldots, t_n$, is $C\sigma$ with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Term Rewriting

A *rewrite system* $\mathcal{R}$ is a finite set of rewrite rules of the form $\ell \to r$ with $\ell, r \in \mathcal{T}(\Sigma, \mathcal{X})$.

The relation $\xrightarrow[\mathcal{R}]{}$ is the smallest binary relation containing $\mathcal{R}$, and closed under application of contexts and substitutions.
i.e. $s \xrightarrow[\mathcal{R}]{} t$ iff $\exists p \in \mathcal{P}os(s), \ell \to r \in \mathcal{R}, \sigma,\ s|_p = \ell\sigma$ and $t = s[r\sigma]_p$.

We note $\xrightarrow[\mathcal{R}]{*}$ the reflexive and transitive closure of $\xrightarrow[\mathcal{R}]{}$.

Example :

$\mathcal{R} = \{+(0, x) \to x, +(s(x), y) \to s(+(x, y))\}$.

$$\begin{array}{rcl} +\big(s(s(0)), +(0, s(0))\big) & \xrightarrow[\mathcal{R}]{} & +\big(s(s(0)), s(0)\big) \\ & \xrightarrow[\mathcal{R}]{} & s\big(+(s(0), s(0))\big) \\ & \xrightarrow[\mathcal{R}]{} & s\big(s\big(+(0, s(0))\big)\big) \\ & \xrightarrow[\mathcal{R}]{} & s(s(s(0))) \end{array}$$

TRS Preserving Regularity

For a TRS $\mathcal{R}$ over Σ and $L \subseteq \mathcal{T}(\Sigma)$,

$$\mathcal{R}^*(L) = \{t \in \mathcal{T}(\Sigma) \mid \exists s \in L, s \xrightarrow[\mathcal{R}]{*} t\}$$

Regularity Preservation

Identify a class $\mathcal{C}$ of TRS such that
for all $\mathcal{R} \in \mathcal{C}$, $\mathcal{R}^*(L)$ is regular if L is regular.

Theorem : [Gilleron STACS 91]

It is undecidable in general whether a given TRS is preserving regularity.

Ground TRS

Theorem : [Brainerd 69]

Ground TRS are preserving regularity.

Given: TA $\mathcal{A}_{\mathsf{in}}$ and ground TRS $\mathcal{R}$. We start with

$$\mathcal{A}_{\mathsf{in}} \cup (\Sigma, Q_{\mathcal{R}}, \emptyset, \{f(q_{r_1}, \ldots, q_{r_n}) \to q_r \mid r = f(r_1, \ldots r_n) \in Q_{\mathcal{R}}\})$$

where $Q_{\mathcal{R}} = \mathit{strict\ subterms}(\mathit{rhs}(\mathcal{R}))$,
and add transitions according to the schema:

$$\begin{tikzcd}
lhs(\mathcal{R}) \ni \ell \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\
f(r_1, \ldots, r_n) \arrow[r, "\mathcal{A}"'] & f(q_{r_1}, \ldots, q_{r_n}) \arrow[u, dashed, "\mathcal{A}"']
\end{tikzcd}$$

no states are added $\to$ termination.
The TA obtained recognizes $\mathcal{R}^*\big(L(\mathcal{A}_{\mathsf{in}})\big)$.

Ground TRS (examples)

$$
\begin{tikzcd}
lhs(\mathcal{R}) \ni \ell \arrow[rr, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & & q \\
f(r_1, \ldots, r_n) \arrow[r, "\mathcal{A}"'] & f(q_{r_1}, \ldots, q_{r_n}) \arrow[ur, dashed, "\mathcal{A}"'] &
\end{tikzcd}
$$

$s(s(0)) \to 0$	$\perp + 1 \to s(\perp)$

$$
\begin{tikzcd}
s(s(0)) \arrow[r, "*", "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\
0 \arrow[ur, dashed, "\mathcal{A}"] &
\end{tikzcd}
$$

$$
\begin{tikzcd}
\perp + 1 \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\
s(\perp) \arrow[r, "\mathcal{A}"'] & s(q_\perp) \arrow[u, dashed, "\mathcal{A}"']
\end{tikzcd}
$$

Linear and Right-Shallow TRS

right-shallow: variables at depth at most 1 in rhs of rules.

Theorem : [Salomaa 88]

Linear and right-shallow TRS preserve regularity.

Given: TA $\mathcal{A}_{\mathsf{in}}$ and linear and right-shallow TRS $\mathcal{R}$.
The construction is similar to the ground TRS case: We start with

$$\mathcal{A}_{\mathsf{in}} \cup (\Sigma, Q_{\mathcal{R}}, \emptyset, \{f(q_{r_1}, \ldots, q_{r_n}) \to q_r \mid r = f(r_1, \ldots r_n) \in Q_{\mathcal{R}}\})$$

where $Q_{\mathcal{R}} = \mathit{strict\ subterms}(\mathit{rhs}(\mathcal{R})) \setminus \mathcal{X}$,
and add transitions according to the schema:

$$\begin{tikzcd}
\ell\sigma \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\
f(r_1, \ldots, r_n)\sigma \arrow[r, "\mathcal{A}"'] & f(q_1, \ldots, q_n) \arrow[u, dashed, "\mathcal{A}"']
\end{tikzcd}$$

where $\ell \in \mathit{lhs}(\mathcal{R})$, substitution $\sigma : \mathit{vars}(\ell) \to Q$, for all $i \leq n$, if $r_i \notin \mathcal{X}$ then $q_i = q_{r_i}$ and $q_i = r_i\sigma$ otherwise.

Linear and Right-Shallow TRS (Examples)

$$\begin{tikzcd} \ell\sigma \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\ f(r_1, \ldots, r_n)\sigma \arrow[r, "\mathcal{A}"'] & f(q_1, \ldots, q_n) \arrow[u, dashed, "\mathcal{A}"'] \end{tikzcd}$$

where $\ell \in lhs(\mathcal{R})$, substitution $\sigma : vars(\ell) \to Q$, for all $i \leq n$, if $r_i \notin \mathcal{X}$ then $q_i = q_{r_i}$ and $q_i = r_i\sigma$ otherwise.

$s(x) - s(y) \to x - y$	$s(x) \to s(0) + x$
$\begin{tikzcd} s(q_1) - s(q_2) \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q_1' - q_2' \arrow[r] & q \\ q_1 - q_2 \arrow[urr, dashed, "\mathcal{A}"'] & & \end{tikzcd}$	$\begin{tikzcd} s(q_1) \arrow[r, "\mathcal{A}"'] \arrow[d, "\mathcal{R}"] & q \\ s(0) + q_1 \arrow[r, "\mathcal{A}"'] & q_{s(0)} + q_1 \arrow[u, dashed, "\mathcal{A}"'] \end{tikzcd}$

Linear and Right-Shallow TRS: Extensions

Other classes of TRS preserving regularity

- [Coquide et al 94] *semi-monadic* or *inverse-growing* TRS: for all $\ell \to r \in \mathcal{R}$, $vars(r) \cap vars(\ell)$ at depth at most 1 in r.
- [Nagaya Toyama RTA 02] right-linear and right-shallow TRS. NOT left-linear.
- [Gyenizse Vagvolgyi GSMTRS 98] linear and *generalized semi-monadic* TRS
- [Takai Kaji Seki RTA 00] right-linear *finite path overlapping* TRS

Right-Linearity and Right-Shallowness Conditions

Relaxing these conditions generaly breaks regularity preservation.

Example : Right-Linearity

let $\mathcal{R} = \{f(x) \to g(x, x)\}$ (flat and left-linear), $L_{\mathsf{in}} = \{f(\ldots f(c))\}$. $\mathcal{R}^*(L_{\mathsf{in}}) \cap \mathcal{T}\big(\{g, c\}\big)$ is the set of balanced binary trees of $\mathcal{T}\big(\{g, c\}\big)$, which is not regular.

Example : Right-Shallowness

With rewrite rules whose left and right hand-side have height at most two, it is possible simulate Turing machine computations, even in the case of words (symbols of arity 0 or 1).

Exceptions (for the right-shallowness)

- [Rety LPAR 99] constructor based (with restrictions on L_{in}).
  ex: $\mathsf{app}(\mathsf{nil}, y) \to y$, $\mathsf{app}\big(\mathsf{cons}(x, y), z\big) \to \mathsf{cons}\big(x, \mathsf{app}(y, z)\big)$.
- [Seki et al RTA 02] Layered Transducing TRS

Linear I/O Separated Layered Transducing TRS

[Seki et al RTA 02]
This class corresponds to linear tree transducers.

over $\Sigma = \Sigma_i \uplus \Sigma_o \uplus Q$, rewrite rules of the form

$$\begin{array}{rcl} f_i(p_1(x_1), ..., p_n(x_n)) & \rightarrow & p(t) \\ p'_1(x_1) & \rightarrow & p'(t') \end{array}$$

where $f_i \in \Sigma_i$, $p_1, \ldots, p_n, p, p'_1, p' \in Q$ $x_1, \ldots, x_n$ are disjoint variables, $t, t' \in \mathcal{T}(\Sigma_o, \mathcal{X})$ such that $vars(t) \subseteq \{x_1, \ldots, x_n\}$ and $vars(t') \subseteq \{x_1\}$.

To know more

Further results closure of tree automata languages:

- closure of extended tree automata languages, modulo [Gallagher Rosendahl 08], [JRV JLAP 08], [JKV LATA 09], [JKV IC 11]
- rewrite strategies (bottom-up, context-sensitive, innermost, outermost...) [Durand et al RTA 07,10,11], [Kojima Sakai RTA 08], [Rety Vuotto JSC 05], [GGJ WRS 08]
- constrained/controlled rewriting [Sénizergues *French Spring School of TCS* 93], [JKS FroCoS 11]
- unranked tree rewriting (XML updates) [JR RTA 08], [JR PPDP 10]

Tree Automata Based Program Verification
Some Techniques and Tools

Program Analysis with Tree Automata / Grammars

(very partial list) focus on 3 approaches

- [Reynolds IP 68] LISP programs $\rightarrow$ lfp solutions of equations
- [Jones Muchnick POPL 79] LISP programs $\rightarrow$ tree grammars
- [Jones 87] lazy higher-order functional programs
- [Heintze Jaffar 90] logic programs $\rightarrow$ set constraints
- [Lugiez Schnoebelen CONCUR 98], [Bouajjani Touili 03+] imperative programs w. prefix rewriting: PA-processes, PAD systems, PRS...
- [Genet et al 98+]
  functional programs, security protocols, Java Bytecode
- [Jones Andersen TCS 07] functional programs

Timbuk

[Genet et al] (IRISA)
`http://www.irisa.fr/celtique/genet/timbuk`

Computation of rewrite closure by tree automata completion, with over-approximations. User defined or infered accelerations.

- analysis of security protocols
  SmartRight, Copy Protection Technology for DVB, Thomson
- analysis of Java Bytecode with Copster

Timbuk library, used in other tools like

- TA4SP, one of the proof back-ends of the AVISPA tool for security protocol verification
- SPADE

SPADE ♠

[Tayssir Touili et al CAV 07] (LIAFA).
`http://www.liafa.jussieu.fr/~touili/spade.html`

Reachability analysis for multithreaded dynamic and recursive programs.

- (PAD) Systems [Touili VISSAS 05]

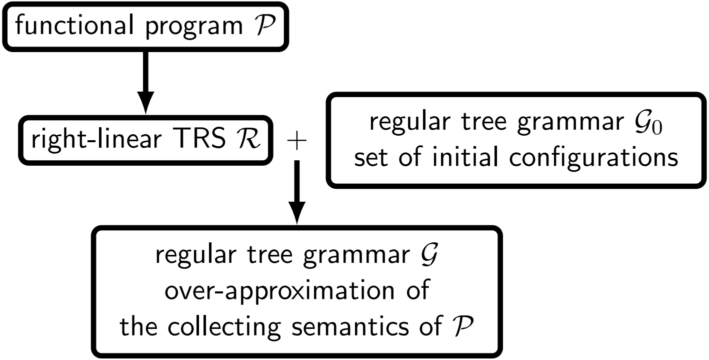
$$X_1 \cdot \ldots \cdot X_n \rightarrow Y_1 \cdot \ldots \cdot Y_m, \quad X_1 \rightarrow Y_1 \parallel \ldots \parallel Y_m$$

Case studies

- Windows Bluetooth driver
- multithreaded program based on the class `java.util.Vector` from the Java Standard Collection Framework
- concurrent insertions on a binary search tree

Approximations of Collecting Semantics

[Jones Andersen TCS 07]



collecting semantics [Cousot^2] (roughly): mapping associating to each program point p the set of configurations reachable at p.

[Kochems Ong RTA 11] finer approximation using indexed linear tree grammars (instead of regular grammars).

Regular Tree Grammars

Definition : Regular Tree Grammars

A is a tuple $\mathcal{G} = \langle \mathcal{N}, S, \Sigma, P \rangle$ where $\mathcal{N}$ is a finite set of nullary *non-terminal* symbols, $S \in \mathcal{N}$ (*axiom* of $\mathcal{G}$), Σ is a signature disjoint from $\mathcal{N}$ and P is a set of *production rules* of the form $X := r$ with $r \in \mathcal{T}(\Sigma \cup \mathcal{N})$.

Example :

$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$, $\mathcal{G} = (\{X_0, X_1\}, X_1, \Sigma, P)$.

$$P = \left\{ \begin{array}{lllllll} X_0 & := & \bot & & X_1 & := & \top \\ X_1 & := & \neg(X_0) & & X_0 & := & \neg(X_1) \\ X_0 & := & \vee(X_0, X_0) & & X_1 & := & \vee(X_0, X_1) \\ X_1 & := & \vee(X_1, X_0) & & X_1 & := & \vee(X_1, X_1) \\ X_0 & := & \wedge(X_0, X_0) & & X_0 & := & \wedge(X_0, X_1) \\ X_0 & := & \wedge(X_1, X_0) & & X_1 & := & \wedge(X_1, X_1) \end{array} \right\}$$

Approximations of Collecting Semantics: Example

Concurrent readers/writers: reachable configurations

$$
\mathcal{R} = \begin{array}{lrcl}
R_1: & \mathsf{state}(0,0) & \rightarrow & \mathsf{state}(0,s(0)) \\
R_2: & \mathsf{state}(X_2,0) & \rightarrow & \mathsf{state}(s(X_2),0) \\
R_3: & \mathsf{state}(X_3,s(Y_3)) & \rightarrow & \mathsf{state}(X_3,Y_3) \\
R_4: & \mathsf{state}(s(X_4),Y_4) & \rightarrow & \mathsf{state}(X_4,Y_4)
\end{array}
$$

$$
\begin{tikzcd}
\mathsf{state}(0,0) \arrow[r, bend left, "1"] \arrow[d, bend right, "2"'] & \mathsf{state}\big(0,s(0)\big) \arrow[l, bend left, "3"] \\
\mathsf{state}\big(s(0),0\big) \arrow[u, bend right, "4"'] \arrow[d, bend right, "2"'] & \\
\mathsf{state}\big(s(s(0)),0\big) \arrow[u, bend right, "4"'] & \\
\vdots &
\end{tikzcd}
$$

Approximations of Collecting Semantics: Example

$$\mathcal{R} = \begin{array}{lrcl} R_1 : & \mathsf{state}(0,0) & \rightarrow & \mathsf{state}(0,s(0)) \\ R_2 : & \mathsf{state}(X_2,0) & \rightarrow & \mathsf{state}(s(X_2),0) \\ R_3 : & \mathsf{state}(X_3,s(Y_3)) & \rightarrow & \mathsf{state}(X_3,Y_3) \\ R_4 : & \mathsf{state}(s(X_4),Y_4) & \rightarrow & \mathsf{state}(X_4,Y_4) \end{array}$$

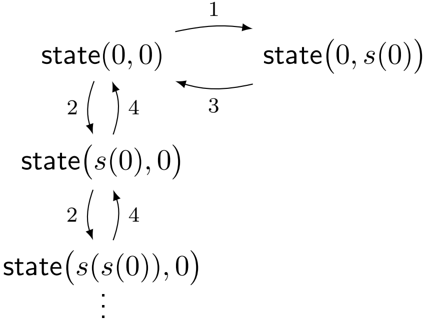
$R_0 := \mathsf{state}(0,0)$

$R_0 := R_1$ $R_1 := \mathsf{state}(0,s(0))$	$\mathsf{state}(0,0) = lhs(R_1)$
$R_0 := R_2$ $R_2 := \mathsf{state}(s(X_2),0)$ $X_2 := 0$	$\mathsf{state}(0,0) = \mathsf{state}(X_2,0)\{X_2 \mapsto 0\}$
$X_2 := s(X_2)$	$\mathsf{state}(s(X_2),0) =$ $\mathsf{state}(X_2,0)\{X_2 \mapsto s(X_2)\}$
$R_1 := R_3$ $R_3 := \mathsf{state}(X_3,Y_3)$ $X_3 := 0,\ Y_3 := 0$	$\mathsf{state}(0,s(0)) =$ $\mathsf{state}(X_3,s(Y_3))\{X_3 \mapsto 0, Y_3 \mapsto 0\}$
$R_2 := R_4$ $R_4 := \mathsf{state}(s(X_4),Y_4)$ $X_4 := X_2,\ Y_4 := 0$	$\mathsf{state}(s(X_2),0)) =$ $\mathsf{state}(s(X_4),Y_4)\{X_4 \mapsto X_2, Y_4 \mapsto 0\}$

Approximations of Collecting Semantics: Example

$$
\mathcal{R} = \begin{array}{lrcl}
R_1: & \mathsf{state}(0,0) & \to & \mathsf{state}(0,s(0)) \\
R_2: & \mathsf{state}(X_2,0) & \to & \mathsf{state}(s(X_2),0) \\
R_3: & \mathsf{state}(X_3,s(Y_3)) & \to & \mathsf{state}(X_3,Y_3) \\
R_4: & \mathsf{state}(s(X_4),Y_4) & \to & \mathsf{state}(X_4,Y_4)
\end{array}
$$

$$
\begin{array}{lcl}
R_0 & := & \mathsf{state}(0,0) \\
\hline
R_0 & := & R_1 \\
R_1 & := & \mathsf{state}(0,s(0)) \\
\hline
R_0 & := & R_2 \\
R_2 & := & \mathsf{state}(s(X_2),0) \\
X_2 & := & 0 \\
\hline
X_2 & := & s(X_2) \\
\hline
R_1 & := & R_3 \\
R_3 & := & \mathsf{state}(X_3,Y_3) \\
\multicolumn{3}{l}{X_3 := 0,\ \ Y_3 := 0} \\
\hline
R_2 & := & R_4 \\
R_4 & := & \mathsf{state}(s(X_4),Y_4) \\
\multicolumn{3}{l}{X_4 := X_2,\ \ Y_4 := 0} \\
\hline
\end{array}
$$



Approximations of Collecting Semantics: Example 2

[Jones Andersen TCS 07]

```
let rec first l1 l2 =
  match l1, l2 with
    [], _ → []
    l::m, x::xs → x::(first m xs);
```

$$
\begin{array}{lrcl}
R_2 : & \mathsf{first}(\mathsf{nil}, X_s) & \rightarrow & \mathsf{nil} \\
R_3 : & \mathsf{first}(\mathsf{cons}(1, M), \mathsf{cons}(X, X_s)) & \rightarrow & \mathsf{cons}(X, \mathsf{first}(M, X_s))
\end{array}
$$

```
let rec sequence y =
  y::(sequence (1::y));
```

$$
R_4 : \quad \mathsf{sequence}(Y) \quad \rightarrow \quad \mathsf{cons}(Y, \mathsf{sequence}(\mathsf{cons}(1, Y)))
$$

```
let g n =
  first n (sequence []);
```

$$
R_1 : \quad \mathsf{g}(N) \quad \rightarrow \quad \mathsf{first}(N, \mathsf{sequence}(\mathsf{nil}))
$$

Part II

Weak Second Order Monadic Logic with k Successors

Logic and Automata

- logic for expressing properties of labeled binary trees
  = specification of tree languages,

Logic and Automata

- logic for expressing properties of labeled binary trees
  = specification of tree languages, example:

$$t \models \forall x\ a(x) \Rightarrow \exists y\ y > x \wedge b(y)$$

- compilation of formulae into automata
  = decision algorithms.
- equivalence between both formalisms
  [Thatcher & Wright's theorem].

Plan

WSkS: Definition

Automata $\rightarrow$ Logic

Logic $\rightarrow$ Automata

Fragments and Extensions of WSkS

Interpretation Structures

$\mathcal{L} :=$ set of predicate symbols $P_1, \ldots P_n$ with arity.

A structure $\mathcal{M}$ over $\mathcal{L}$ is a tuple

$$\mathcal{M} := \left\langle \mathcal{D}, P_1^{\mathcal{M}}, \ldots, P_n^{\mathcal{M}} \right\rangle$$

where

- $\mathcal{D}$ is the domain of $\mathcal{M}$,
- every $P_i^{\mathcal{M}}$ (interpretation of P_i) is a subset of $\mathcal{D}^{arity(P_i)}$ (relation).

Term as Structure

Σ signature, $k =$ maximal arity.

$$\mathcal{L}_\Sigma := \{=, <, S_1, \ldots, S_k, L_a \bigm| a \in \Sigma\}.$$

to $t \in \mathcal{T}(\Sigma)$, we associate a structure $\underline{t}$ over $\mathcal{L}_\Sigma$

$$\underline{t} := \left\langle \mathcal{P}os(t), =, <, S_1, \ldots, S_k, L_a^{\underline{t}}, L_b^{\underline{t}}, \cdots \right\rangle$$

where

- domain = positions of t $\quad (\mathcal{P}os(t) \subset \{1, \ldots, k\}^*)$
- $=$ equality over $\mathcal{P}os(t)$,
- $<$ prefix ordering over $\mathcal{P}os(t)$,
- $S_i = \big\{\langle p, p \cdot i\rangle \mid p, p \cdot i \in \mathcal{P}os(t)\big\}$ (i^{th} successor position),
- $L_a^{\underline{t}} = \{p \in \mathcal{P}os(t) \mid t(p) = a\}$.

FOL with k Successors

- first order variables $x, y \ldots$
- $$\text{form} \quad ::= \quad x = y \mid x < y \\ \mid S_1(x,y) \mid \ldots \mid S_k(x,y) \mid L_a(x) \quad a \in \Sigma \\ \mid \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \neg\text{form} \\ \mid \exists x\, \text{form} \mid \forall x\, \text{form}$$

Notation: $\phi(x_1, \ldots, x_m)$,
where $x_1, \ldots, x_m$ are the free variables of ϕ.

WSkS: Syntax

- first order variables $x, y \dots$
- second order variables $X, Y \dots$
- $$\begin{array}{rcl} \text{form} & ::= & x = y \mid x < y \mid x \in X \\ & & \mid S_1(x,y) \mid \dots \mid S_k(x,y) \mid L_a(x) \quad a \in \Sigma \\ & & \mid \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \neg\text{form} \\ & & \mid \exists x \,\text{form} \mid \exists X \,\text{form} \mid \forall x \,\text{form} \mid \forall X \,\text{form} \end{array}$$

Notation: $\phi(x_1, \dots, x_m, X_1, \dots, X_n)$,
where $x_1, \dots, x_m$, $X_1, \dots, X_n$ are the free variables of ϕ.

WSkS: Semantics

- $t \in \mathcal{T}(\Sigma)$,
- valuation σ of first order variables into $\mathcal{P}os(t)$,
- valuation δ of second order variables into subsets of $\mathcal{P}os(t)$,
- $\underline{t}, \sigma, \delta \models x = y$ iff $\sigma(x) = \sigma(y)$,
- $\underline{t}, \sigma, \delta \models x < y$ iff $\sigma(x) <_{\mathit{prefix}} \sigma(y)$,
- $\underline{t}, \sigma, \delta \models x \in X$ iff $\sigma(x) \in \delta(X)$,
- $\underline{t}, \sigma, \delta \models S_i(x, y)$ iff $\sigma(y) = \sigma(x) \cdot i$,
- $\underline{t}, \sigma, \delta \models L_a(x)$ iff $t(\sigma(x)) = a$ i.e. $\sigma(x) \in L_a^t$,
- $\underline{t}, \sigma, \delta \models \phi_1 \wedge \phi_2$ iff $\underline{t}, \sigma, \delta \models \phi_1$ and $\underline{t}, \sigma, \delta \models \phi_2$,
- $\underline{t}, \sigma, \delta \models \phi_1 \vee \phi_2$ iff $\underline{t}, \sigma, \delta \models \phi_1$ or $\underline{t}, \sigma, \delta \models \phi_2$,
- $\underline{t}, \sigma, \delta \models \neg\phi$ iff $\underline{t}, \sigma, \delta \not\models \phi$,

WSkS: Semantics (Quantifiers)

- $\underline{t}, \sigma, \delta \models \exists x\ \phi$ iff $x \notin dom(\sigma)$, x free in ϕ and exists $p \in \mathcal{P}os(t)$ s.t. $\underline{t}, \sigma \cup \{x \mapsto p\}, \delta \models \phi$,
- $\underline{t}, \sigma, \delta \models \forall x\ \phi$ iff $x \notin dom(\sigma)$, x free in ϕ and for all $p \in \mathcal{P}os(t)$, $\underline{t}, \sigma \cup \{x \mapsto p\}, \delta \models \phi$,
- $\underline{t}, \sigma, \delta \models \exists X\ \phi$ iff $X \notin dom(\delta)$, X free in ϕ and exists $P \subseteq \mathcal{P}os(t)$ s.t. $\underline{t}, \sigma, \delta \cup \{X \mapsto P\} \models \phi$,
- $\underline{t}, \sigma, \delta \models \forall X\ \phi$ iff $X \notin dom(\delta)$, X free in ϕ and for all $P \subseteq \mathcal{P}os(t)$, $\underline{t}, \sigma, \delta \cup \{X \mapsto P\} \models \phi$.

WSkS: Languages

Definition : WSkS-definability

For $\phi \in$ WSkS closed (without free variables) over $\mathcal{L}_\Sigma$,

$$L(\phi) := \{t \in \mathcal{T}(\Sigma) \mid \underline{t} \models \phi\}.$$

Example :

$\Sigma = \{a : 2, b : 2, c : 0\}$. Language of terms in $\mathcal{T}(\Sigma)$

- containing the pattern $a(b(x_1, x_2), x_3)$:
  $\exists x \exists y\ S_1(x, y) \wedge L_a(x) \wedge L_b(y)$
- such that every a-labelled node has a b-labelled child.
  $\forall x \exists y\ L_a(x) \Rightarrow \bigvee_{i=1}^{2} S_i(x, y) \wedge L_b(y)$
- such that every a-labelled node has a b-labelled descendant.
  $\forall x \exists y\ L_a(x) \Rightarrow x < y \wedge L_b(y)$

WSkS: Examples

- root position:

WSkS: Examples

- root position: root$(x) \equiv \neg\exists y\ y < x$
- inclusion:

WSkS: Examples

- root position: root$(x) \equiv \neg \exists y \; y < x$
- inclusion: $X \subseteq Y \equiv \forall x (x \in X \Rightarrow x \in Y)$
- intersection:

WSkS: Examples

- root position: root$(x) \equiv \neg \exists y \; y < x$
- inclusion: $X \subseteq Y \equiv \forall x (x \in X \Rightarrow x \in Y)$
- intersection: $Z = X \cap Y \equiv \forall x \; (x \in Z \Leftrightarrow (x \in X \wedge x \in Y))$
- emptiness:

WSkS: Examples

- root position: root$(x) \equiv \neg\exists y \; y < x$
- inclusion: $X \subseteq Y \equiv \forall x (x \in X \Rightarrow x \in Y)$
- intersection: $Z = X \cap Y \equiv \forall x \; (x \in Z \Leftrightarrow (x \in X \wedge x \in Y))$
- emptiness: $X = \emptyset \equiv \forall x \; x \notin X$
- finite union:

WSkS: Examples

- root position: $\mathrm{root}(x) \equiv \neg\exists y\; y < x$
- inclusion: $X \subseteq Y \equiv \forall x (x \in X \Rightarrow x \in Y)$
- intersection: $Z = X \cap Y \equiv \forall x\; (x \in Z \Leftrightarrow (x \in X \wedge x \in Y))$
- emptiness: $X = \emptyset \equiv \forall x\; x \notin X$
- finite union:
  $$X = \bigcup_{i=1}^{n} X_i \equiv \big(\bigwedge_{i=1}^{n} X_i \subseteq X\big) \wedge \forall x\; \big(x \in X \Rightarrow \bigvee_{i=1}^{n} x \in X_i\big)$$
- partition:

WSkS: Examples

- root position: $\mathrm{root}(x) \equiv \neg \exists y \; y < x$
- inclusion: $X \subseteq Y \equiv \forall x (x \in X \Rightarrow x \in Y)$
- intersection: $Z = X \cap Y \equiv \forall x \; (x \in Z \Leftrightarrow (x \in X \wedge x \in Y))$
- emptiness: $X = \emptyset \equiv \forall x \; x \notin X$
- finite union:
  $$X = \bigcup_{i=1}^{n} X_i \equiv \big(\bigwedge_{i=1}^{n} X_i \subseteq X\big) \wedge \forall x \; \big(x \in X \Rightarrow \bigvee_{i=1}^{n} x \in X_i\big)$$
- partition:
  $$X_1, \ldots, X_n \text{ partition } X \equiv X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset$$

WSkS: Examples (2)

- singleton:

WSkS: Examples (2)

- singleton:
  $\mathsf{sing}(X) \equiv X \neq \emptyset \land \forall Y \left(Y \subseteq X \Rightarrow (Y = X \lor Y = \emptyset) \right)$
- $\leq$ (without $<$)

WSkS: Examples (2)

- singleton:
  sing$(X) \equiv X \neq \emptyset \land \forall Y \; \big(Y \subseteq X \Rightarrow (Y = X \lor Y = \emptyset)\big)$
- $\leq$ (without $<$)

$$x \leq y \;\; \equiv \;\; \forall X \left(\begin{array}{l} y \in X \\ \land \, \forall z \, \forall z' \, (z' \in X \land \bigvee_{i \leq k} S_i(z, z')) \Rightarrow z \in X \end{array} \right) \\ \Rightarrow x \in X$$

or

$$x \leq y \;\; \equiv \;\; \exists X \; \big(\forall z \; z \in X \Rightarrow (\exists z' \bigvee_{i \leq k} S_i(z', z) \land z' \in X) \lor z = x\big) \\ \land \, y \in X$$

Thatcher & Wright's Theorem

Theorem : Thatcher and Wright

Languages of WSkS formulae $=$ regular tree languages.

pr.: 2 directions (2 constructions):

- TA $\rightarrow$ WSkS,
- WSkS $\rightarrow$ TA.

Plan

WSkS: Definition

Automata $\rightarrow$ Logic

Logic $\rightarrow$ Automata

Fragments and Extensions of WSkS

Regular Languages $\to$ WSkS Languages

Let $\Sigma = \{a_1, \ldots, a_n\}$.

Theorem :

For all tree automaton $\mathcal{A}$ over Σ, there exists $\phi_{\mathcal{A}} \in$ WSkS such that $L(\phi_A) = L(\mathcal{A})$.

$\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ with $Q = \{q_0, \ldots, q_m\}$.
$\phi_{\mathcal{A}}$: existence of an accepting run of $\mathcal{A}$ on $t \in \mathcal{T}(\Sigma)$.

$$\phi_{\mathcal{A}} := \exists Y_0 \ldots \exists Y_m \; \phi_{\mathsf{lab}}(\overline{Y}) \wedge \phi_{\mathsf{acc}}(\overline{Y}) \wedge \phi_{\mathsf{tr}_0}(\overline{Y}) \wedge \phi_{\mathsf{tr}}(\overline{Y})$$

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{lab}}(\overline{Y})$: every position is labeled with one state exactely.

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{lab}}(\overline{Y})$: every position is labeled with one state exactely.

$$\phi_{\mathsf{lab}}(\overline{Y}) \equiv \forall x \bigvee_{0\leq i\leq m} x \in Y_i \ \wedge \bigwedge_{\substack{0\leq i,j\leq m \\ i\neq j}} \big(x \in Y_i \Rightarrow \neg x \in Y_j\big)$$

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{lab}}(\overline{Y})$: every position is labeled with one state exactely.

$$\phi_{\mathsf{lab}}(\overline{Y}) \equiv \forall x \bigvee_{0 \le i \le m} x \in Y_i \ \wedge \bigwedge_{\substack{0 \le i,j \le m \\ i \ne j}} \big(x \in Y_i \Rightarrow \neg x \in Y_j\big)$$

$\phi_{\mathsf{acc}}(\overline{Y})$: the root is labeled with a final state

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{lab}}(\overline{Y})$: every position is labeled with one state exactely.

$$\phi_{\mathsf{lab}}(\overline{Y}) \equiv \forall x \bigvee_{0 \le i \le m} x \in Y_i \;\wedge \bigwedge_{\substack{0 \le i,j \le m \\ i \ne j}} \big(x \in Y_i \Rightarrow \neg x \in Y_j\big)$$

$\phi_{\mathsf{acc}}(\overline{Y})$: the root is labeled with a final state

$$\phi_{\mathsf{acc}}(\overline{Y}) \equiv \forall x_0 \; \mathsf{root}(x_0) \Rightarrow \bigvee_{q_i \in Q^{\mathsf{f}}} x_0 \in Y_i$$

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{tr}_0}(\overline{Y})$: transitions for constants symbols

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{tr}_0}(\overline{Y})$: transitions for constants symbols

$$\phi_{\mathsf{tr}_0}(\overline{Y}) \equiv \bigwedge_{a \in \Sigma_0} \left(\forall x \; L_a(x) \Rightarrow \bigvee_{a \rightarrow q_i \in \Delta} x \in Y_i \right)$$

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{tr}_0}(\overline{Y})$: transitions for constants symbols

$$\phi_{\mathsf{tr}_0}(\overline{Y}) \equiv \bigwedge_{a \in \Sigma_0} \left(\forall x \; L_a(x) \Rightarrow \bigvee_{a \to q_i \in \Delta} x \in Y_i \right)$$

$\phi_{\mathsf{tr}}(\overline{Y})$: transitions for non-constant symbols

Regular Languages $\rightarrow$ WSkS Languages

$\phi_{\mathsf{tr}_0}(\overline{Y})$: transitions for constants symbols

$$\phi_{\mathsf{tr}_0}(\overline{Y}) \equiv \bigwedge_{a \in \Sigma_0} \Big(\forall x\; L_a(x) \Rightarrow \bigvee_{a \to q_i \in \Delta} x \in Y_i\Big)$$

$\phi_{\mathsf{tr}}(\overline{Y})$: transitions for non-constant symbols

$$\begin{aligned}\phi_{\mathsf{tr}}(\overline{Y}) \quad \equiv \quad & \bigwedge_{f \in \Sigma_j, 0 < j \leq k} \forall x\, \forall y_1 \ldots \forall y_j \\ & \big(L_f(x) \wedge S_1(x, y_1) \wedge \ldots \wedge S_j(x, y_j)\big) \\ & \quad \Downarrow \\ & \bigvee_{f(q_{i_1}, \ldots, q_{i_j}) \to q_i \in \Delta} x \in Y_i \wedge y_1 \in Y_{i_1} \wedge \ldots \wedge y_j \in Y_{i_j}\end{aligned}$$

Plan

WSkS: Definition

Automata $\rightarrow$ Logic

Logic $\rightarrow$ Automata

Fragments and Extensions of WSkS

Thatcher & Wright's Theorem

Theorem :

Every WSkS language is regular.

For all formula $\phi \in$ WSkS over Σ (without free variables) there exists a tree automaton $\mathcal{A}_\phi$ over Σ, such that $L(\mathcal{A}_\phi) = L(\phi)$.

Corollary :

WSkS is decidable.

pr.: reduction to emptiness decision for $\mathcal{A}_\phi$.

Thatcher & Wright's Theorem

$\mathcal{A}_\phi$ is effectively constructed from ϕ, by induction.

- automata for atoms

  $\Rightarrow$ need of automata for formula with free variables.

  it will characterize
- Boolean closures for Boolean connectors.
- $\exists$ quantifier: projection.

Thatcher & Wright's Theorem

When ϕ contains free variables, $\mathcal{A}_\phi$ will characterize both terms AND valuations satisfying ϕ: $L(\mathcal{A}_\phi) \equiv \{\langle t, \sigma, \delta\rangle \mid \underline{t}, \sigma, \delta \models \phi\}$. Below we define the product $\langle t, \sigma, \delta\rangle$.

✓ for free second order variables:

$$\begin{array}{c} t \in \mathcal{T}(\Sigma) \\ \delta : \{X_1, \ldots, X_n\} \to 2^{\mathcal{P}os(t)} \end{array} \quad \mapsto \quad t \times \delta \in \mathcal{T}(\Sigma \times \{0,1\}^n)$$

arity of $\langle a, \overline{b}\rangle$ in $\Sigma \times \{0,1\}^n$ = arity of a in Σ.

for all $p \in \mathcal{P}os(t)$, $(t \times \delta)(p) = \langle t(p), b_1, \ldots, b_n\rangle$ where for all $i \leq n$,

- $b_i = 1$ if $p \in \delta(X_i)$,
- $b_i = 0$ otherwise.

✓ free first order variables are interpreted as singletons.

$\mathrm{WS}k\mathrm{S}_0$

We consider a simplified language (wlog).

- no first order variables,
- only second order variables $X, Y \ldots$,
- $$\begin{array}{lll} \text{form} & ::= & X \subseteq Y \mid Y = X \cdot 1 \mid \ldots \mid Y = X \cdot k \\ & & \mid X \subseteq L_a \quad a \in \Sigma \\ & & \mid \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \neg\text{form} \\ & & \mid \exists X \text{ form} \mid \forall X \text{ form} \end{array}$$

interpretation $Y = X \cdot i$: $X = \{x\}$, $Y = \{y\}$ and $y = x \cdot i$.

ex: singleton

WSkS$_0$

We consider a simplified language (wlog).

- no first order variables,
- only second order variables $X, Y \ldots$,
- $$\begin{array}{lll} \text{form} & ::= & X \subseteq Y \mid Y = X \cdot 1 \mid \ldots \mid Y = X \cdot k \\ & & \mid X \subseteq L_a \quad a \in \Sigma \\ & & \mid \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \neg\text{form} \\ & & \mid \exists X \, \text{form} \mid \forall X \, \text{form} \end{array}$$

interpretation $Y = X \cdot i$: $X = \{x\}$, $Y = \{y\}$ and $y = x \cdot i$.

ex: singleton

$$\text{singleton}(X) \equiv \exists Y \quad \big(Y \subseteq X \wedge Y \neq X \wedge \\ \neg\exists Z \, (Z \subseteq X \wedge Z \neq X \wedge Z \neq Y)\big)$$

WSkS $\to$ WSkS$_0$

Lemma :

For all formula $\phi(x_1, \ldots, x_m, X_1, \ldots, X_n) \in$ WSkS,
there exists a formula $\phi'(X'_1, \ldots, X'_m, X_1, \ldots, X_n) \in$ WSkS$_0$
s.t. $\underline{t}, \sigma, \delta \models \phi(x_1, \ldots, x_m, X_1, \ldots, X_n)$
iff $\underline{t}, \sigma' \cup \delta \models \phi'(X'_1, \ldots, X'_m, X_1, \ldots, X_n)$, with $\sigma' : X'_i \mapsto \{\sigma(x_i)\}$.

pr.: several steps of formula rewriting:

1. elimination of $<$,
2. elimination of $S_i(x, y)$ $(i \leq k)$, $L_a(x)$ $(a \in \Sigma)$,
   elimination of first order variables (use singleton(X)).

Compilation of WSkS$_0$ into Automata

notation: $\Sigma_{[m]} := \Sigma \times \{0,1\}^m$.

For all $\phi(X_1, \ldots, X_n) \in$ WSkS$_0$ and $m \geq n$,
we construct a tree automaton $[\![\phi]\!]_m$ over $\Sigma_{[m]}$ recognizing

$$\left\{ t \times \delta \mid \delta : \{X_1, \ldots, X_m\} \to 2^{\mathcal{P}os(t)},\ \underline{t}, \delta \models \phi(X_1, \ldots, X_n) \right\}$$

Projection, Cylindrification

projection

$proj_n : \quad \bigcup_{m \geq n} \mathcal{T}(\Sigma_{[m]}) \to \mathcal{T}(\Sigma_{[n]})$
delete components $n+1, \ldots, m$.

Lemma : projection

For all $n \leq m$, if $L \subseteq \mathcal{T}(\Sigma_{[m]})$ is regular then $proj_n(L)$ is regular.

cylindrification $(m \geq n)$
$cyl_{n,m} : L \subseteq \mathcal{T}(\Sigma_{[n]}) \mapsto \{t \in \mathcal{T}(\Sigma_{[m]}) \mid proj_n(t) \in L\}$

Lemma : cylindrification

For all $n \leq m$, if $L \subseteq \mathcal{T}(\Sigma_{[n]})$ is regular, then $cyl_{n,m}(L)$ is regular.

Compilation: $X_1 \subseteq X_2$

Automaton $[\![X_1 \subseteq X_2]\!]_2$:

- signature $\Sigma_{[2]} = \Sigma \times \{0, 1\}^2$.

Compilation: $X_1 \subseteq X_2$

Automaton $[\![X_1 \subseteq X_2]\!]_2$:

- signature $\Sigma_{[2]} = \Sigma \times \{0,1\}^2$.
- states: q_0
- final states: q_0
- transitions:

$$\begin{array}{rcl} \langle a,0,0\rangle(q_0,\ldots,q_0) & \to & q_0 \\ \langle a,0,1\rangle(q_0,\ldots,q_0) & \to & q_0 \\ \langle a,1,1\rangle(q_0,\ldots,q_0) & \to & q_0 \end{array}$$

For $m \geq 2$,

$$[\![X_1 \subseteq X_2]\!]_m := cyl_{2,m}\big([\![X_1 \subseteq X_2]\!]_2\big)$$

Compilation: $X_1 = X_2 \cdot 1$

Automaton $[\![X_1 = X_2 \cdot 1]\!]_2$:

- signature $\Sigma_{[2]} = \Sigma \times \{0, 1\}^2$.

Compilation: $X_1 = X_2 \cdot 1$

Automaton $[\![X_1 = X_2 \cdot 1]\!]_2$:

- signature $\Sigma_{[2]} = \Sigma \times \{0,1\}^2$.
- states: q_0, q_1, q_2
- final states: q_2
- transitions:

$$
\begin{array}{lcl}
\langle a,0,0\rangle(q_0,\ldots,q_0) & \to & q_0 \\
\langle a,1,0\rangle(q_0,\ldots,q_0) & \to & q_1 \\
\langle a,0,1\rangle(q_1,q_0,\ldots,q_0) & \to & q_2 \\
\langle a,0,0\rangle(q_0,\ldots,q_0,q_2,q_0,\ldots,q_0) & \to & q_2
\end{array}
$$

For $m \geq 2$,

$$[\![X_2 = X_1 \cdot 1]\!]_m := cyl_{2,m}\big([\![X_2 = X_1 \cdot 1]\!]_2\big)$$

Compilation: $X_1 \subseteq L_a$

Automate $[\![X_1 \subseteq L_a]\!]_1$:

- signature $\Sigma_{[2]} = \Sigma \times \{0, 1\}^2$.

Compilation: $X_1 \subseteq L_a$

Automate $[\![X_1 \subseteq L_a]\!]_1$:

- signature $\Sigma_{[2]} = \Sigma \times \{0,1\}^2$.
- states: q_0
- final states: q_0
- transitions:

$$\begin{array}{rcll} \langle a, 0\rangle(q_0, \dots, q_0) & \to & q_0 & \\ \langle b, 0\rangle(q_0, \dots, q_0) & \to & q_0 & (b \neq a) \\ \langle a, 1\rangle(q_0, \dots, q_0) & \to & q_0 & \end{array}$$

For $m \geq 1$,

$$[\![X_1 \subseteq L_a]\!]_m := cyl_{1,m}\big([\![X_1 \subseteq L_a]\!]_1\big)$$

Compilation: Boolean Connectors

- $[\![\phi(X_1, \ldots, X_n) \vee \phi(X_1, \ldots, X_{n'})]\!]_m :=$
  $[\![\phi(X_1, \ldots, X_n)]\!]_m \cup [\![\phi(X_1, \ldots, X_{n'})]\!]_m$
  with $m \geq \max(n, n')$
- $[\![\phi(X_1, \ldots, X_n) \wedge \phi(X_1, \ldots, X_{n'})]\!]_m :=$
  $[\![\phi(X_1, \ldots, X_n)]\!]_m \cap [\![\phi(X_1, \ldots, X_{n'})]\!]_m$
  with $m \geq \max(n, n')$
- $[\![\neg\phi(X_1, \ldots, X_n)]\!]_m := \mathcal{T}(\Sigma_{[m]}) \setminus [\![\phi(X_1, \ldots, X_n)]\!]_m$
  for $m \geq n$.

Compilation: Quantifiers

- $[\![\exists X_{n+1}\, \phi(X_1, \ldots, X_{n+1})]\!]_n := \mathit{proj}_n\big([\![\phi(X_1, \ldots, X_{n+1})]\!]_{n+1}\big)$
- NB: this construction does not preserve determinism.
- $[\![\exists X_{n+1}\, \phi(X_1, \ldots, X_{n+1})]\!]_m :=$
  $\mathit{cyl}_{n,m}\big([\![\exists X_{n+1}\, \phi(X_1, \ldots, X_{n+1})]\!]_n\big)$ for $m \geq n$.
- $\forall = \neg\exists\neg$

Thatcher & Wright's Theorem

Theorem :

For all formula $\phi \in \mathsf{WS}k\mathsf{S}_0$ over Σ without free variables, there exists a tree automaton $\mathcal{A}_\phi$ over Σ, such that $L(\mathcal{A}_\phi) = L(\phi)$.

$\mathcal{A}_\phi = [\![\phi]\!]_0$ can be computed explicitely!

Corollary :

For all formula $\phi \in \mathsf{WS}k\mathsf{S}$ over Σ without free variables there exists a tree automaton $\mathcal{A}_\phi$ over Σ, such that $L(\mathcal{A}_\phi) = L(\phi)$.

using translation of $\mathsf{WS}k\mathsf{S}$ into $\mathsf{WS}k\mathsf{S}_0$ first.

Size of $\mathcal{A}_\phi$

Theorem : Stockmeyer and Meyer 1973

For all n there exists $\exists x_1 \neg \exists y_1 \exists x_2 \neg \exists y_2 \ldots \exists x_n \neg \exists y_n \ \phi \in$ FOL such that for every automaton $\mathcal{A}$ recognizing the same language

$$\mathrm{size}(\mathcal{A}) \geq \left. 2^{2^{\cdots^{2^{\mathrm{size}(\phi)}}}} \right\} n$$

Plan

WSkS: Definition

Automata $\rightarrow$ Logic

Logic $\rightarrow$ Automata

Fragments and Extensions of WSkS

WSkS and FO

Using the 2 directions of the Thatcher & Wright theorem:

$$\mathsf{WS}k\mathsf{S} \ni \phi \mapsto \mathcal{A} \mapsto \exists Y_1 \ldots \exists Y_n \, \psi$$

with $\psi \in$ FOL.

Corollary :

Every WSkS formula is equivalent to a formula $\exists Y_1 \ldots \exists Y_n \, \psi$ with ψ first order.

FO $\subsetneq$ WSkS

Proposition :

The language L of terms with an even number of nodes labeled by a is regular (hence WSkS-definable) but not FO-definable.

pr.: with Ehrenfeucht-Fraïssé games.

Ehrenfeucht-Fraïssé Games

goal: prove FO equivalence of finite structures
(wrt finite set of predicates $\mathcal{L}$).

Definition

for two finite $\mathcal{L}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ $\mathfrak{A} \equiv_m \mathfrak{B}$ iff for all ϕ closed, of quantifier depth m, $\mathfrak{A} \models \phi$ iff $\mathfrak{B} \models \phi$

Ehrenfeucht-Fraïssé Games

$\mathcal{G}_m(\mathfrak{A}, \mathfrak{B})$

1 Spoiler chooses $a_1 \in dom(\mathfrak{A})$ or $b_1 \in dom(\mathfrak{B})$

$1'$ Duplicator chooses $b_1 \in dom(\mathfrak{B})$ or $a_1 \in dom(\mathfrak{A})$

$\vdots$

m' Duplicator chooses $b_m \in dom(\mathfrak{B})$ or $a_m \in dom(\mathfrak{A})$

Duplicator wins if $\{a_1 \mapsto b_1, \ldots, a_m \mapsto b_m\}$ is an injective partial function compatible with the relations of $\mathfrak{A}$ and $\mathfrak{B}$ ($\forall P \in \mathcal{P}$, $P^{\mathfrak{A}}(a_{i_1}, \ldots, a_{i_n})$ iff $P^{\mathfrak{B}}(b_{i_1}, \ldots, b_{i_n})$)
$=$ partial isomorphism.
Otherwise Spoiler wins.

Theorem : Ehrenfeucht-Fraïssé

$\mathfrak{A} \equiv_m \mathfrak{B}$ iff Duplicator has a winning strategy for $\mathcal{G}_m(\mathfrak{A}, \mathfrak{B})$.

Ehrenfeucht-Fraïssé Theorem

more generally: equivalence of finite structures + valuation of n free variables.

for two finite $\mathcal{L}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ and $\alpha_1, \ldots, \alpha_n \in dom(\mathfrak{A})$, $\beta_1, \ldots, \beta_n \in dom(\mathfrak{B})$, $m \geq 0$,

$$\mathfrak{A}, \alpha_1, \ldots, \alpha_n \equiv_m \mathfrak{B}, \beta_1, \ldots, \beta_n$$

iff for all $\phi(x_1, \ldots, x_n)$ of quantifier depth m,

$$\mathfrak{A}, \sigma_a \models \phi(\overline{x}) \text{ iff } \mathfrak{B}, \sigma_b \models \phi(\overline{x})$$

where
$$\begin{aligned} \sigma_a &= \{x_1 \mapsto \alpha_1, \ldots, x_n \mapsto \alpha_n\}, \\ \sigma_b &= \{x_1 \mapsto \beta_1, \ldots, x_n \mapsto \beta_n\}. \end{aligned}$$

Games: the partial isomorphisms must extend $\{\alpha_1 \mapsto \beta_1, \ldots, \alpha_n \mapsto \beta_n\}$.

FO $\subsetneq$ WSkS

let $\Sigma = \{a : 1, \perp : 0\}$.

Lemma :

For all $m \geq 3$ and all $i, j \geq 2^m - 1$,
Duplicator has a winning strategy for $\mathcal{G}_m(a^i(\perp), a^j(\perp))$.

Corollary :

The language $L \subseteq \mathcal{T}(\Sigma)$ of terms with an even number of nodes labeled by a is not FO-definable.

- Star-free languages $=$ FO definable holds for words [McNaughton Papert] but not for trees.
- It is an active field of research to characterize regular tree languages definable in FO.
  e.g. [Benedikt Segoufin 05] $\approx$ locally threshold testable.

Restriction to Antichains

Definition :

An antichain is a subset $P \subseteq \mathcal{P}os(t)$ s.t. $\forall p, p' \in P$, $p \not< p'$ and $p \not> p'$.

antichain-WSkS: second-order quantifications are restricted to antichains.

Theorem :

If $\Sigma_1 = \emptyset$, the classes of antichain-WSkS languages and regular languages over Σ conincide.

Theorem :

chain-WSkS is strictly weaker than WSkS.

MSO on Graphs

Weak second-order monadic theory of the grid

Σ finite alphabet,

$$\mathcal{L}_{\mathsf{grid}} := \{=, S_{\to}, S_{\uparrow}, L_a \mid a \in \Sigma\}$$

Grid $G : \mathbb{N} \times \mathbb{N} \to \Sigma$; Interpretation structure:

$$\underline{G} := \langle \mathbb{N} \times \mathbb{N}, =, x+1, y+1, L_a^{\underline{G}}, L_b^{\underline{G}}, \ldots \rangle.$$

Proposition :

The weak monadic second-order theory of the grid is undecidable.

csq: weak MSO of graphs is undecidable.

MSO on Graphs (Remarks)

- algebraic framework [Courcelle]:
  MSO decidable on graphs generated by a hedge replacement graph grammar = least solutions of equational systems based on graph operations: $\| : 2$, $exch_{i,j} : 1$, $forget_i : 1$, $edge : 0$, $ver : 0$.
- related notion: graphs with bounded *tree width*.
- FO-definable sets of graphs of bounded degree = locally threshold testable graphs (some local neighborhood appears n times with $n <$ threshold - fixed).

Undecidable Extensions

Left concatenation: new predicate

$$S'_1 = \left\{ \langle p, 1 \cdot p \rangle \mid p, 1 \cdot p \in \mathcal{P}os(t) \right\}$$

Proposition :

WS2S + left concatenation predicate is undecidable.

Predicate of equal length.

Proposition :

WS2S + $|x| = |y|$ is undecidable.

MONA

[Klarlund et al 01]
http://www.brics.dk/mona/

- decision procedures for WS1S and WS2S
- by translation of formulas into automata

Part III

Automata for Unranked Trees

Plan

- Unranked Trees and Reasoning Tasks over XML Documents
  - XML Processing
- Automata for Unranked Ordered Trees
- Automata for Unranked Unordered Trees
- Automata for Unranked Mixed Trees
- Regular Languages modulo Associativity and Commutativity
- Verification of XML Updates

Automata for

- ranked terms = first order terms over a signature
  - every symbols has a fixed arity
- → functional program analysis
- unranked terms = finite trees (directed, rooted) labelled over a finite alphabet
  - one node can have arbitrarily (though finitely) many childrens
  - the number of children of a node does not depend on its label
- → Web data, regular term languages modulo A and AC

A Brief History of Tree Automata

- 60's 70's, logic for computer science
  [Thatcher 67], [Takachi 75]: unranked labeled trees
- end of 80's: application to automated deduction
  [Dauchet Tison et al] (ranked trees = terms)
- 90's feature trees over infinite set of features: unranked trees
  [Smolka 92] – applications to computational linguistic
  [Blackburn 94], [Carpenter 92].
- 2000 and later: XML processing - unranked trees
  [Vianu CSL 01], [Schwentick 07].

Imperative Program

[Bouajjani Touili CAV 02]

```
void X() {
  while(true) {
    if Y() {
    thread_create(&t1,Z)
    } else { return }
    }
  }
```

$$
\begin{array}{lcll}
\mathtt{X} & \to & \mathtt{Y}\cdot\mathtt{X} & (r_1)\\
\mathtt{Y} & \to & \mathtt{t} & (r_2)\\
\mathtt{Y} & \to & \mathtt{f} & (r_3)\\
\mathtt{t}\cdot\mathtt{X} & \to & \mathtt{X}\parallel\mathtt{Z} & (r_4)\\
\mathtt{f} & \to & 0 & (r_5)
\end{array}
$$

Reachability analysis:

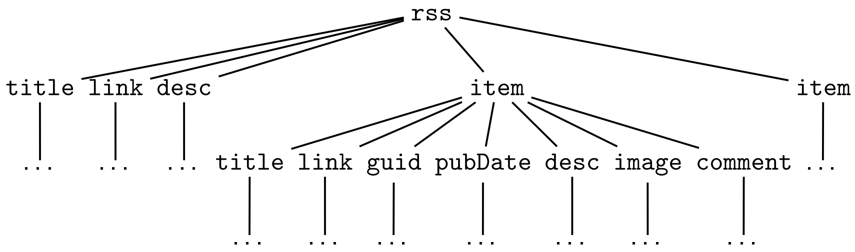
- The set of reachable terms is regular, but
- we want $\cdot$ Associative,
- and $\parallel$ Associative and Commutative,
- and regular term languages are not closed modulo A and AC.

$\to$ consider unranked trees as representative.

Web data (XML Document)

```xml
<rss version="2.0">
    <title>My blog</title>
    <link>http://myblog.blogspot.com</link>
    <description>bla bla bla</description>
    <item>
      <title>Concert</title>
      <link>http://myblog.blogspot.com/me/Mon blog/...</link>
      <guid>5f7da0aa-a593-4a2e</guid>
      <pubDate>Fri, 21 Mar 2009 14:40:02 +0100</pubDate>
      <description>...</description>
      <image href="..."></image>
      <comment link="..." count="0" enabled="0">...</comment>
    </item>
    <item>
      <title>Journée de surf</title>
      ...
    </item>
</rss>
```

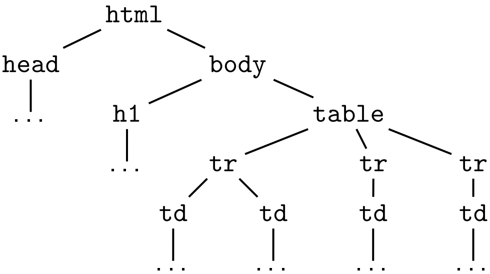
Web Data



rss
title
link
desc
item
item
...
...
...
title
link
guid
pubDate
desc
image
comment
...
...
...
...
...
...
...
...

HTML Document

```
<html>
 <head>...</head>
 <body>
  <h1>...</h1>
  <table>
   <tr>
    <td>...</td>
    <td>...</td>
   </tr>
   <tr>
    <td>...</td>
   </tr>
   <tr>
    <td>...</td>
   </tr>
  </table>
 </body>
</html>
```



XML Document Types

documents = unranked trees

conformity / validation

- class of documents with a predefined structure
  (valid documents)

  ```
  <table>
    <tr> <td> c11 </td> <td> c12 </td> </tr>
    <tr> <td> c21 </td> <td> c22 </td> </tr>
  </table>
  ```

- defined by a schema (DTD, XML schema, Relax NG...)
  = tree language
- All the schema formalisms in use currently correspond to tree automata [Schwentick JCSS 07] [Murata et al 05].

Reasoning Tasks over XML Documents

- type definitions (DTD, XML schema, Relax NG...) $\subseteq$ automata
  - validation = membership problem
  - schema entailment = inclusion problems
- querying
  - query satisfiability = emptiness problem
- integrity constraints (keys, inclusion):
  - consistency [Fan Libkin JACM 02]
- unranked tree transformations
  - type checking
    given L_{in}, regular input language
    T transformation (in XSLT...)
    L_{out} regular output language
    question: do we have $T(L_{\mathsf{in}}) \subseteq L_{\mathsf{out}}$?

Plan

Unranked Trees and Reasoning Tasks over XML Documents

Automata for Unranked Ordered Trees

- Unranked Ordered Trees
- Hedge Automata, Determinism
- Decision Problems
- Binary Encoding
- Boolean Closure
- Minimization

Automata for Unranked Unordered Trees

Automata for Unranked Mixed Trees

Regular Languages modulo Associativity and Commutativity

Verification of XML Updates

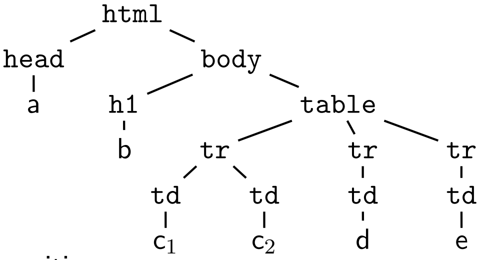
Unranked Ordered Trees

- Σ is a finite alphabet.

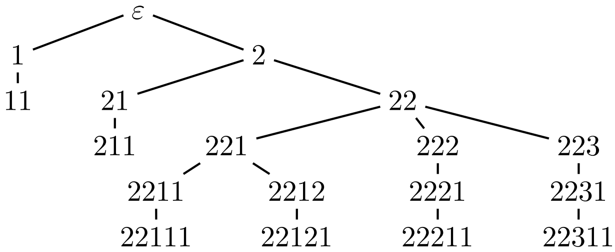
$$\begin{array}{rcl} \text{tree} & := & a(\text{hedge}) \quad (a \in \Sigma) \\ \text{hedge} & := & \text{tree}^* \end{array}$$

- a hedge can be empty. $a()$ is denoted by a.
- The set of all unranked ordered trees over Σ is denoted $\mathcal{O}(\Sigma)$.
- The set of hedges over Σ is denoted $\mathcal{H}(\Sigma)$.
- set of positions $\subset \mathbb{N}^*$: as for terms.

Example: Tree of $\mathcal{O}(\Sigma)$



positions:



Example: Language $\subseteq \mathcal{O}(\Sigma)$

- $\Sigma = \{a, b\}$.
- $L :=$ terms of $\mathcal{O}(\Sigma)$
  - height at most 1,
  - root is labelled by a,
  - even number of leaves, all leaves labelled by b.
- $L = \{a, a(bb), a(bbbb), ...\}$.
- finite description: $L = a\big((bb)^*\big)$

Hedge Automata (HA)

Definition : Hedge Automata

A Hedge Automaton (HA) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where Q is a finite set of states, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $a(L) \to q$ with $a \in \Sigma$ and $L \subseteq Q^*$ is a regular language.

A run of $\mathcal{A}$ on $t \in \mathcal{O}(\Sigma)$ is a tree $r \in \mathcal{O}(Q)$ such that

- r and t have the same domain,
- for all $p \in \mathcal{P}os(t)$, with $t(p) = a$, $r(p) = q$, there exists $a(L) \to q \in \Delta$ such that $r(p1) \dots r(pn) \in L$, where n is the number of successors of p in $\mathcal{P}os(t)$.

The run r is accepting (successful) iff $r(\varepsilon) \in Q^{\mathsf{f}}$.

HA Languages

- language of $\mathcal{A}$: $L(\mathcal{A})$ is the set of terms on which there exists an accepting run of $\mathcal{A}$,
- language of $\mathcal{A}$ in state $q \in Q$: $L(\mathcal{A}, q)$ is the set of terms t such that there exists a run r of $\mathcal{A}$ on t with $r(\varepsilon) = q$,
- $L(\mathcal{A}) = \bigcup_{q \in Q_\mathsf{f}} L(\mathcal{A}, q)$.
- equivalently, $L(\mathcal{A}, q)$ is the smallest set of terms $a(t_1, \ldots, t_n) \in \mathcal{O}(\Sigma)$ $(n \geq 0)$ such that there exists a transition $a(L) \to q$, and states $q_1, \ldots, q_n \in Q$ with $t_i \in L(\mathcal{A}, q_i)$ s.t. $i \leq n$ and $q_1 \ldots q_n \in L$.

HA Language: Example 1

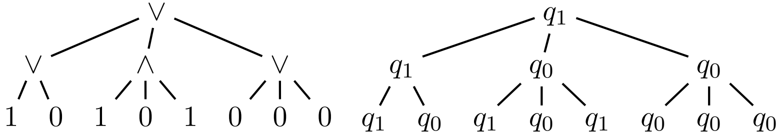
- $\Sigma = \{a, b\}$.
- $L :=$ terms of $\mathcal{O}(\Sigma)$
  - height 1,
  - root is labelled by a,
  - even number of leaves, all leaves labelled by b.
- $L = \{a, a(bb), a(bbbb), ...\}$.
- finite description: $L = L(\mathcal{A})$ with
  $\mathcal{A} := \big(\Sigma, \{q_a, q_b\}, \{q_a\}, \{b \to q_b, a((q_b q_b)^*) \to q_a\}\big)$.

Boolean Expressions with Variadic $\wedge$ and $\vee$

- $\Sigma = \{\wedge, \vee, 0, 1\}$,
- states $\{q_0, q_1\}$,
- transitions:

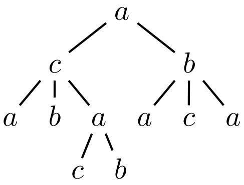
$$
\begin{array}{rcl@{\qquad}rcl}
0 & \to & q_0 & 1 & \to & q_1 \\
\wedge(q_1^* \, q_0 \, (q_0 \mid q_1)^*) & \to & q_0 & \wedge(q_1 \, q_1^*) & \to & q_1 \\
\vee(q_0 \, q_0^*) & \to & q_0 & \vee(q_0^* \, q_1 \, (q_0 \mid q_1)^*) & \to & q_1 \\
\neg(q_0) & \to & q_1 & \neg(q_1) & \to & q_0
\end{array}
$$

- example: Boolean expression and associated run



HA Language: Example 3

- $\Sigma = \{a, b, c\}$.
- $L :=$ terms of $\mathcal{O}(\Sigma)$
  - with 2 b's at positions p_1 and p_2, and
  - one c on the smallest common ancestor of p_1 and p_2.

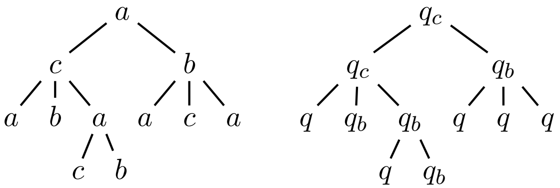


HA Language: Example 3

- $\Sigma = \{a, b, c\}$.
- $L :=$ terms of $\mathcal{O}(\Sigma)$
  - with 2 b's at positions p_1 and p_2, and
  - one c on the smallest common ancestor of p_1 and p_2.

$\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$, avec $Q = \{q, q_b, q_c\}$, $Q^{\mathsf{f}} = \{q_c\}$, $\Delta =$

$$
\begin{array}{rcl rcl rcl}
a(Q^*) & \to & q & a(Q^* q_b Q^*) & \to & q_b & a(Q^* q_c Q^*) & \to & q_c \\
b(Q^*) & \to & q_b & c(Q^* q_b Q^*) & \to & q_b & b(Q^* q_c Q^*) & \to & q_c \\
c(Q^*) & \to & q & c(Q^* q_b Q^* q_b Q^*) & \to & q_c & c(Q^* q_c Q^*) & \to & q_c
\end{array}
$$



Normalized Hedge Automata

Definition :

A HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is called normalized if for all $a \in \Sigma$ and $q \in Q$, there is at most one transition of the form $a(L) \to q$ in Δ.

When $\mathcal{A}$ is normalized, we denote $a(L_{a,q}) \to q$ the unique transition with a and q.

Proposition :

For all HA $\mathcal{A}$, there exista a normalized HA $\mathcal{A}_{\mathsf{n}}$ recognizing the same language.

The size of $\mathcal{A}_{\mathsf{n}}$ is linear in the size of $\mathcal{A}$.

Complete and Deterministic Hedge Automata

Semantical definitions

Definition :

A HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is complete if for all $t \in \mathcal{O}(\Sigma)$, there exists at least one state $q \in Q$ s.t. $t \in L(\mathcal{A}, q)$.

Definition :

A HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is deterministic if for all $t \in \mathcal{O}(\Sigma)$, there exists at most one state $q \in Q$ s.t. $t \in L(\mathcal{A}, q)$.

Complete and Deterministic Hedge Automata

Syntactical definitions

Definition :

A HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is complete if
for all $a \in \Sigma$ and all finite sequence $q_1, \ldots, q_n \in Q^*$,
there exists a transition $a(L) \to q \in \Delta$ with $q_1 \ldots q_n \in L$.

Definition :

A HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is deterministic if
for all transitions $a(L_1) \to q_1$ and $a(L_2) \to q_2$ in Δ,
either $L_1 \cap L_2 = \emptyset$, or $q_1 = q_2$.

Determinism: Examples

HA for Boolean expressions evaluation : deterministic.
$\Sigma = \{\wedge, \vee, 0, 1\}$, $Q = \{q_0, q_1\}$ et Δ:

$$\begin{array}{rclrcl}
0 & \rightarrow & q_0 & 1 & \rightarrow & q_1 \\
\wedge(q_1^* \, q_0 \, (q_0 \mid q_1)^*) & \rightarrow & q_0 & \wedge(q_1 q_1^*) & \rightarrow & q_1 \\
\vee(q_0 q_0^*) & \rightarrow & q_0 & \vee(q_0^* \, q_1 \, (q_0 \mid q_1)^*) & \rightarrow & q_1
\end{array}$$

Language with 2 b's and common ancestor c: not deterministic.
$\Sigma = \{a, b, c\}$, $Q = \{q, q_b, q_c\}$, $Q^{\mathsf{f}} = \{q_c\}$, Δ:

$$\begin{array}{rclrclrcl}
a(Q^*) & \rightarrow & q & a(Q^* q_b Q^*) & \rightarrow & q_b & a(Q^* q_c Q^*) & \rightarrow & q_c \\
b(Q^*) & \rightarrow & q_b & c(Q^* q_b Q^*) & \rightarrow & q_b & b(Q^* q_c Q^*) & \rightarrow & q_c \\
c(Q^*) & \rightarrow & q & c(Q^* q_b Q^* q_b Q^*) & \rightarrow & q_c & c(Q^* q_c Q^*) & \rightarrow & q_c
\end{array}$$

HA Completion

Proposition :

For all HA $\mathcal{A}$, there exists a complete HA $\mathcal{A}_\mathsf{c}$ recognizing the same language.

The size of $\mathcal{A}_\mathsf{c}$ is linear is the size of $\mathcal{A}$.

HA Completion

Proposition :

For all HA $\mathcal{A}$, there exists a complete HA $\mathcal{A}_\mathsf{c}$ recognizing the same language.

The size of $\mathcal{A}_\mathsf{c}$ is linear is the size of $\mathcal{A}$.

pr.: add a *trash* state $q_\perp$ and transitions :

$$a\Big(\bigcap_{q \in Q} Q^* \setminus L_{a,q} \cup Q_\perp^* q_\perp Q_\perp^*\Big) \to q_\perp$$

HA Determinization

Proposition :

For all HA $\mathcal{A}$, there exists a deterministic HA $\mathcal{A}_{\mathsf{d}}$ recognizing the same language.

The size of $\mathcal{A}_{\mathsf{d}}$ is exponential in the size of $\mathcal{A}$ (lower bound).
pr.: subset construction

HA Determinization

Proposition :

For all HA $\mathcal{A}$, there exists a deterministic HA $\mathcal{A}_\mathsf{d}$ recognizing the same language.

The size of $\mathcal{A}_\mathsf{d}$ is exponential in the size of $\mathcal{A}$ (lower bound).

pr.: subset construction for $\mathcal{A} = (Q, Q_\mathsf{f}, \Delta)$ (normalised):

- $\mathcal{A}_d = \left(2^Q, Q^\mathsf{f}, \Delta_d\right)$,
- $Q^\mathsf{f} = \{S \subseteq Q \mid S \cap Q_\mathsf{f} \neq \emptyset\}$
- Δ_d: $a(L_{a,S}) \to S$ $(S \subseteq Q)$

$$L_{a,S} = \bigcap_{q \in S} S_{a,q} \setminus \bigcup_{q \notin S} S_{a,q}$$

with

$$S_{a,q} = \{S_1 \dots S_n \in Q_d^* \mid \exists q_1 \in S_1, \dots, \exists q_n \in S_n, q_1 \dots q_n \in L_{a,q}\}$$

HA: Membership Decision

Proposition : $\in$

The problem of membership is decidable in polynomial time for the HAs whose languages $L_{a,q}$ are given by NFAs.

- linear time for DHA whose languages $L_{a,q}$ are given by DFA,
- NP-complete if the languages $L_{a,q}$ are given by alternating automata.

HA: Emptiness Decision

Proposition : $\emptyset$

The problem of emptiness is decidable in polynomial time for HAs whose languages $L_{a,q}$ are given by NFAs.

pr.: state marking. $M \subseteq Q$,

- initialy, $M = \emptyset$.
- at each step , if $a(L_{a,q}) \to q \in \Delta$ and $L_{a,q} \cap M^* \neq \emptyset$ then $M := M \cup \{q\}$.

- PSPACE-complete if the languages $L_{a,q}$ are given by alternating automata.

HA: Decision of Inclusion and Equivalence

Proposition : $\subseteq$, $\equiv$

The problem of inclusion (resp. equivalence) is EXPTIME-complete for HAs whose languages $L_{a,q}$ are given by NFAs.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap (\mathcal{O}(\Sigma) \setminus L(\mathcal{A}_2)) = \emptyset$.

- in PTIME for DHAs whose languages $L_{a,q}$ are given by DFAs.
- PSPACE-complete for DHAs whose languages $L_{a,q}$ are given by alternating automata.

Currying

Transformation into binary trees with @ and constants.

We associate the following signature to an alphabet Σ:

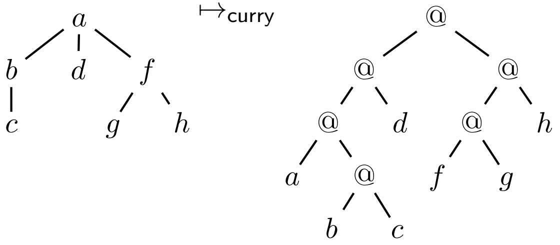
$$\Sigma_{@} := \{a : 0 \mid a \in \Sigma\} \cup \{@ : 2\}$$

The function curry : $\mathcal{O}(\Sigma) \to \mathcal{T}(\Sigma_{@})$, is defined recursively:

- $\text{curry}(a) := a$,
- $\text{curry}\big(a(t_1, \ldots, t_n)\big) := @\big(\text{curry}\big(a(t_1, \ldots, t_{n-1})\big), \text{curry}(t_n)\big)$.

Currying: Example 1

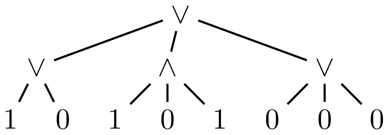
- $\mathsf{curry}(a) = a$,
- $\mathsf{curry}\big(a(t_1, \ldots, t_n)\big) = @\big(\mathsf{curry}\big(a(t_1, \ldots, t_{n-1})\big), \mathsf{curry}(t_n)\big)$



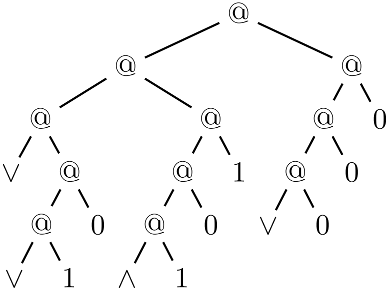
Currying: Example 2

- $\mathsf{curry}(a) = a$,
- $\mathsf{curry}\big(a(t_1, \ldots, t_n)\big) = @\big(\mathsf{curry}\big(a(t_1, \ldots, t_{n-1})\big), \mathsf{curry}(t_n)\big)$

transforming unranked Boolean expressions into binary.



$\mapsto_{\mathsf{curry}}$



Curying: Properties

Lemma :

curry is a bijection from $\mathcal{O}(\Sigma)$ into $\mathcal{T}(\Sigma_@)$.

Proposition :

$L \subseteq \mathcal{O}(\Sigma)$ is a HA language iff curry(L) is regular.

pr.:: Let $\mathcal{A} = \langle Q, Q^{\mathsf{f}}, \Delta\rangle$ normalized HA, and
$B_{a,q} = \langle P_{a,q}, \mathsf{init}_{a,q}, F_{a,q}, R_{a,q}\rangle$ be a NFA recognizing $L_{a,q}$ f.a.
$a \in \Sigma$, $q \in Q$, $a(L_{a,q}) \to q \in \Delta$.
curry$(\mathcal{A}) = \langle Q \cup \bigcup_{a,q} P_{a,q}, Q^{\mathsf{f}}, \Delta'\rangle$ where Δ' contains:

- $a \to q$ if $B_{a,q}$ recognizes the empty word,
- $a \to \mathsf{init}_{a,q}$ for all $q \in Q$,
- $@(p, q) \to p'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$, and
- $@(p, q) \to q'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$ and $p' \in F_{a',q'}$.

HA: Boolean Operations

Proposition :

The class of HA languages is closed under union, intersection and complement.

pr.: $\mathsf{curry}(L_1 \cup L_2) = \mathsf{curry}(L_1) \cup \mathsf{curry}(L_2)$
Hence we can reuse the construction for ranked TA
$L_1 \cup L_2 = \mathsf{curry}^{-1}(\mathsf{curry}(L_1) \cup \mathsf{curry}(L_2))$

HA: Closure under Morphisms

projection $h : \mathcal{O}(\Sigma) \to \mathcal{O}(\Sigma')$, defined by extension to trees of an application $h : \Sigma \to \Sigma'$.

$$h(L) = \{h(t) \mid t \in L\} \quad \text{and} \quad h^{-1}(L') = \{t \in \mathcal{O}(\Sigma) \mid h(t) \in L'\}$$

Proposition :

The class of HA languages is closed under projections and inverse projections.

pr.: it follows form the closure of regular ranked tree languages under linear morphisms and inverses morphisms.

Minimization

Definition of minimal deterministic HA:
2 questions must be addressed

1. for which definition of determinism?
   (minimization makes sense only for deterministic automata)
2. what to minimize?

Minimization

For ranked tree automata, the answer to both questions is clear:

1. ranked DTA: every step of computation is deterministic
2. we want to minimize the number of states

For unranked tree automata, this is not so clear:

1. even for DHA[DFA] (DHA whose horizontal languages are defined by DFA), if we have $a(L) \to q$ and $a(L') \to q'$ and $L \cap L' = \emptyset$, in configuration $a(q_1 \dots q_n)$ we must test both $q_1 \dots q_n \in L$ and $q_1 \dots q_n \in L'$ before firing the right transition. Which one is tested first?
   In the construction of a TA for curry(HA), we choose ND.
2. there are states for the DHA and for the DFAs for the horizontal languages.

Minimization of DHA

First approach: we ignore the formalism for horizontal languages, i.e. we chose

1. DFAs (whatever for the horizontal automata)
2. number of states of the DFA

Congruence of a language $L \subseteq \mathcal{O}(\Sigma)$:

$$s \equiv_L t \quad \text{iff} \quad \forall C\ C[s] \in L \Leftrightarrow C[t] \in L$$

Minimal DHA for the HA language L:

- states: $\{[t]_{\equiv_L} \mid t \in \mathcal{O}(\Sigma)\}$,
- final states: $\{[t]_{\equiv_L} \mid t \in L\}$ (we simply write $[t]$ below),
- transitions:
  $\{a(L_{a,[t]}) \to [t] \mid L_{a,[t]} = \{[t_1] \dots [t_n] \mid a(t_1 \dots t_n) \equiv_L t\}\}$,

Minimization of DHA (2)

2 drawbacks for the first approach

- the complexity of the effective construction depends on the formalism for horizontal languages.
- no analogous of Myhill-Nerode theorem for DFA or DTA (ranked):

$$\begin{array}{lcl} L \text{ is an HA language} & \Rightarrow & \equiv_L \text{ has finite index} \\ L \text{ is an HA language} & \not\Leftarrow & \equiv_L \text{ has finite index} \end{array}$$

Minimization of DHA[DTA]

Second approach: we consider both vertical and horizontal states and transitions, i.e. we chose

1. DFA[DTA] (DHA whose horizontal language are defined by disjoint DFAs)
2. number of states of the DFA + number of states of the horizontal (disjoint) DTAs

Minimization of DHA[DTA] (2)

first idea: use the curry encoding and minimize the ranked TA.

problem: the TA associated to an HA wrt curry is not deterministic; we have $a \to \mathsf{init}_{a,q}$ for all $q \in Q$.
other question: uniqueness of minimal automaton?

$\to$ stepwise automata [Niehren et al 04]:
one unique transition from each $a \in \Sigma$ to the start state of a deterministic machine that will read the state sequence below a and output a state.

Moreover, vertical states $=$ horizontal states.

Deterministic Stepwise Automata

Definition : stepwise automata

A deterministic stepwise hedge automaton (DSHA) is a tuple $\mathcal{A} = (\Sigma, Q, Q_f, \delta_0, \delta)$, where Σ, Q, and Q_f are as usual, $\delta_0 : \Sigma \to Q$ is a function assigning to each letter of the alphabet an initial state, and $\delta : Q \times Q \to Q$ is the transition function.

For $a \in \Sigma$,
$$\begin{array}{rcl} \delta_a : & & Q^* \to Q \\ \delta_a(\varepsilon) & = & \delta_0(a) \\ \delta_a(w \cdot q) & = & \delta(\delta_a(w), q) \end{array}$$

A run of $\mathcal{A}$ on $t \in \mathcal{O}(\Sigma)$ is a tree $r \in \mathcal{O}(Q)$ such that

- r and t have the same domain,
- for all $p \in \mathcal{P}os(t)$, $r(p) = \delta_{t(p)}(r(p1) \dots r(pn))$, where n is the number of successors of p in $\mathcal{P}os(t)$.

The run r is accepting (successful) iff $r(\varepsilon) \in Q^{\mathsf{f}}$.

Stepwise Automata & Ranked TA

stepwise DSHA $\mathcal{A}$	ranked DTA curry$(\mathcal{A})$
$\delta_0(a) = q$	$a \to q$
$\delta(q_1, q_2) = q$	$@(q_1, q_2) \to q$

Lemma :

For all $t, t' \in \mathcal{O}(\Sigma)$ and $q, q' \in Q$,
if $t \in L(\mathcal{A}, q)$ and $t' \in L(\mathcal{A}, q')$, then $t \,@\, t' \in L(A, \delta(q, q'))$.

Lemma :

For all DSHA $\mathcal{A}$, curry$(L(\mathcal{A})) = L($curry$(\mathcal{A}))$.

Minimal Stepwise Automata

Corollary :

DSHA recognize all HA unranked tree languages.

Corollary :

For each HA language $L \subseteq \mathcal{O}(\Sigma)$ there is a unique (up to renaming of states) minimal DSHA accepting L.

Plan

- previous part: unranked ordered trees
  - XML documents
  - hedge automata (HA)
  - see TATA book `http://tata.gforge.inria.fr` chapter 8
- this part: unranked unordered trees
  - web data
  - Presburger automata (PA)
  - see [Seidl, Schwentick, Muscholl. *Numerical Document Queries*. PODS 03]

Unranked Unordered Trees

- Σ is a finite alphabet.

$$\begin{array}{rcl} \mathsf{tree} & := & a(\mathsf{multiset}) \quad (a \in \Sigma) \\ \mathsf{multiset} & := & \{\mathsf{tree}, \ldots, \mathsf{tree}\} \end{array}$$

- $a\big(\{t_1, \ldots, t_n\}\big)$ is denoted $a(t_1, \ldots, t_n)$.
- rem: the multiset can be empty. $a(\emptyset)$ is denoted a.
- The set of unranked unordered trees over Σ is denoted $\mathcal{U}(\Sigma)$.

Examples of Languages of Trees of $\mathcal{U}(\Sigma)$

- $\Sigma = \{a, b\}$.
- terms of $\mathcal{U}(\Sigma)$
  - of height 1,
  - with one b at the root,
  - the leaves are a or b:

  $i.$ with an even number of a ($\in$ HA),

  $ii.$ with the same number of a than b ($\notin$ HA).

Presburger Arithmetic

Presburger Formulae:

$$
\begin{array}{lcll}
\text{term} & ::= & x & \text{(first order variables)} \\
 & \mid & n & \text{(natural number)} \\
 & \mid & \text{term} + \text{term} & \\
\text{form} & ::= & \text{term} = \text{term} & \\
 & \mid & \neg\text{form} \mid \text{form} \vee \text{form} \mid \text{form} \wedge \text{form} & \\
 & \mid & \forall x\, \text{form} \mid \exists x\, \text{form} &
\end{array}
$$

Interpretation in the domain of natural numbers.
$(n_1, \ldots, n_p) \models \phi(x_1, \ldots, x_p)$ ($x_1, \ldots, x_p$ free variables)
iff $\phi(n_1, \ldots, n_p)$ is evaluated to *true*.

Presburger Arithmetic

- notation: terms $n\,x$ for $\underbrace{x + \ldots + x}_{n}$,
- the natural can be restricted to 0 and 1,
- the atoms can be restricted to $x = n$ and $x = y + z$.

Examples:

- $x \leq y$

Presburger Arithmetic

- notation: terms $n\,x$ for $\underbrace{x + \ldots + x}_{n}$,
- the natural can be restricted to 0 and 1,
- the atoms can be restricted to $x = n$ and $x = y + z$.

Examples:

- $x \leq y \equiv \exists x'\, y = x + x'$.
- $odd(x)$

Presburger Arithmetic

- notation: terms $n\,x$ for $\underbrace{x + \ldots + x}_{n}$,
- the natural can be restricted to 0 and 1,
- the atoms can be restricted to $x = n$ and $x = y + z$.

Examples:

- $x \leq y \equiv \exists x'\, y = x + x'$.
- $odd(x) \equiv \exists y\, x = y + y + 1$.

Presburger Arithmetic

Theorem : Presburger Arithmetic

Presburger Arithmetic is decidable.

- lower bound 2-EXPTIME (Fischer and Rabin 1974)
- upper bound 3-EXPTIME (Klaedtke 2004, with an automata construction)
- NP-complete for the existential fragment.

Presburger Arithmetic

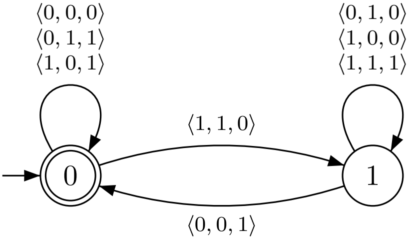
Decidability of Presburger Arithmetic.

We can associate to a formula $\phi(x_1, \ldots, x_p)$ a finite automaton over the alphabet $\{0,1\}^p$ recognizing the set of $\langle b_{1,1}, \ldots, b_{p,1}\rangle \ldots \langle b_{1,k}, \ldots, b_{p,k}\rangle$ such that $b_{1,1} \ldots b_{1,k}$, ..., $b_{p,1} \ldots b_{p,k}$ are the binary representations of integers $n_1, \ldots, n_p$ satisfying ϕ.

Hence we can decide wether there exists $n_1, \ldots, n_p$ such that $(n_1, \ldots, n_p) \models \phi(x_1, \ldots, x_p)$.

Presburger Arithmetic and Automata

finite automaton for $x_1 + x_2 = x_3$



Parikh Projection

$\Sigma = \{a_1, \ldots, a_p\}$.

Definition : Parikh Projection

The Parikh projection of a word $w \in \Sigma^*$ is the tuple $\#(w) := (m_1, \ldots, m_p)$ where m_i $_{(i \leq p)}$ is the number of occurrences of a_i in w.

For a set $L \subseteq \Sigma^*$, we denote $\#(L) := \{\#(w) \mid w \in L\}$.

Theorem :

For all context-free language $L \subseteq \Sigma^*$, there exists a Presburger formula $\phi(x_1, \ldots, x_p)$ such that $\#(L) := \{(n_1, \ldots, n_p) \models \phi(x_1, \ldots, x_p)\}$.

When L is regular, the Presburger formula is computed in linear time (in the size of the NFA defining L).

Theorem :

For all Presburger formula $\phi(x_1, \ldots, x_p)$, one can build a NFA $\mathcal{A}$ such that $\#\big(L(\mathcal{A})\big) = \{(n_1, \ldots, n_k) \models \phi(x_1, \ldots, x_k)\}$.

Semi-linear sets

Definition :

- A *linear* set is a subset of $\mathbb{N}^p$ of the form $\left\{\overline{v_0} + \overline{v_1} + \ldots + \overline{v_m} \;\middle|\; m \geq 0, \overline{v_1}, \ldots, \overline{v_m} \in B\right\}$, for $\overline{v_0} \in \mathbb{N}^p$ and $B \subset \mathbb{N}^p$ finite (fixed).
- A *semi-linear* set is a finite union of linear sets.

Theorem : Parikh

Models of Presburger formulae $\equiv$ semi-linear sets.

Presburger Automata (PA)

Definition : Presburger Automata

A *Presburger Automaton* (PA) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where $Q = \{q_1, \ldots, q_p\}$ is a finite set od *states*, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $a(\phi) \to q$ with $a \in \Sigma$, $q \in Q$, and $\phi = \phi(x_1, \ldots, x_p)$ is a Presburger formula with one free variable x_i for each state q_i.

The language of $\mathcal{A}$ in state $q \in Q$, denoted $L(\mathcal{A}, q)$, is the smallest subset of terms $a(t_1, \ldots, t_n) \in \mathcal{U}(\Sigma)$ such that

- there exists $i_1, \ldots, i_n \leq p$ such that for all $j \leq n$, $t_j \in L(\mathcal{A}, q_{i_j})$,
- there exists a transition $a(\phi) \to q \in \Delta$ such that $\#(q_{i_1}, \ldots, q_{i_n}) \models \phi(x_1, \ldots, x_p)$.

The language of $\mathcal{A}$ is $L(\mathcal{A}) = \bigcup_{q \in Q_{\mathsf{f}}} L(\mathcal{A}, q)$.

PA: Example 1

$\Sigma = \{a, b, f\}$.
Set of trees of $\mathcal{U}(\Sigma)$ where all the a and b label the leaves:

PA: Example 1

$\Sigma = \{a, b, f\}$.
Set of trees of $\mathcal{U}(\Sigma)$ where all the a and b label the leaves:

$$\mathcal{A} = \big(\{q\}, \{q\}, \{a(x_q = 0) \to q, b(x_q = 0) \to q, f(\mathit{true}) \to q\}\big)$$

PA: Example 2

$\Sigma = \{a, b, c\}$.
Set of trees of $\mathcal{U}(\Sigma)$ with the same number of a and of b under each node.

PA: Example 2

$\Sigma = \{a, b, c\}$.
Set of trees of $\mathcal{U}(\Sigma)$ with the same number of a and of b under each node.

$$\mathcal{A} = \big(\{q_a, q_b, q\}, \{q_a, q_b, q\}, \{a(\phi) \to q_a, b(\phi) \to q_b, c(\phi) \to q\}\big)$$

with $\phi \equiv x_{q_a} = x_{q_b}$.

PA: Example 3

$\Sigma = \{a, b\}$.
Set of trees of $\mathcal{U}(\Sigma)$ where all internal node are labeled by a and every node labeled by a has at least as much sons without b than sons containing b.

PA: Example 3

$\Sigma = \{a, b\}$.
Set of trees of $\mathcal{U}(\Sigma)$ where all internal node are labeled by a and every node labeled by a has at least as much sons without b than sons containing b.

$Q = Q_{\mathsf{f}} = \{q_a, q_b\}$.
The state q_b accepts the trees containing a b and q_a accepts the others.

$$\Delta = \left\{ \begin{array}{lcl} a(x_{q_a} \geq x_{q_b} = 0) & \rightarrow & q_a \end{array} \quad \begin{array}{rcl} a(x_{q_a} \geq x_{q_b} > 0) & \rightarrow & q_b \\ b(x_{q_a} = x_{q_b} = 0) & \rightarrow & q_b \end{array} \right\}$$

Normalized Presburger Automata

Definition :

A PA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ sur Σ is called *normalized* if for all $a \in \Sigma$ and $q \in Q$, there is at most one transition of the form $a(\phi) \to q$ in Δ.

When $\mathcal{A}$ is normalized, we note $a(\phi_{a,q}) \to q$ the unique transition with a and q.

Proposition :

For all PA $\mathcal{A}$, there exists a normalized PA $\mathcal{A}_n$ recognizing the same language.

The size of $\mathcal{A}_n$ is linear in the size of $\mathcal{A}$.

Complete Presburger Automata

Definition : Complete PA

A PA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is *complete* if for all $a \in \Sigma$ and all $q_{i_1}, \ldots, q_{i_n} \in Q^*$ there exists at least one transition $a(\phi) \to q \in \Delta$ such that $\#(q_{i_1}, \ldots, q_{i_n}) \models \phi$.

Proposition : Completion

For all PA $\mathcal{A}$, there exists a complete PA $\mathcal{A}_c$ recognizing the same language.

The size of $\mathcal{A}_c$ is linear in the size of $\mathcal{A}$.

Complete Presburger Automata

Definition : Complete PA

A PA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is *complete* if for all $a \in \Sigma$ and all $q_{i_1}, \ldots, q_{i_n} \in Q^*$ there exists at least one transition $a(\phi) \to q \in \Delta$ such that $\#(q_{i_1}, \ldots, q_{i_n}) \models \phi$.

Proposition : Completion

For all PA $\mathcal{A}$, there exists a complete PA $\mathcal{A}_c$ recognizing the same language.

The size of $\mathcal{A}_c$ is linear in the size of $\mathcal{A}$.

pr.: Let $\mathcal{A} = (Q, Q_{\mathsf{f}}, \Delta)$ be a PA (normalized).
We add the state $q_\perp$: $\mathcal{A}_c = \big(Q \cup \{q_\perp\}, Q_{\mathsf{f}}, \Delta_c\big)$ with

$$\begin{array}{rrcl} \Delta_c := & a\big(\phi_{a,q} \wedge x_{q_\perp} = 0\big) & \to & q \text{ s.t. } a(\phi_{a,q}) \to q \in \Delta \\ \cup & a\big(\bigwedge_{q \in Q} \neg\phi_{a,q} \vee x_{q_\perp} > 0\big) & \to & q_\perp \end{array}$$

Deterministic Presburger Automata

Definition : Deterministic PA

A PA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ over Σ is *deterministic* if for all $a \in \Sigma$ and all $q_{i_1}, \ldots, q_{i_n} \in Q^*$, if $a(\phi) \to q \in \Delta$ and $a(\phi') \to q' \in \Delta$ are such that $\#(q_{i_1}, \ldots, q_{i_n}) \models \phi$ and $\#(q_{i_1}, \ldots, q_{i_n}) \models \phi'$, then $q = q'$.

Lemma :

The determinism of PA is decidable.

pr.: Determinism is expressed the Presburger formula

$$\bigwedge_{\substack{a(\phi) \to q \\ a(\phi') \to q' \\ q \neq q'}} \neg(\phi \wedge \phi')$$

Determinization of PA

Proposition : Determinization

For all PA $\mathcal{A}$, there exists a deterministic PA $\mathcal{A}_d$ recognizing the same language.

The size of $\mathcal{A}_d$ is exponential in the size of $\mathcal{A}$ (lower bound).

pr.: Let $\mathcal{A} = (Q, Q_\mathsf{f}, \Delta)$, normalized with $Q = \{q_1, \ldots, q_b\}$.

$$\mathcal{A}_d = \left(2^Q, \{S \subseteq Q \mid S \cap Q_\mathsf{f} \neq \emptyset\}, \Delta_d\right)$$

Presburger formulae in Δ_d: a free variable x_S for each $S \subseteq Q$.

$$a\Big(\bigwedge_{q \in S} \psi_{a,q} \wedge \bigwedge_{q \notin S} \neg\psi_{a,q}\Big) \to S$$

with

$$\psi_{a,q} \equiv \underset{p \in Q}{\exists} x_p \underset{\substack{P \subseteq Q \\ p \in P}}{\exists} x_{P,p}\, \phi_{a,q} \wedge \bigwedge_{p \in Q} x_p = \sum_{\substack{P \subseteq Q \\ p \in P}} x_{P,p} \wedge \bigwedge_{P \subseteq Q} x_P = \sum_{p \in P} x_{P,p}$$

ε-Transitions and Elimination

Remark :

PA with transitions $q \to q' \equiv$ PA.

PA: Boolean Operations

Proposition :

The class of languages of PA is closed under intersection and complementation.

- $\cup$ disjoint union (linear) or product (quadratic, preserves determinism).
- $\cap$ de Morgan law or product (quadratic).
- $\neg$ complete, determinize, invert final states (exponential).

PA: Membership Decision

Proposition : $\in$

The problem of membership is decidable for PA.

- in polynomial time for DPA,
- NP-complete for NPA.

PA: Emptiness Decision

Proposition : $\emptyset$

The problem of emptiness is decidable in polynomial time for PA.

pr.: states marking, construction of $M_i \subseteq Q$.

- initially, $M_0 = \emptyset$.
- at each step, if for $q \in Q \setminus M_i$, $\bigwedge_{p \in Q \setminus M_i} x_p = 0 \wedge \bigvee_{a \in \Sigma} \phi_{a,q}$ is satisfiable, then $M_{i+1} := M_i \cup \{q\}$.

PA: Decision of Inclusion, Equivalence

Proposition : $\subseteq$, $\equiv$

The problems of inclusion and equivalence are decidable for PA.

pr.: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap (\mathcal{U}(\Sigma) \setminus L(\mathcal{A}_2)) = \emptyset$.

Weak Second Order Monadic Logic of Presburger

Syntax of formulae of PMSO.

- first order variables $x \ldots$
- second order variables $X \ldots$
- $$\begin{array}{rcl}
\text{form} & ::= & x = y \mid x \downarrow y \mid a(x) \mid x \in X \mid x/\text{pres} \quad (a \in \Sigma) \\
 & \mid & \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \neg\text{form} \\
 & \mid & \exists x\,\text{form} \mid \exists X\,\text{form} \mid \forall x\,\text{form} \mid \forall X\,\text{form} \\
\text{pres} & ::= & \text{term} = \text{term} \\
 & \mid & \neg\text{pres} \mid \text{pres} \vee \text{pres} \mid \text{pres} \wedge \text{pres} \\
 & \mid & \forall z\,\text{pres} \mid \exists z\,\text{pres} \\
\text{term} & ::= & z \quad \text{(first order integer variable)} \\
 & \mid & n \quad \text{(natural number)} \\
 & \mid & [X] \quad (X \text{ second order var.}) \\
 & \mid & \text{term} + \text{term}
\end{array}$$

formulas pres such that the variables z are bounded.

rem. no atoms $x \to y$ as for ordered trees.

Weak Second Order Monadic Logic of Presburger

Semantics of PMSO.

- interpretation domain : set $\|t\|$ of nodes of a tree $t \in \mathcal{U}(\Sigma)$,
- σ: first order variables $\to \|t\|$,
- ρ: second order variable $\to 2^{\|t\|}$,
- $t, \sigma, \rho \models x = y$ iff $\sigma(x)$ is $\sigma(y)$,
- $t, \sigma, \rho \models x \downarrow y$ iff $\sigma(x)$ is the father of $\sigma(y)$ in t,
- $t, \sigma, \rho \models a(x)$ iff $\sigma(x)$ labeled by a in t,
- $t, \sigma, \rho \models x \in X$ iff $\sigma(x) \in \rho(X)$,
- $t, \sigma, \rho \models x/\phi$ iff $(n_1, \ldots, n_p) \models \phi$ where n_i is the number of sons (in t) of $\sigma(x)$ in $\rho(X_i)$ (with $dom(\rho) = \{X_1, \ldots, X_p\}$),
- $t, \sigma, \rho \models \psi_1 \vee \psi_2$ iff $t, \sigma, \rho \models \psi_1$ or $t, \sigma, \rho \models \psi_2$,
- $t, \sigma, \rho \models \psi_1 \wedge \psi_2$ iff $t, \sigma, \rho \models \psi_1$ and $t, \sigma, \rho \models \psi_2$,
- $t, \sigma, \rho \models \neg\psi$ iff $u, \sigma \not\models \psi$,
- $t, \sigma, \rho \models \exists x\, \psi$ iff there exists $p \in \|t\|$ s.t. $t, \sigma \cup \{p \to x\}, \rho \models \psi$,
- $t, \sigma, \rho \models \exists X \psi$ iff the exists $P \subseteq \|t\|$ s.t. $t, \sigma, \rho \cup \{P \to X\} \models \psi$,

PMSO: Examples

- root:

  $x = \varepsilon \equiv \neg \exists y \, y \downarrow x$
- leaf:

  $\mathsf{leaf}(x) \equiv \neg \exists y \, x \downarrow y$
- $x \downarrow y \equiv \exists Y \; Y = \{y\} \wedge x/_{[Y]=1}$
- prefix ordering = transitive cloture of $\downarrow$:

  $x \downarrow^{*} y \equiv \forall X \left(x \in X \wedge \forall z \forall z' \, (z \in X \wedge z \downarrow z' \Rightarrow z' \in X) \right) \Rightarrow y \in X$

PMSO Languages

Definition : language

The language defined by the closed PMSO formula ψ over Σ is the set of terms $t \in \mathcal{U}(\Sigma)$ s.t. $t \models \psi$.

PMSO Language: example 1

The set of trees of the form $f(a, \ldots, a, b, \ldots, b)$ with the same number of a and b.

PMSO Language: example 1

The set of trees of the form $f(a, \ldots, a, b, \ldots, b)$ with the same number of a and b.

$$
\begin{aligned}
\exists X_a \, \exists X_b \, f(\varepsilon) \quad & \wedge \forall y \, (y \in X_a \Leftrightarrow a(y)) \wedge (y \in X_b \Leftrightarrow b(y)) \\
& \wedge \forall y \; \varepsilon \downarrow y \Rightarrow (\mathsf{leaf}(y) \wedge (y \in X_a \vee y \in X_b)) \\
& \wedge \; \varepsilon /_{[X_a]=[X_b]}
\end{aligned}
$$

PMSO: PA example 2

$\Sigma = \{a, b\}$.
The set of trees of $\mathcal{U}(\Sigma)$ s.t. every internal node is labeled by a and every node labeled by a has at least as much sons without b than sons containing b.

PMSO: PA example 2

$\Sigma = \{a, b\}$.
The set of trees of $\mathcal{U}(\Sigma)$ s.t. every internal node is labeled by a and every node labeled by a has at least as much sons without b than sons containing b.

$$
\begin{array}{rl}
\exists X_a \, \exists X_b \, \forall x & (b(x) \Rightarrow \mathsf{leaf}(x)) \\
\wedge & (a(x) \wedge x/_{[X_a] \geq [X_b] > 0} \Rightarrow x \in X_b) \\
\wedge & (a(x) \wedge x/_{[X_a] \geq [X_b] = 0} \Rightarrow x \in X_a) \\
\wedge & (b(x) \wedge x/_{[X_a] = [X_b] = 0} \Rightarrow x \in X_b)
\end{array}
$$

$Q = Q_{\mathsf{f}} = \{q_a, q_b\}$.
The state q_b accepts the trees containing a b and q_a accepts the others.

$$
\Delta = \left\{ \begin{array}{lclclcl}
a(x_{q_a} \geq x_{q_b} = 0) & \to & q_a & \quad & a(x_{q_a} \geq x_{q_b} > 0) & \to & q_b \\
 & & & & b(x_{q_a} = x_{q_b} = 0) & \to & q_b
\end{array} \right\}
$$

PMSO: Examples of Queries

Base of clients of an online music store, stored in an unordered unranked tree.

A client is a subtree:

- root labeled by `client`
- informations in the sons (purchase, labeled at root by the kind).

Query for clients x who have purchased more jazz than blues:

$$\begin{array}{rll} query_1(x) \equiv & & (\exists X_{\texttt{jazz}}\, \forall y\, y \in X_{\texttt{jazz}} \Leftrightarrow \texttt{jazz}(y)) \\ & \wedge & (\exists X_{\texttt{blues}}\, \forall y\, y \in X_{\texttt{blues}} \Leftrightarrow \texttt{blues}(y)) \\ & \wedge & \texttt{client}(x) \wedge x/_{[X_{\texttt{jazz}}]>[X_{\texttt{blues}}]} \end{array}$$

Query for clients x who have purchased more jazz than anything else:

$$\begin{array}{rll} query_1(x) \equiv & & (\exists X_{\texttt{jazz}}\, \forall y\, y \in X_{\texttt{jazz}} \Leftrightarrow \texttt{jazz}(y)) \\ & \wedge & (\exists X_{\texttt{other}}\, \forall y\, y \in X_{\texttt{other}} \Leftrightarrow \neg\texttt{jazz}(y)) \\ & \wedge & \texttt{client}(x) \wedge x/_{[X_{\texttt{jazz}}]>[X_{\texttt{other}}]} \end{array}$$

PMSO and PA

Theorem

$L \subseteq \mathcal{O}(\Sigma)$ is definable in PMSO iff L is a PA language.

Plan

Presburger Hedge Automata (PHA)

Definition : Presburger Hedge Automata

A *Presburger Hedge Automaton* (PHA) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where $Q = \{q_1, \ldots, q_p\}$ is a finite set of *states*, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $a\big(\bigvee_i (L_i \wedge \phi_i)\big) \to q$ with $a \in \Sigma$, $q \in Q$, $L_i \subseteq Q^*$ regular language and $\phi_i = \phi(x_1, \ldots, x_p)$ is a Presburger formula with a free variable for each state.

for all $w \in Q^*$ we define $w \models L_i \wedge \phi_i$ if $w \in L_i$ and $\#(w) \models \phi_i$.

PHA: Languages

The language $L(\mathcal{A}, q)$ of $\mathcal{A}$ in state $q \in Q$ is the smallest set of ordered unranked trees $a(t_1, \ldots, t_n) \in \mathcal{O}(\Sigma)$ s.t.

- there exists $i_1, \ldots, i_n \leq p$ such that for all $j \leq n$, $t_j \in L(\mathcal{A}, q_{i_j})$,
- there exists a transition $a\big(\bigvee_i (L_i \wedge \phi_i)\big) \to q \in \Delta$ such that $q_{i_1} \ldots q_{i_n} \models \bigvee_i (L_i \wedge \phi_i)$, i.e. there exists i s.t.
  - $q_{i_1} \ldots q_{i_n} \in L_i$,
  - $\#(q_{i_1} \ldots q_{i_n}) \models \phi_i(x_1, \ldots, x_p)$.

The language of $\mathcal{A}$ is $L(\mathcal{A}) = \bigcup_{q \in Q_\mathsf{f}} L(\mathcal{A}, q)$.

PHA: Proprerties

Proposition :

- The class of PHA languages is closed under union and intersection,
- The class of PHA languages is not closed under complementation.

- $\cup$, $\cap$: product
- csq undecidability of the problem of universality.

Proposition :

DPHA $\neq$ NPHA.

PHA: Decision Problems

Lemma :

Given a finite set Q, $L \subseteq Q^*$, regular, and $\phi = \phi(x_1, \ldots, x_p)$ a Presburger formula ($p = |Q|$), it is decidable whether there exists $w \in L$ such that $\#(w) \models \phi$.

Proposition : $\in$

The membership is decidable for PHA.

Proposition : $\emptyset$

The emptiness is decidable for PHA.

Proposition : $\forall$

Universality is undecidable for PHA.

PHA: Logic

Theorem :

A set of trees of $\mathcal{O}(\Sigma)$ is recognizable by a PHA iff it is defined by a PMSO formula of the form $\exists X_1 \ldots \exists X_k \, \phi$ where ϕ is first order.

Corollary :

- EPMSO (existential fragment) is decidable in $\mathcal{O}(\Sigma)$.
- PMSO is undecidable over $\mathcal{O}(\Sigma)$.

Mixed Trees

- $\Sigma = \Sigma_A \cup \Sigma_{AC}$.

$$\begin{array}{rcl} \text{tree} & := & a(\text{hedge}) \mid c(\text{multiset}) \quad (a \in \Sigma_A, c \in \Sigma_{AC}) \\ \text{hedge} & := & \text{tree}, \ldots, \text{tree} \\ \text{multiset} & := & \{\text{tree}, \ldots, \text{tree}\} \end{array}$$

- $c\big(\{t_1, \ldots, t_n\}\big)$ is denoted $c(t_1, \ldots, t_n)$.
- The set of mixed unranked trees over Σ is denoted $\mathcal{M}(\Sigma)$.

Presburger m-Tree Automata (PMA)

Definition : Presburger m-Tree Automata

A *Presburger m-Tree Automaton* (PMA) over an alphabet $\Sigma = \Sigma_A \cup \Sigma_{AC}$ is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where $Q = \{q_1, \ldots, q_p\}$ is a finite set of *states*, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form:

- $a\big(L\big) \to q$ with $a \in \Sigma_A$, $q \in Q$, $L \subseteq Q^*$ is a regular language or
- $c\big(\phi\big) \to q$ with $c \in \Sigma$, $q \in Q$, $\phi = \phi(x_1, \ldots, x_p)$ is a Presburger formula with one free variable for each state.

PMA: Languages

The language $L(\mathcal{A}, q)$ of the PMA $\mathcal{A}$ in state $q \in Q$, is the smallest set of mixed trees

- $a(t_1, \ldots, t_n)$, $a \in \Sigma_A$, such that
  - there exists $i_1, \ldots, i_n \leq p$ with $t_j \in L(\mathcal{A}, q_{i_j})$ for all $j \leq n$,
  - there exists a transition $a(L) \to q \in \Delta$ such that $q_{i_1} \ldots q_{i_n} \in L$,
- or $c(t_1, \ldots, t_n)$, $c \in \Sigma_{AC}$, such that
  - there exists $i_1, \ldots, i_n \leq p$ with $t_j \in L(\mathcal{A}, q_{i_j})$ for all $j \leq n$,
  - there exists a transition $c(\phi) \to q \in \Delta$ such that $\#(q_{i_1} \ldots q_{i_n}) \models \phi(x_1, \ldots, x_p)$.

The language of $\mathcal{A}$ is $L(\mathcal{A}) = \bigcup_{q \in Q_\mathsf{f}} L(\mathcal{A}, q)$.

PMA: Properties and Decision Results

Proposition :

The class of PMA languages is closed under all Boolean operations.

Proposition :

DPMA $\equiv$ NPMA.

Proposition : $\in$

Membership is decidable for PMA.

Proposition : $\emptyset$

Emptiness is decidable for PMA.

Consequences of the analogous results for PHA ($\mathrm{PMA} \subseteq \mathrm{PHA}$).

PMA: Logic

Theorem :

The class of languages of $\mathcal{M}(\Sigma)$ definable by PMSO formulae is the class of PMA languages.

Corollary

PMSO over $\mathcal{M}(\Sigma)$ is decidable.

Plan

Unranked Trees and Reasoning Tasks over XML Documents

Automata for Unranked Ordered Trees

Automata for Unranked Unordered Trees

Automata for Unranked Mixed Trees

Regular Languages modulo Associativity and Commutativity

- Variants of HA
- Regular Languages modulo Associativity
- Regular Languages modulo Associativity and Commutativity

Verification of XML Updates

Imperative Program

[Bouajjani Touili CAV 02]

```
void X() {
  while(true) {
    if Y() {
    thread_create(&t1,Z)
    } else { return }
    }
  }
```

$$
\begin{array}{lcll}
\mathtt{X} & \to & \mathtt{Y}\cdot\mathtt{X} & (r_1)\\
\mathtt{Y} & \to & \mathtt{t} & (r_2)\\
\mathtt{Y} & \to & \mathtt{f} & (r_3)\\
\mathtt{t}\cdot\mathtt{X} & \to & \mathtt{X}\parallel\mathtt{Z} & (r_4)\\
\mathtt{f} & \to & 0 & (r_5)
\end{array}
$$

Reachability analysis:

- The set of reachable terms is regular, but
- we want $\cdot$ Associative,
- and $\parallel$ Associative and Commutative,
- and regular term languages are not closed modulo A and AC.

$\to$ consider unranked trees as representative.

Extensions of HA

Definition : CF-HA

A CF-HA is a tuple $(\Sigma, Q, Q_{\mathsf{f}}, \Delta)$, where Q, Q_{f} are as for HA and the transitions of Δ have the form $a(L) \to q$ with $a \in \Sigma$, $q \in Q$, and $L \subseteq Q^*$ is a context-free language.

Definition : CS-HA

A CS-HA is a tuple $(\Sigma, Q, Q_{\mathsf{f}}, \Delta)$ where Q, Q_{f} are as for HA and the transitions of Δ have the form $a(L) \to q$ with $a \in \Sigma$, $q \in Q$, and $L \subseteq Q^*$ is a context-sensitive language.

CF-HA: Example

$\Sigma = \{a, b, f\}$, language L of trees of $\mathcal{O}(\Sigma)$:

- whose internal nodes are labeled by f,
- with the same number of leaves a than leaves b under every node.

CF-HA: Example

$\Sigma = \{a, b, f\}$, language L of trees of $\mathcal{O}(\Sigma)$:

- whose internal nodes are labeled by f,
- with the same number of leaves a than leaves b under every node.

language of the CF-HA $(Q, Q_{\mathsf{f}}, \Delta)$ with $Q = \{q, q_a, q_b\}$ and

$$\Delta = \{a \to q_a, \quad b \to q_b, \quad f(L) \to q\}$$

L is the language generated by the context-free grammar

$$\begin{array}{lcl} N & := & \varepsilon \;\Big|\; \text{all permutations of } N N_a N_b \;\Big|\; q \\ N_a & := & q_a \quad N_b := q_b \end{array}$$

Rem.: L is not a HA language.

Regular Languages modulo A (A-TA)

Signature $\Sigma = \Sigma_\emptyset \uplus \{a\}$.
The symbols of a is binary and follows the associativity axiom:

$$a(x_1, a(x_2, x_3)) = a(a(x_1, x_2), x_3) \tag{A}$$

Given a TA $\mathcal{B}$ over Σ, we note

$$\mathsf{A}\big(L(\mathcal{B})\big) := \big\{t \in \mathcal{T}(\Sigma) \;\big|\; t \xleftrightarrow[\mathsf{A}]{*} s \in L(\mathcal{B})\big\}$$

(A-TA language)

Proposition :

- the class of regular tree languages is strictly included in the class of A-TA languages.
- The class of A-TA languages is not closed under intersection.

Correspondences $\mathcal{T}(\Sigma) \leftrightarrow \mathcal{O}(\Sigma)$

$\Sigma = \Sigma_\emptyset \uplus \{a\}$ where a is the only associative symbol.

$$\begin{array}{lrcl} \mathit{flat}: & \mathcal{T}(\Sigma) & \to & \mathcal{O}(\Sigma) \\ \mathit{hflat}: & \mathcal{T}(\Sigma)^* & \to & \mathcal{H}(\Sigma) \\ \mathit{flat}^{-1}: & \mathcal{O}(\Sigma) & \to & \mathcal{T}(\Sigma) \end{array}$$

Definitions ($g \in \Sigma_n \setminus \Sigma_\mathsf{A}$):

$$\begin{array}{rcl} \mathit{flat}\big(g(t_1,\ldots,t_n)\big) & = & g(\mathit{flat}(t_1)\ldots\mathit{flat}(t_n)) \\ \mathit{flat}\big(a(t_1,t_2)\big) & = & a(\mathit{hflat}(t_1\,t_2)) \\ \mathit{hflat}\big(g(s_1,\ldots,s_n)\,t_2\ldots t_m\big) & = & \mathit{flat}\big(g(s_1,\ldots,s_n)\big)\,\mathit{hflat}(t_2\ldots t_m) \\ \mathit{hflat}\big(a(s_1,s_2)\,t_2\ldots t_m\big) & = & \mathit{hflat}(s_1 s_2\,t_2\ldots t_m) \\ \mathit{flat}^{-1}\big(g(t_1\ldots t_n)\big) & = & g(\mathit{flat}^{-1}(t_1),\ldots,\mathit{flat}^{-1}(t_n)) \\ \mathit{flat}^{-1}\big(a(t_1\ldots t_m)\big) & = & a(\mathit{flat}^{-1}(t_1), a(\mathit{flat}^{-1}(t_2),\ldots, \\ & & a(\mathit{flat}^{-1}(t_{m-1}),\mathit{flat}^{-1}(t_m)))) \\ & & (m \geq 2) \end{array}$$

A-TA $\leftrightarrow$ CF-HA

Proposition :

A-TA $\equiv$ CF-HA via flattening.

- $\subseteq$ for all TA $\mathcal{A}$ there exists a CF-HA $\mathcal{A}'$ such that $L(\mathcal{A}') = \mathit{flat}\big(L(\mathcal{A})\big) = \mathit{flat}\big(\mathsf{A}(L(\mathcal{A}))\big)$.

TA $\mathcal{A}$	CF-HA $\mathcal{A}'$
$a(q_1, q_2) \to q$	$N_q := N_{q_1} N_{q_2}$, $N_q := q$
$g(q_1, \ldots, q_k) \to q$	$g(q_1 \ldots q_k) \to q$

- $\supseteq$ for all CF-HA $\mathcal{A}'$ there exists a TA $\mathcal{A}$ such that $L(\mathcal{A}) = \mathit{flat}^{-1}\big(L(\mathcal{A}')\big)$ (*i.e.* $\mathit{flat}\big(\mathsf{A}(L(\mathcal{A}))\big) = L(\mathcal{A}')$).

CF-HA $\mathcal{A}'$	TA $\mathcal{A}$
$N := N_1 N_2$	$a(q_{N_1}, q_{N_2}) \to q_N$
$I := N_1 N_2$	$a(q_{N_1}, q_{N_2}) \to q$
$g(q_1 \ldots q_k) \to q$	$g(q_1, \ldots, q_k) \to q$

Generalized Tree Automata (GTA)

Definition : GTA

A *generalized Tree Automata* (GTA) over a signature Σ is a tuple $\mathcal{B} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$ where Q is a finite set of *states*, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form $f(q_1, \ldots, q_n) \to q$ or $a(q_1, q_2) \to a(q_1', q_2')$ with $f \in \Sigma_n$ $(n \geq 0)$ and $q_1, \ldots, q_n, q_1', \ldots, q_n' \in Q$.

For a TA or GTA $\mathcal{B}$, we define the languages:

$$\begin{array}{rcl} L(\mathcal{B}) & := & \{t \in \mathcal{T}(\Sigma) \mid t \xrightarrow[\Delta]{*} q \in Q^{\mathsf{f}}\} \\ L_{\mathsf{A}}(\mathcal{B}) & := & \{t \in \mathcal{T}(\Sigma) \mid t \xrightarrow[\Delta/\mathsf{A}]{*} q \in Q^{\mathsf{f}}\} \end{array}$$

where $\xrightarrow[\Delta/\mathsf{A}]{*}$ is rewriting modulo A: $\xrightarrow[\Delta/\mathsf{A}]{*} := \xleftrightarrow[\mathsf{A}]{*} \circ \xrightarrow[\Delta]{} \circ \xleftrightarrow[\mathsf{A}]{*}$

Languages of TA and GTA modulo A

Proposition :

For all GTA $\mathcal{B}$, there exists a TA $\mathcal{B}'$ such that $L(\mathcal{B}') = L(\mathcal{B})$.

Proposition :

For all TA $\mathcal{B}$, $L_{\mathsf{A}}(\mathcal{B}) = \mathsf{A}\big(L(\mathcal{B})\big)$.

Csq: the emptiness of $L_{\mathsf{A}}(\mathcal{B})$ for a TA $\mathcal{B}$ is decidable.

Proposition :

For a GTA $\mathcal{B}$, in general $L_{\mathsf{A}}(\mathcal{B}) \neq \mathsf{A}\big(L(\mathcal{B})\big)$.

$\Delta_{\mathcal{B}} = \{a \to q_1, b \to q_2, f(q_1, q_2) \to f(q_3, q_3), f(q_3, q_2) \to q_2\}$.
$L(\mathcal{B}, q_2) = \{b\}$
$L_{\mathsf{A}}(\mathcal{B}, q_2) \ni f(f(a, b), b)$

CS-HA $\leftrightarrow$ GTA modulo A

Proposition :

A-GTA $\equiv$ CS-HA via flattening.

- $\subseteq$ for all GTA $\mathcal{A}$ there exists a CS-HA $\mathcal{A}'$ such that $L(\mathcal{A}') = \mathit{flat}\big(L_{\mathsf{A}}(\mathcal{A})\big)$.

GTA $\mathcal{A}$	CS-HA $\mathcal{A}'$
$a(q_1, q_2) \to q$	$N_q := N_{q_1} N_{q_2}$
$a(q_1, q_2) \to a(q_3, q_4)$	$N_{q_3}\, N_{q_4} := N_{q_1}\, N_{q_2}$
	$N_q := q$
$g(q_1, \ldots, q_k) \to q$	$g(q_1 \ldots q_k) \to q$

- $\supseteq$ for all CS-HA $\mathcal{A}'$ there exists a GTA $\mathcal{A}$ such that $L_{\mathsf{A}}(\mathcal{A}) = \mathsf{A}\big(\mathit{flat}^{-1}\big(L(\mathcal{A}')\big)\big)$ (*i.e.* $\mathit{flat}\big(L_{\mathsf{A}}(\mathcal{A})\big) = L(\mathcal{A}')$)

CS-HA $\mathcal{A}'$	TA $\mathcal{A}$
$N := N_1 N_2,\ N \neq I$	$a(q_{N_1}, q_{N_2}) \to q_N$
$N_1\, N_2 := N_3\, N_4$	$a(q_{N_3}, q_{N_4}) \to a(q_{N_1}, q_{N_2})$
$I := N_1 N_2$ for $L_{a,q}$	$a(q_{N_1}, q_{N_2}) \to q$
$g(q_1 \ldots q_k) \to q$	$g(q_1, \ldots, q_k) \to q$

GTA & CS-HA: Decision Results

Proposition : $\in$ CS-HA

The membership problem is PSPACE-complete for CS-HA.

Proposition : $\in$ GTA

The membership problem $t \in L_{\mathsf{A}}(\mathcal{B})$ given a GTA $\mathcal{B}$ is PSPACE-complete.

Proposition : $\emptyset$ CS-HA

The emptiness problem is undecidable for CS-HA.

Proposition : $\emptyset$ GTA

The emptiness problem $L_{\mathsf{A}}(\mathcal{B})$ given a GTA $\mathcal{B}$ is undecidable.

Regular Languages modulo AC (AC-TA)

Signature $\Sigma = \Sigma_\emptyset \uplus \{a\}$.
The symbol a is binary and follows the axioms of associativity and commutativity:

$$a(x_1, a(x_2, x_3)) = a(a(x_1, x_2), x_3) \tag{A}$$

$$a(x_1, x_2)) = a(x_2, x_1) \tag{C}$$

For a TA $\mathcal{B}$ over Σ, we note
$\mathsf{AC}\big(L(\mathcal{B})\big) := \big\{ t \in \mathcal{T}(\Sigma) \;\big|\; t =_{\mathsf{AC}} s \in L(\mathcal{B}) \big\}$ (language of AC-TA)

Proposition :

- the class of regular tree languages is strictly included in the class of AC-TA languages.
- La class of AC-TA languages is closed under Boolean operations.

Correspondence between PA, HA and AC-TA

Proposition :

There is a equivalence, via flattening, between

1. PA
2. CF-HA whose (CF) languages in transitions are closed under permutation
3. AC-TA

pr.:

- $1 \equiv 2$ by Parikh's theorem (equivalence between solutions of Presburger formulas and Parikh projection of CF languages)
- $2 \equiv 3$ with above constructions for A-TA and of previous lecture with flat, flat^{-1}.

Summary

CS-HA $\equiv$ A-GTA | Boolean closures
$\emptyset$ undecidable.

CF-HA $\equiv$ A-TA | no closure under $\cap$ and $\neg$
$\emptyset$ decidable.

HA $\equiv$ TA | Boolean closures
$\emptyset$ decidable.

PA $\equiv$ CF-HA/perm $\equiv$ AC-TA
Boolean closures
$\emptyset$ decidable.

Plan

Unranked Trees and Reasoning Tasks over XML Documents

Automata for Unranked Ordered Trees

Automata for Unranked Unordered Trees

Automata for Unranked Mixed Trees

Regular Languages modulo Associativity and Commutativity

Verification of XML Updates

Verification of XML Transformations

Typechecking XML transformations languages: different models for different languages

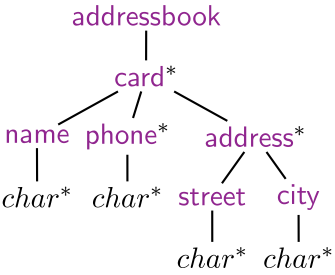
- [Milo Suciu Vianu 03]: k-PTTs
  (XSLT fragment without data joins); backward type inference.
- [Maneth Berlea Perst Seidl 05],
  [Engelfriet Maneth Seidl 09]: language LT and MTTs.
- [Martens Neven 04]: top-down tree transducers.
- [Tozawa 01] backward type inference for XSLT without XPath, with recursive calls (alternating TA).
- [Frish Hozawa 07] backward type inference for MTTs (optimizations).

XQuery Update Facility

W3C recommandation 2011

- [Fundulaki Maneth 04] XACU: model based on the W3C XQuery Update Facility draft.
- [Benedikt Cheney 10] formal model, operational semantics.
- [Bravo Cheney Fundulaki 08] synthesis of schema, verification tool ACCoN.
- [Gardner et al 08] local Hoare reasoning about W3C DOM update library (Context Logic).
- [JR PPDP 10]
  - formal model: parameterized rewrite rules
  - forward/backward type inference, typechecking, reachability verification regular tree model checking
  - verification of access control policies

A DTD and a HA



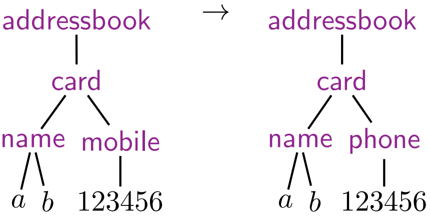
$$
\begin{array}{rcl}
\mathsf{addressbook}(p_{\mathsf{c}}{}^{*}) & \rightarrow & p_{\mathsf{b}} \\
\mathsf{card}(p_{\mathsf{n}}\ p_{\mathsf{h}}{}^{*}\ p_{\mathsf{a}}{}^{*}) & \rightarrow & p_{\mathsf{c}} \\
\mathsf{card}(p_{\mathsf{n}}) & \rightarrow & p_{\mathsf{ec}} \\
\mathsf{name}(p^{*}) & \rightarrow & p_{\mathsf{n}} \\
\mathsf{phone}(p^{*}) & \rightarrow & p_{\mathsf{h}} \\
\mathsf{address}(p_{\mathsf{st}}\ p_{\mathsf{ci}}) & \rightarrow & p_{\mathsf{a}} \\
\mathsf{street}(p^{*}) & \rightarrow & p_{\mathsf{st}} \\
\mathsf{city}(p^{*}) & \rightarrow & p_{\mathsf{ci}} \\
\mathsf{a} & \rightarrow & p \\
\mathsf{b} & \rightarrow & p \\
\mathsf{c} & \rightarrow & p \\
\vdots & &
\end{array}
$$

XUF$_{\text{reg}}$ Rule: Rename

"replace a label mobile by phone"

$$\text{mobile}(x) \to \text{phone}(x)$$

- the variable x stands for a sequence of trees (hedge).
- the rule can be applied to any node labeled by mobile.



$\mathsf{XUF}_{\mathsf{reg}}$ Rule: Insert first

"insert a tree of type p_{ec} (card with only a name) as the first children of addressbook"

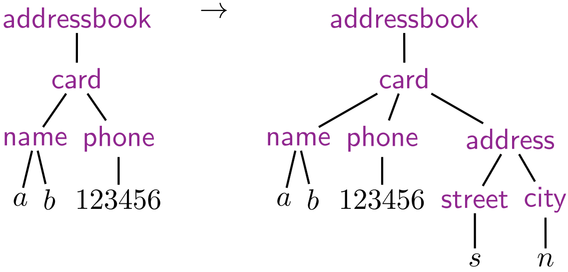
$$\mathsf{addressbook}(x) \to \mathsf{addressbook}(p_{\mathsf{ec}}\ x)$$

- the rewrite rule can be applied to any node labeled by addressbook.
- p_{ec} is a state of a given HA $\mathcal{A}$.
- it stands for an arbitrary tree in the language recognized by $\mathcal{A}$ in state p_{ec}.
- this parametrized rule represents an infinity of rules.
  see [Gilleron 91], [Löding 02, 07].

XUF_{reg} Rule: Insert last

"insert a tree of type p_{a} (address) as the last children of card"

$$\mathsf{card}(x) \to \mathsf{card}(x\ p_{\mathsf{a}})$$



$\mathsf{XUF}_{\mathsf{reg}}$ Rule: Insert into

"insert a tree of type p_{ec} as a children of addressbook"

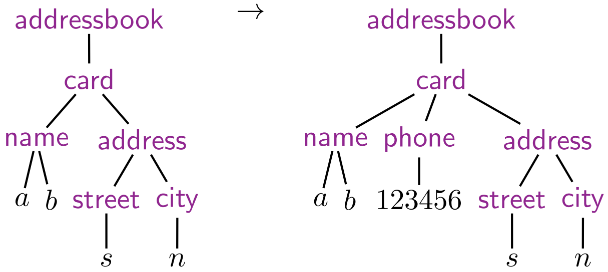
$$\mathsf{addressbook}(x\ y) \to \mathsf{addressbook}(x\ p_{\mathsf{ec}}\ y)$$

- each of the variables x and y stands for an arbitrary sequence of trees (hedge).

XUF_{reg} Rule: Insert before

"insert a tree of type p_{h} (phone) as preceding sibling of address"

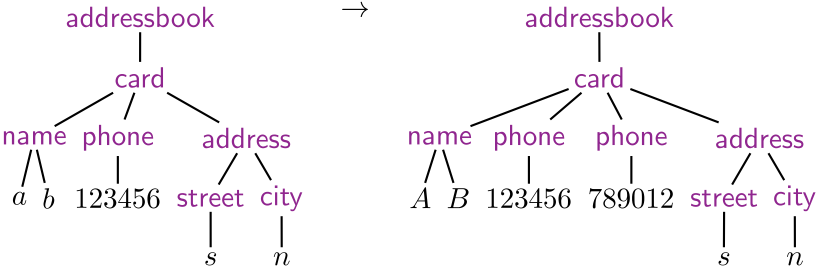
$$\text{address}(x) \rightarrow p_{\mathsf{h}}\ \text{address}(x)$$



XUF$_{\text{reg}}$ Rule: Insert after

"insert a tree of type p_{h} (phone) as following sibling of phone"

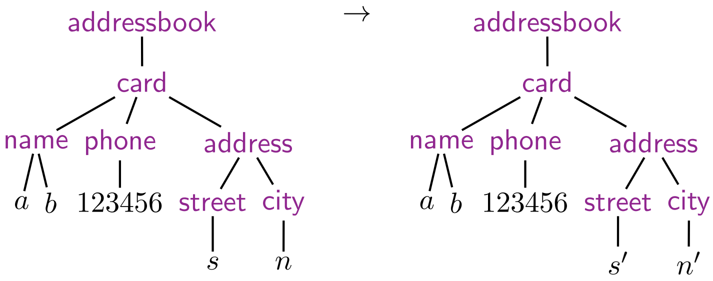
$$\mathsf{phone}(x) \to \mathsf{phone}(x)\ p_{\mathsf{h}}$$



XUF_{reg} Rule: Replace

"replace a subtree (headed by) address by an arbitrary tree of type p_{a}"

$$\text{address}(x) \to p_{\mathsf{a}}$$



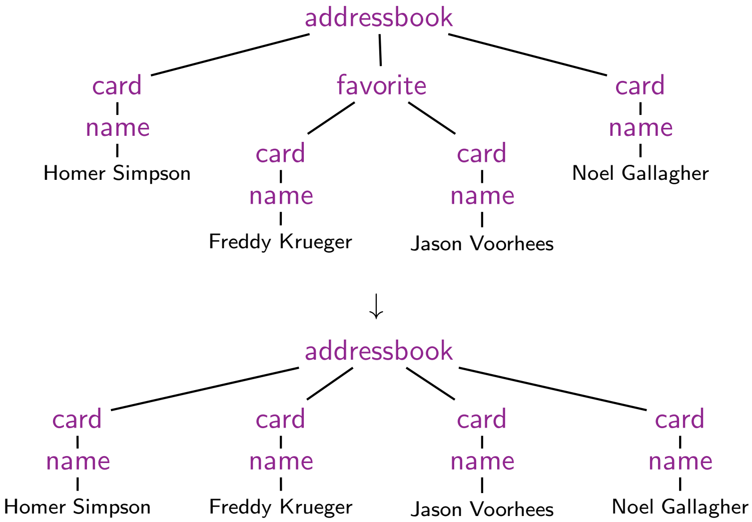
XUF_{reg} Rule: Delete

"delete a whole subtree headed by card"

$$\text{card}(x) \to ()$$

- $()$ is the empty sequence of trees.

Delete single node (not a XQuery Update Primitive)



Extended $\mathrm{XUF}_{\mathrm{reg}}$ Rule: Delete single node

"delete a single node labeled by favorite"

$$\mathsf{favorite}(x) \to x$$

- the trees in the sequence of children x are moved up to the position of the deleted node.
- *collapsing rule*

Extended XUF_{reg} Rule: Multiple Replace

"replace a subtree (headed by) card by an sequence of n trees of respective types $p_1, \ldots, p_n$"

$$\text{card}(x) \to p_1 \ldots p_n$$

- this parametrized rule represents an infinity of rules.
- the right hand sides of these rules are hedges (not trees).

Summary

$\mathsf{XUF}_{\mathsf{reg}}$				XUF			
$a(x)$	$\to$	$b(x)$	REN				
$a(x)$	$\to$	$a(p\,x)$	$\mathsf{INS}_{\mathsf{first}}$	$a(x)$	$\to$	$p\,a(x)$	$\mathsf{INS}_{\mathsf{before}}$
$a(x)$	$\to$	$a(x\,p)$	$\mathsf{INS}_{\mathsf{last}}$	$a(x)$	$\to$	$a(x)\,p$	$\mathsf{INS}_{\mathsf{after}}$
$a(x\,y)$	$\to$	$a(x\,p\,y)$	$\mathsf{INS}_{\mathsf{into}}$				
$a(x)$	$\to$	p	RPL_1	$a(x)$	$\to$	$p_1 \ldots p_n$	RPL
$a(x)$	$\to$	$()$	DEL	$a(x)$	$\to$	x	$\mathsf{DEL}_{\mathsf{s}}$

Forward Type Inference for $\mathsf{XUF}_{\mathsf{reg}}$

Theorem :

For $\mathcal{R}$ in $\mathsf{XUF}_{\mathsf{reg}}$, L_{in} HA language, $\mathcal{R}^*(L_{\mathsf{in}})$ is a HA language.

pr.:

- let $\mathcal{A}_{\mathsf{in}}$ be the HA for L_{in} and $\mathcal{A}$ be the HA parameter of $\mathcal{R}$,
- the HA $\mathcal{A}_{\mathsf{out}}$ is obtained from $\mathcal{A}_{\mathsf{in}} \uplus \mathcal{A}$, by adding transitions to the horizontal NFAs:

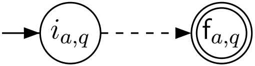
Forward Type Inference for $\mathsf{XUF}_{\mathsf{reg}}$

Theorem :

For $\mathcal{R}$ in $\mathsf{XUF}_{\mathsf{reg}}$, L_{in} HA language, $\mathcal{R}^*(L_{\mathsf{in}})$ is a HA language.

pr.:

- let $\mathcal{A}_{\mathsf{in}}$ be the HA for L_{in} and $\mathcal{A}$ be the HA parameter of $\mathcal{R}$,
- the HA $\mathcal{A}_{\mathsf{out}}$ is obtained from $\mathcal{A}_{\mathsf{in}} \uplus \mathcal{A}$, by adding transitions to the horizontal NFAs:

case $\mathsf{INS}_{\mathsf{first}}$: $a(x) \to a(p\,x) \in \mathcal{R}$, $a\Bigg($ 

$\Bigg) \to q$

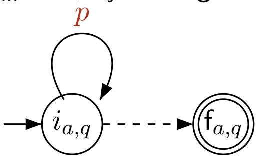
Forward Type Inference for $\mathsf{XUF}_{\mathsf{reg}}$

Theorem :

For $\mathcal{R}$ in $\mathsf{XUF}_{\mathsf{reg}}$, L_{in} HA language, $\mathcal{R}^*(L_{\mathsf{in}})$ is a HA language.

pr.:

- let $\mathcal{A}_{\mathsf{in}}$ be the HA for L_{in} and $\mathcal{A}$ be the HA parameter of $\mathcal{R}$,
- the HA $\mathcal{A}_{\mathsf{out}}$ is obtained from $\mathcal{A}_{\mathsf{in}} \uplus \mathcal{A}$, by adding transitions to the horizontal NFAs:

case $\mathsf{INS}_{\mathsf{first}}$: $a(x) \to a(p\,x) \in \mathcal{R}$, $a\Big($ 

$\Big) \to q$

Forward Type Inference for $\mathsf{XUF}_{\mathsf{reg}}$

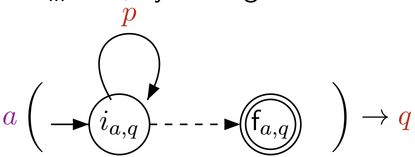
Theorem :

For $\mathcal{R}$ in $\mathsf{XUF}_{\mathsf{reg}}$, L_{in} HA language, $\mathcal{R}^*(L_{\mathsf{in}})$ is a HA language.

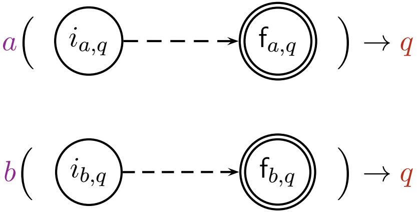
pr.:

- let $\mathcal{A}_{\mathsf{in}}$ be the HA for L_{in} and $\mathcal{A}$ be the HA parameter of $\mathcal{R}$,
- the HA $\mathcal{A}_{\mathsf{out}}$ is obtained from $\mathcal{A}_{\mathsf{in}} \uplus \mathcal{A}$, by adding transitions to the horizontal NFAs:

case $\mathsf{INS}_{\mathsf{first}}$: $a(x) \to a(p\,x) \in \mathcal{R}$,



case REN: $a(x) \to b(x) \in \mathcal{R}$



Forward Type Inference for $\mathrm{XUF}_{\mathrm{reg}}$

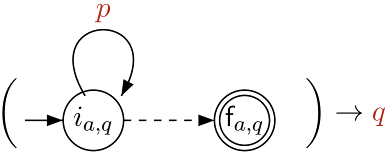
Theorem :

For $\mathcal{R}$ in $\mathrm{XUF}_{\mathrm{reg}}$, L_{in} HA language, $\mathcal{R}^*(L_{\mathrm{in}})$ is a HA language.

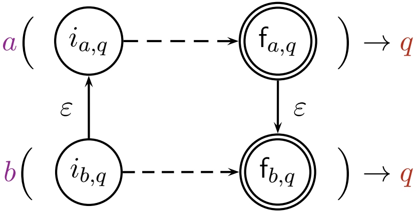
pr.:

- let $\mathcal{A}_{\mathrm{in}}$ be the HA for L_{in} and $\mathcal{A}$ be the HA parameter of $\mathcal{R}$,
- the HA $\mathcal{A}_{\mathrm{out}}$ is obtained from $\mathcal{A}_{\mathrm{in}} \uplus \mathcal{A}$, by adding transitions to the horizontal NFAs:

case $\mathrm{INS}_{\mathrm{first}}$: $a(x) \to a(p\,x) \in \mathcal{R}$, a



case REN: $a(x) \to b(x) \in \mathcal{R}$

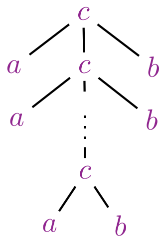


Forward Type Inference for XUF

Lemma

For some $\mathcal{R}$ *in* XUF *and* L_{in} HA *language,* $\mathcal{R}^*(L_{\mathsf{in}}) \notin$ HA.

example: $\mathcal{R} = \{c(x) \to x\}$ (one DEL_{s}) and $L_{\mathsf{in}} =$



$$\mathcal{R}^*(L_{\mathsf{in}}) \cap c\big(\{a,b\}^*\big) = \{c(a^n b^n) \mid n \geq 0\}$$

Forward Type Inference for XUF

Theorem :

For $\mathcal{R}$ in XUF, L_{in} a CF-HA language, $\mathcal{R}^*(L_{\text{in}})$ is a CF-HA language, with a PTIME, polynomial size construction.

pr.: similar as XUF_{reg} for for REN, $\text{INS}_{\text{first}}$, INS_{last}, INS_{into}.

- REN: if $a(x) \to b(x) \in \mathcal{R}$, then add $I'_{b,q} := I'_{a,q}$
- $\text{INS}_{\text{first}}$: if $a(x) \to b(p\,x) \in \mathcal{R}$, then add $I'_{b,q} := p\,I'_{a,q}$

add collapsing transitions for $\text{INS}_{\text{before}}$, $\text{INS}_{\text{after}}$, RPL, DEL, DEL_{s}.

- RPL: if $a(x) \to p_1 \ldots p_n \in \mathcal{R}$ and $a(L) \to q$ transition, $L \neq \emptyset$, then add the collapsing transition $p_1 \ldots p_n \to q$.

Backward Type Inference for XUF

Theorem :

For $\mathcal{R}$ in XUF, L_{out} a HA language, $(\mathcal{R}^{-1})^*(L_{\mathsf{out}})$ is a HA language, with a EXPTIME, exponential size construction.

pr.: tree automata completion; construction of a finite sequence of HA

$$\mathcal{A}_{\mathsf{out}} = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n = (\mathcal{R}^{-1})^*(\mathcal{A}_{\mathsf{out}})$$

by addition of new transitions.

Applications

Theorem :

For $\mathcal{R}$ in XUF, L_{in} a CF-HA language, $\mathcal{R}^*(L_{\text{in}})$ is a CF-HA language, with a PTIME, polynomial size construction.

Consequences:

- reachability for XUF
  - decidable in PTIME.
- typechecking XUF
  given two HA for L_{in}, L_{out} and $\mathcal{R}$ in XUF, $\mathcal{R}^*(L_{\text{in}}) \subseteq L_{\text{out}}$?
  - EXPTIME-complete
  - PTIME if L_{out} is given by a deterministic and complete HA.
- type synthesis:
  - given $\mathcal{R}$ in XUF and an input type L_{in},
    a CF-HA for L_{out} is constructed in PTIME.
  - given $\mathcal{R}$ in XUF and an output type L_{out},
    a HA for L_{in} is constructed in EXPTIME.

Rule Based Access Control Policies

Definition : [Fundulaki Maneth][Bravo et al, ACCOn]

An access control policy (ACP) is given by two finite sets of rules

- $\mathcal{R}_+$: authorized operations of XUF
- $\mathcal{R}_-$: forbidden operations of XUF

example

- $$\mathcal{R}_+ = \left\{ \begin{array}{rcl} \mathsf{addressbook}(x) & \rightarrow & \mathsf{addressbook}(p_{\mathsf{ec}}\ x), \\ \mathsf{card}(x) & \rightarrow & () \end{array} \right\}$$
  - user can insert card with name, delete card.
- $\mathcal{R}_- = \{\mathsf{name}(x) \rightarrow p_{\mathsf{n}}\}$
  - user cannot change a name.

Inconsistency

Inconsistency

An ACP $\langle \mathcal{R}_+, \mathcal{R}_- \rangle$ is inconsistent if one rule of $\mathcal{R}_-$ can be simulated through a sequence of rules of $\mathcal{R}_+$

example: changing name in a card is simulated by deleting and then inserting.

Theorem : [Fundulaki Maneth 04] [Moore 09]

Inconsistency is undecidable for $\mathsf{XUF}_{\mathsf{reg}}$.

Local Inconsistency

Definition : Local Inconsistency

An ACP $\langle \mathcal{R}_+, \mathcal{R}_- \rangle$ is locally inconsistent for t
if there exists u such that $t \xrightarrow[\mathcal{R}_-]{} u$ and $t \xrightarrow[\mathcal{R}_+]{*} u$.

Theorem : [JR PPDP 10]

Local inconsistency is decidable in PTIME for XUF.

pr.:

- compute a HA recognizing $L_- = \{u \mid t \xrightarrow[\mathcal{R}_-]{} u\}$
- compute a CF-HA recognizing $L_+ = \mathcal{R}_+^*\big(\{t\}\big)$
- check that $L_- \cap L_+ \neq \emptyset$.

Summary

- model for Xquery Update facility primitives (and extensions) as parameterized rewrite rules for unranked trees
- results of forward/backward type inference (rewrite closure)
- reachability decision
- decision of local inconsistency of ACPs

Extensions

- rules controlled with downward regular XPath expressions.
- better model: generalize $\mathcal{R}^*$ to $\mathcal{R}^e$ (e regular expression over rule names).
- unranked unordered trees (Presburger Automata).

Part IV

Tree Automata Defined as Sets of Horn Clauses

Plan

Definition of tree automata as set of first order (universal) clauses.
Languages = Herbrand models.

+ uniform formalism for the definition of several classes of automata (alternating, 2-ways, with constraints...)
+ enables the use of techniques and tools from automatic deduction in order to solve the classical decision problems
− complexity
− not easy to analyse the history
(and construct a witness, a run...)

Clauses: Syntax

- terms in $\mathcal{T}(\Sigma, \mathcal{X})$ over signature Σ (Σ_n: symbol of arity n)
- finite set $\mathcal{P}$ of predicate symbols P, Q,... (notation $\mathcal{P}_n$)
  basically we will only considerer predicates of arity 0 or 1.
- literals positive: $P(t)$, denoted $+P(t)$
  negative: $\neg P(t)$, denoted $-P(t)$
- clause: disjunction of literals $\pm P_1(t_1) \vee \ldots \vee \pm P_k(t_k)$
  empty clause ($k = 0$), denoted $\perp$.
- Horn clause: at most one positive literal
  $-P_1(t_1) \vee \ldots \vee -P_k(t_k) \vee +P(t)$, denoted
  $P_1(t_1), \ldots, P_k(t_k) \Rightarrow P(t)$.
- goal $=$ negative clause
  $-P_1(t_1) \vee \ldots \vee -P_k(t_k)$, denoted $P_1(t_1), \ldots, P_k(t_k) \Rightarrow \perp$.

Herbrand Models

- a Herbrand structure $\mathcal{H}$ has domain $\mathcal{T}(\Sigma)$ and fonctions $f^{\mathcal{H}}(t_1, \ldots, t_n) := f(t_1, \ldots, t_n)$ (ground term with the symbol f at the root).
- $\mathcal{H}$ is completely defined by the set of ground atoms $P(t)$ such that $\mathcal{H} \models P(t)$.

Theorem :

A set of clauses S is satisfiable iff it admits a Herbrand model.

Theorem :

Every satisfiable set S of Horn clauses admits a smallest (wrt inclusion) Herbrand model $\mathcal{H}_S$.

Smallest Herbrand Models

Theorem :

Every satisfiable set S of Horn clauses admits a smallest (wrt inclusion) Herbrand model $\mathcal{H}_S$.

A set of Horn clauses S defines the following operator T_S over sets of ground atoms

$$\begin{array}{rcl} T_S(L) & = & \left\{ P(t\sigma) \;\middle|\; \begin{array}{l} t\sigma\,\text{ground}, P_1(t_1),\ldots,P_n(t_n) \Rightarrow P(t) \in S, \\ P_1(t_1\sigma),\ldots,P_n(t_n\sigma) \in L \end{array} \right\} \\ & \cup & \{\bot\}\ \text{if}\ \begin{array}[t]{l} P_1(t_1),\ldots,P_n(t_n) \Rightarrow \bot \in S, \\ P_1(t_1\sigma),\ldots,P_n(t_n\sigma) \in L \end{array} \end{array}$$

The smallest fixpoint of T_S is $\bigcup_{n\geq 1} T_S^n(\emptyset)$,

- if it contains $\bot$, then S is not satisfiable,
- otherwise, it is the smallest Herbrand model of S.

Languages and Automata

The language of a satisfiable set S of Horn clauses for a predicate Q is:

$$L(S, Q) = \{t \mid Q(t) \in \mathcal{H}_S\}$$

Let $\mathcal{A} = (\Sigma, \{q_1, \ldots, q_k\}, F, \Delta)$ be a bottom-up tree automaton.
Let $\mathcal{P} = \{Q_1, \ldots, Q_k\}$ be a set of unary predicates.
We associate to $\mathcal{A}$ the (satisfiable) set of Horn clauses

$$S_{\mathcal{A}} := \left\{ \begin{array}{r} Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \\ \mid f(q_1, \ldots, q_n) \to q \in \Delta \end{array} \right\}$$

Lemma :

For all state q, $L(\mathcal{A}, q) = L(S_{\mathcal{A}}, Q)$.

Clauses/Classes of Automata

clauses of standard automate ($x_1, \ldots, x_n$ pairwise distinct)

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \tag{reg}$$

ε-transitions

$$Q_1(x) \Rightarrow Q(x) \tag{ε}$$

alternating clauses

$$Q_1(x), \ldots, Q_n(x) \Rightarrow Q(x) \tag{alt}$$

2-ways (bidirectional) clauses ($x_1, \ldots, x_n$ pairwise distinct)

$$Q\big(f(x_1, \ldots, x_n)\big) \Rightarrow Q_i(x_i) \tag{bidi}$$

Plan

Definition

Saturation Results

Tree Automata with Equality Constraints

Tree Automata with One Memory

Decision Problems, Satisfiability

Let S be a satisfiable set of Horn clauses and let Q be a predicate.

- membership: the ground term $t \in L(S, Q)$ iff $S \cup \{Q(t) \Rightarrow \bot\}$ is unsatisfiable
- emptiness: $L(S, Q) \neq \emptyset$ iff $S \cup \{Q(x) \Rightarrow \bot\}$ is unsatisfiable.
- membership of instance: there exists σ such that $t\sigma \in L(S, Q)$ iff $S \cup \{Q(t) \Rightarrow \bot\}$ is unsatisfiable.
- emptiness of intersection: $L(S, Q_1) \cap \ldots \cap L(S, Q_p) \neq \emptyset$ iff $S \cup \{Q_1(x), \ldots, Q_p(x) \Rightarrow \bot\}$ is unsatisfiable.

$\Rightarrow$ we are interested in automated deduction techniques for deciding the satisfiability when S represents an automaton.

Resolution

Clauses:

$$\frac{C \vee +Q(s) \quad -Q(t) \vee D}{C\sigma \vee D\sigma}$$

where σ is the most general unifier (*mgu*) of s and t.

Horn clauses:

$$\frac{P_1(s_1), \ldots, P_m(s_m) \Rightarrow Q_1(s) \quad Q_1(t_1), Q_2(t_2), \ldots, Q_n(t_n) \Rightarrow Q(t)}{P_1(s_1\sigma), \ldots, P_m(s_m\sigma), Q_2(t_2\sigma), \ldots, Q_n(t_n\sigma) \Rightarrow Q(t\sigma)}$$

where σ is the *mgu* of s and t_1.

Theorem : correction, completeness

A set S of Horn clauses is unsatisfiable iff one can derive $\perp$ by resolution starting from S.

Terminaison of Resolution

The application of the resolution rule to automata clauses (reg) does not terminate.

$$\frac{P_1(x_1), P_2(x_2) \Rightarrow \boxed{Q_1(g(x_1, x_2))} \qquad \boxed{Q_1(y_1)}, Q_2(y_2) \Rightarrow Q(f(y_1, y_2))}{P_1(x_1), P_2(x_2), Q_2(y_2) \Rightarrow Q\big(f(g(x_1, x_2), y_2)\big)}$$

Complete Strategies for Resolution

$$\frac{\overset{(C)}{P_1(s_1), \ldots, P_m(s_m) \Rightarrow \boxed{Q_1(s)}} \qquad \overset{(D)}{\boxed{Q_1(t_1)}, \ldots, Q_n(t_n) \Rightarrow Q(t)}}{P_1(s_1\sigma), \ldots, P_m(s_m\sigma), Q_2(t_2\sigma), \ldots, Q_n(t_n\sigma) \Rightarrow Q(t\sigma)}$$

ordered resolution for $\succ$:

- $Q_1(s)$ maximal for $\succ$ in C,
- $Q_1(t_1)$ maximal for $\succ$ in D.

ordered resolution with selection :

selection function : clause $\mapsto$ subset of negative literals.

- no literal is selected in C,
- $Q_1(s)$ is maximal for $\succ$ in C,
- $Q_1(t_1)$ is selected in D or
- no literal is selected in D and $Q_1(t)$ is maximal in D.

Completeness of Ordered Resolution

Theorem :

Ordered resolution with selection is complete for Horn clauses.

Starting from any unsatisfiable set S, we shall derive $\perp$.

Choice of an Ordered Strategy with Selection

- ordering $\succ$ s.t. $P(s) \succ Q(t)$ iff $s > t$ for the subterm ordering $>$.
- selection function sel:
  - negative literals $-Q(t)$ where t is not a variable.

Lemma :

Every tree automaton (finite set clauses of type (reg)) is saturated under resolution ordered by $\succ$.

Resolution ordered by $\succ$ and with selection by sel cannot be applied between automata clauses (reg) like in

$$\frac{P_1(x_1), \ldots, P_m(x_m) \Rightarrow Q_1(g(\overline{x})) \qquad Q_1(y_1), \ldots, Q_n(y_n) \Rightarrow Q(f(\overline{y}))}{P_1(x_1), \ldots, P_m(x_m), Q_2(y_2), \ldots, Q_n(y_n) \Rightarrow Q\big(f(g(\overline{x}), y_2, \ldots, y_n)\big)}$$

because no literal are selected in the clauses and $Q_1(y_1)$ is not maximal in $\{Q_1(y_1), \ldots, Q_n(y_n), Q(f(\overline{y}))\}$ $(Q(f(\overline{y})) \succ Q_1(y_1))$.

Transformation of Alternating Automata (reg + alt = reg)

Alternating automata = finite set of clauses

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(reg)}$$

and

$$Q_1(x), \ldots, Q_n(x) \Rightarrow Q(x) \qquad \text{(alt)}$$

Proposition :

Given an alternating tree automaton $\mathcal{A}$ over Σ, we can construct in exponential time a deterministic bottom-up tree automaton $\mathcal{A}'$ recognizing the same language.

Construction by application of ordered resolution with selection, following an appropriated strategy.

Transformation of Alternating Automata

- start from a set $\mathcal{A}$ of clauses of the form (reg) and (alt).
- saturate with resolution ordered (by $\succ$) with selection (by sel).
- all the clauses produced belong to a type containing an exponential number of clauses

$$Q_1(x_{i_1}), \ldots, Q_k(x_{i_k}), \underline{Q'_1(f(\overline{x}))}, \ldots, \underline{Q'_m(f(\overline{x}))} \Rightarrow Q(f(\overline{x})) \tag{f}$$

- hence saturation terminates with a set $\mathcal{A}''$
- the application of resolution to $\mathcal{A}'' \cup \{Q(t) \Rightarrow \bot\}$ (for a ground term t) only involves clauses of the form (reg).
- hence, for all Q, $L(\mathcal{A}'', Q) = L(\mathcal{A}''|_{\mathsf{reg}}, Q)$.

Transformation of Bidirectional Alternating Automata (reg + alt + bidi = reg)

Bidirectional (2-way) alternating automata = finite set of clauses

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(reg)}$$

and

$$Q_1(x), \ldots, Q_n(x) \Rightarrow Q(x) \qquad \text{(alt)}$$

and

$$Q\big(f(x_1, \ldots, x_n)\big) \Rightarrow Q_i(x_i) \qquad \text{(bidi)}$$

Proposition :

Given a bidirectional alternating tree automaton $\mathcal{A}$ over Σ, we can construct in exponential time a bottom-up deterministic tree automaton $\mathcal{A}'$ recognizing the same language.

same principle as for alternating tree automata, with

- other ordering and selection function for defining the resolution strategy,
- and a new rule called ε-splitting.

Decision of Instance Membership

tree automaton

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(reg)}$$

$+$ query

$$Q(t) \Rightarrow \bot$$

Resolution ordered by $\succ$ with selection by sel and ε-splitting terminates. invariant: the resolution only produces clauses of the following 2 types:

$$P_1(s_1), \ldots, P_m(s_m), q_1, \ldots, q_k \Rightarrow [\, q \,] \qquad \text{(gs)}$$

where $m, k \geq 0$, and $s_1, \ldots, s_m$ are subterms of t.

$$P_1(y_{i_1}), \ldots, P_k(y_{i_k}), \underline{P'_1(f(y_1, \ldots, y_n))}, \ldots, \underline{P'_m(f(\overline{y}))} \Rightarrow [\, q \,] \quad \text{(gf)}$$

where $k, m \geq 0$, $k + m > 0$, $i_1, \ldots, i_k \leq n$, and $y_1, \ldots, y_n$ distinct.

Plan

Testing Equalities between Brother Subterms

In standard tree automata clauses, the variables $x_1, \ldots, x_n$ are pairwise distinct

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(reg)}$$

With variable sharing in

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(brother)}$$

we force equalities between brother subterms.

example:

$$\begin{array}{rcl} & \Rightarrow & Q(a), \\ Q(x_1), Q(x_2) & \Rightarrow & Q\big(f(x_1, x_2)\big), \\ Q(x), Q(x) & \Rightarrow & Q_{\mathsf{f}}\big(f(x, x)\big) \end{array}$$

Testing Equality between Brother Subterms

Tree automata with equality tests between brother subterms are strictly more expresive than bottom-up tree automata.

Theorem :

The emptiness problem is EXPTIME-complete for tree automata with equality tests between brother subterms.

Testing Equalities and Disequalities between Brother Subterms

[Bogaert Tison STACS 1992]: tests of $=$ and $\neq$ between brother subterms (see chapter 4 TATA book).

- determinizable in exponentiel time
- all Boolean closures
- emptiness decidable in PTIME for deterministic
- emptiness EXPTIME-complete for non-deterministic

[Reuss Seidl 2010]: clausal presentation with $\neq$.

Testing Arbitrary Equalities

One can perform more general tests with clauses with equalities

$$Q_1(x_1), \ldots, Q_n(x_n), u_1 = v_1, \ldots, u_k = v_k \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \tag{test}$$

where $k \geq 0$, $u_1, v_1, \ldots, u_k, v_k \in \mathcal{T}\big(\Sigma, \{x_1, \ldots, x_n\}\big)$.

Without restrictions, emptiness is undecidable.

Arbitrary Equality Tests: Decidable Class

[JRV JLAP 08]
We distinguish some predicates call test predicates, and assume a partial ordering $\succ$ over predicates such that $Q \succ Q_0$ for all Q, test predicate and Q_0 non-test.

$$Q_1(x_1), \ldots, Q_n(x_n), u_1 = v_1, \ldots, u_k = v_k \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \quad \text{(test)}$$

where Q is a test predicate, and for all $i \leq n$, $Q \succ Q_i$

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \quad \text{(reg')}$$

where either all $Q, Q_1, \ldots, Q_n$ are not test predicates, or Q is a test predicate and at most one $Q_i = Q$ and the others are not test.

Arbitrary equality tests: decidable class

example: stuttering lists

$$
\begin{array}{rcl}
 & \Rightarrow & Q_0(0) \\
 & \Rightarrow & Q_1(\mathsf{nil}) \\
Q_0(x) & \Rightarrow & Q_0(s(x)) \\
Q_0(x), Q_1(y) & \Rightarrow & Q_1(\mathsf{cons}(x, y)) \\
Q_0(x), Q_2(y) & \Rightarrow & Q_2(\mathsf{cons}(x, y))
\end{array}
$$

$$Q_0(x), Q_1(y), y = \mathsf{cons}(x, y') \Rightarrow Q_2(\mathsf{cons}(x, y))$$

Arbitrary Equality Tests: Decidable Class

Theorem :

The satisfiability of a set of clauses of type (test) and (reg') and a goal clause $Q(t) \Rightarrow \bot$ is decidable.

pr.: Saturation by ordered paramodulation with selection and ϵ-splitting.

Extension to langages (and equality tests) modulo equational theories, by adding clauses

$$\Rightarrow \ell = x$$

of a restricted form.

Example:

$$\mathsf{car}(\mathsf{cons}(x, y)) = x, \mathsf{cdr}(\mathsf{cons}(x, y)) = y, \mathsf{cons}\big(\mathsf{car}(y), \mathsf{cdr}(y)\big) = y$$

Plan

Definition

Saturation Results

Tree Automata with Equality Constraints

Tree Automata with One Memory

Pushdown Tree Automata with One Tree Memory

[Guessarian 83, Schimpf Gallier 85, Coquidé et al 94]

- Σ: input signature; on input: terms of $\mathcal{T}(\Sigma)$
- Γ: stack alphabet; the auxiliary memory is a stack of Γ^*

$$Q_1(x_1, s_1), \ldots, Q_n(x_n, s_n) \Rightarrow Q\big(f(\overline{x}), h(y_1, \ldots, y_m)\big) \qquad \text{(read)}$$

where $f \in \Sigma$, $h \in \Gamma$, $s_1, \ldots, s_n \in \mathcal{T}(\Gamma, \{y_1, \ldots, y_m\})$

$$Q_1(x, s) \Rightarrow Q\big(x, h(y_1, \ldots, y_m)\big) \qquad \text{(pda-}\varepsilon\text{)}$$

where $h \in \Gamma$, $s \in \mathcal{T}(\Gamma, \{y_1, \ldots, y_m\})$.

Same expressiveness as context-free tree grammars

$$X(x_1, \ldots, x_n) \rightarrow r$$

where $X \in \mathbb{N}$, non-terminal of arity n, $x_1, \ldots, x_n$ distinct variables, $r \in \mathcal{T}\big(\Sigma \cup \mathbb{N}, \{x_1, \ldots, x_n\}\big)$

Tree Automata with One Memory

[Comon Cortier 05]

$$Q_1(x_1, y_1), Q_2(x_2, y_2) \Rightarrow Q\big(f(x_1, x_2), h(y_1, y_2)\big) \qquad \text{(push)}$$

$$Q_1\big(x_1, h(y_{11}, y_{12})\big), Q_2(x_2, y_2) \Rightarrow Q\big(f(x_1, x_2), y_{11}\big) \qquad (\mathsf{pop}_{11})$$
$$Q_1(x_1, \perp), Q_2(x_2, y_2) \Rightarrow Q\big(f(x_1, x_2), \perp\big)$$

$\ldots$ (pop_{12}), (pop_{21}), (pop_{22})

$$\Rightarrow Q(a, \perp) \qquad (\mathsf{int}_0)$$

$$Q_1(x_1, y_1), Q_2(x_2, y_2) \Rightarrow Q\big(f(x_1, x_2), y_1\big) \qquad (\mathsf{int}_1)$$

$$Q_1(x_1, y_1), Q_2(x_2, y_2) \Rightarrow Q\big(f(x_1, x_2), y_2\big) \qquad (\mathsf{int}_2)$$

where $h \in \Gamma_2$, $a \in \Sigma_0$, $f \in \Sigma_2$.

Tree Automata with One Memory: Languages and Closure

(input) Language

$$L(\mathcal{A}, Q) = \{t \mid Q(t, s) \in \mathcal{H}_S\}$$

Memory Language

$$M(\mathcal{A}, Q) = \{s \mid Q(t, s) \in \mathcal{H}_S\}$$

Languages of tree automata with one memory are closed under $\cup$ but not under $\cap$ or $\neg$.

[CJP FOSSACS 07] addition of a visibly condition:

- $\Sigma = \Sigma_{push} \uplus \Sigma_{pop_{11}} \uplus \ldots \uplus \Sigma_{int_0} \uplus \Sigma_{int_1} \uplus \Sigma_{int_2}$
- the input symbol determines the possible operations on memory
- full Boolean closure

Tree Automata with One Memory: Languages and Closure

Theorem :

The emptiness problem is decidable in polynomial time for tree automata with one memory.

Lemma :

For all $\mathcal{A}$, Q, $L(\mathcal{A}, Q) = \emptyset$ iff $M(\mathcal{A}, Q) = \emptyset$.

- For all $\mathcal{A}$, Q, $M(\mathcal{A}, q)$ is a language of bidirectional alternating automata (clauses reg + alt + bidi = reg).
- definition by "*projection*" of clauses on second component of states.
- actually $M(\mathcal{A}, q)$ in a smaller class $\mathcal{H}_3$ [Nielson Seidl SAS 02] decidable in cubic time.

Tree Automata with One Memory and Contraints

extension with constraints testing the content of the memory.

$$Q_1(x_1, y_1), Q_2(x_2, y_2), y_1 = y_2 \Rightarrow Q\big(f(x_1, x_2), y_1\big) \qquad (\mathsf{int}_1^=)$$

$$Q_1(x_1, y_1), Q_2(x_2, y_2), y_1 = y_2 \Rightarrow Q\big(f(x_1, x_2), y_2\big) \qquad (\mathsf{int}_2^=)$$

also $\neq$ tests in the model of [CJP FOSSACS 07].

- emptiness is decidable
- Boolean closure with visibly condition and restriction of constraints to structural equality.

TA1M for Verification

Tree automata with one memory and constraints can recognize the following data-structures

- balanced binary trees
- powerlists
  (description and verification of data parallel algorithms)
- red-black trees (binary search trees)
  1. every node is black or red
  2. the root is black
  3. all the leaves are black
  4. the 2 children of a red node are black
  5. all paths have the same number of black nodes