Tree Automata and Symbolic Constraints Solving

Florent Jacquemard

IRCAM & INRIA
florent.jacquemard@ircam.fr

Automata are computation models

a b c d e

accepted

not

Tree automata are computation models

Tree automata for reasoning tasks

big
tree
set

inference

decision
procedure

answer

(finite model)

Automata as Finite Representation of Infinite Tree Sets

tree set S	problem
reachable configurations of a system/program	verification (infinite state model checking) $\mathsf{err} \in S$? $S \cap S_{\mathsf{err}} = \varnothing$?
forward/backward closure of a set by a transformation	type checking
set of solutions	symbolic constraint solving consistency $S = \varnothing$? (i.e. $\exists$ solution?)

Reachability Analysis of Programs

Concurrent readers/writers

Example from [Clavel et al. 07 LNCS 4350]

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

(1) writers can access the file if nobody else is accessing it

(2) readers can access the file if no writer is accessing it

(3,4) readers and writers can leave the file at any time

Properties expected:

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

Concurrent readers/writers: reachable configurations

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

initial configuration:

$\mathsf{state}(0,0)$

Concurrent readers/writers: reachable configurations

1. $\mathsf{state}(0, 0) \to \mathsf{state}(0, s(0))$
2. $\mathsf{state}(r, 0) \to \mathsf{state}(s(r), 0)$
3. $\mathsf{state}(r, s(w)) \to \mathsf{state}(r, w)$
4. $\mathsf{state}(s(r), w) \to \mathsf{state}(r, w)$

reachable configurations:

$\mathsf{state}(0, 0)$

Concurrent readers/writers: reachable configurations

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

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

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

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: reachable configurations

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

reachable configurations:

$\mathsf{state}(0,0)$

tree automaton:

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

Concurrent readers/writers: reachable configurations

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

reachable configurations:

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

tree automaton:

$$\begin{array}{rcl} 0 & \to & q_0 \\ \mathsf{state}\big(q_0,q_0\big) & \to & q \\ s\big(q_0\big) & \to & q_1 \\ \mathsf{state}\big(q_0,q_1\big) & \to & q \end{array}$$

Concurrent readers/writers: reachable configurations

1. $\mathsf{state}(0,0) \to \mathsf{state}(0,s(0))$
2. $\mathsf{state}(r,0) \to \mathsf{state}(s(r),0)$
3. $\mathsf{state}(r,s(w)) \to \mathsf{state}(r,w)$
4. $\mathsf{state}(s(r),w) \to \mathsf{state}(r,w)$

reachable configurations:

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

tree automaton:

$$\begin{aligned}
0 &\to q_0 \\
\mathsf{state}(q_0,q_0) &\to q \\
s(q_0) &\to q_1 \\
\mathsf{state}(q_0,q_1) &\to q \\
\mathsf{state}(q_1,q_0) &\to q \\
s(q_1) &\to q_2 \\
\mathsf{state}(q_2,q_0) &\to q \\
s(q_2) &\to q_2
\end{aligned}$$

System Timbuk [Genet Tong 2004 JAR].
Automated construction, guess the *acceleration* $s(q_2) \to q_2$

Tree Automata: Definition

A (ranked) tree automaton is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

- Σ is a ranked signature,
- Q is a finite set of states,
- $F \subseteq Q$ is the subset of final states,
- Δ is a set of transitions of the form $a(q_1, \ldots, q_n) \to q$.

$$\begin{array}{rcl} \Sigma & = & \{0:0, s:1, \mathsf{state}:2\} \\ Q & = & \{q_0, q_1, q_2, q\} \\ F & = & \{q\} \end{array}$$

$$\Delta = \left\{ \begin{array}{rcl} 0 & \to & q_0 \\ \mathsf{state}(q_0, q_0) & \to & q \\ s(q_0) & \to & q_1 \\ \mathsf{state}(q_0, q_1) & \to & q \\ \mathsf{state}(q_1, q_0) & \to & q \\ s(q_1) & \to & q_2 \\ \mathsf{state}(q_2, q_0) & \to & q \\ s(q_2) & \to & q_2 \end{array} \right\}$$

Tree Automata: Definition

A (ranked) tree automaton is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

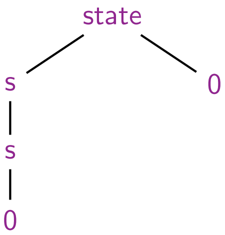
- Σ is a ranked signature,
- Q is a finite set of states,
- $F \subseteq Q$ is the subset of final states,
- Δ is a set of transitions of the form $a(q_1, \ldots, q_n) \to q$.

Regular tree languages

$$\begin{array}{rcl}
\mathcal{L}(\mathcal{A}, q) & = & \left\{ a \in \Sigma_0 \;\middle|\; a \to q \in \Delta \right\} \\
& \cup & \left\{ a(t_1, \ldots, t_n) \;\middle|\; \begin{array}{l} a \in \Sigma_n \\ t_1 \in \mathcal{L}(\mathcal{A}, q_1), \ldots, t_n \in \mathcal{L}(\mathcal{A}, q_n) \\ a(q_1, \ldots, q_n) \to q \in \Delta \end{array} \right\}
\end{array}$$

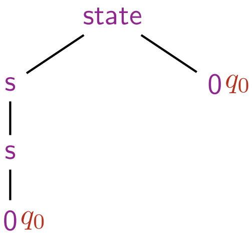
$$\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$$

Tree Automata: Run



$$
\begin{aligned}
\Sigma &= \{0:0, s:1, \mathsf{state}:2\} \\
Q &= \{q_0, q_1, q_2, q\} \\
F &= \{q\}
\end{aligned}
\qquad
\Delta = \left\{
\begin{array}{rcl}
0 & \rightarrow & q_0 \\
\mathsf{state}(q_0, q_0) & \rightarrow & q \\
s(q_0) & \rightarrow & q_1 \\
\mathsf{state}(q_0, q_1) & \rightarrow & q \\
\mathsf{state}(q_1, q_0) & \rightarrow & q \\
s(q_1) & \rightarrow & q_2 \\
\mathsf{state}(q_2, q_0) & \rightarrow & q \\
s(q_2) & \rightarrow & q_2
\end{array}
\right\}
$$

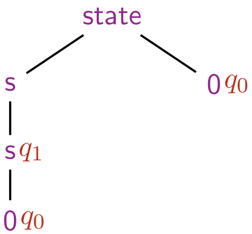
Tree Automata: Run



$$\begin{aligned}\Sigma &= \{0:0, s:1, \mathsf{state}:2\}\\ Q &= \{q_0, q_1, q_2, q\}\\ F &= \{q\}\end{aligned}$$

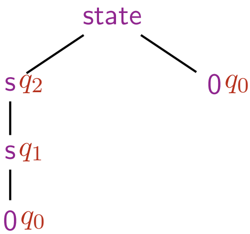
$$\Delta = \left\{\begin{array}{rcl} 0 & \rightarrow & q_0 \\ \mathsf{state}(q_0, q_0) & \rightarrow & q \\ s(q_0) & \rightarrow & q_1 \\ \mathsf{state}(q_0, q_1) & \rightarrow & q \\ \mathsf{state}(q_1, q_0) & \rightarrow & q \\ s(q_1) & \rightarrow & q_2 \\ \mathsf{state}(q_2, q_0) & \rightarrow & q \\ s(q_2) & \rightarrow & q_2 \end{array}\right\}$$

Tree Automata: Run



$$
\begin{aligned}
\Sigma &= \{0:0, s:1, \mathsf{state}:2\} \\
Q &= \{q_0, q_1, q_2, q\} \\
F &= \{q\}
\end{aligned}
\qquad
\Delta = \left\{
\begin{array}{rcl}
0 & \to & q_0 \\
\mathsf{state}(q_0, q_0) & \to & q \\
s(q_0) & \to & q_1 \\
\mathsf{state}(q_0, q_1) & \to & q \\
\mathsf{state}(q_1, q_0) & \to & q \\
s(q_1) & \to & q_2 \\
\mathsf{state}(q_2, q_0) & \to & q \\
s(q_2) & \to & q_2
\end{array}
\right\}
$$

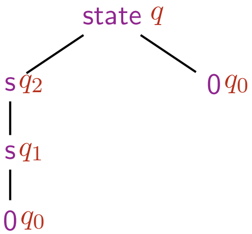
Tree Automata: Run



$$\begin{array}{rcl} \Sigma & = & \{0:0, s:1, \mathsf{state}:2\} \\ Q & = & \{q_0, q_1, q_2, q\} \\ F & = & \{q\} \end{array}$$

$$\Delta = \left\{ \begin{array}{rcl} 0 & \rightarrow & q_0 \\ \mathsf{state}(q_0, q_0) & \rightarrow & q \\ s(q_0) & \rightarrow & q_1 \\ \mathsf{state}(q_0, q_1) & \rightarrow & q \\ \mathsf{state}(q_1, q_0) & \rightarrow & q \\ s(q_1) & \rightarrow & q_2 \\ \mathsf{state}(q_2, q_0) & \rightarrow & q \\ s(q_2) & \rightarrow & q_2 \end{array} \right\}$$

Tree Automata: Run



$$\begin{aligned} \Sigma &= \{0:0, s:1, \mathsf{state}:2\} \\ Q &= \{q_0, q_1, q_2, q\} \\ F &= \{q\} \end{aligned}$$

$$\Delta = \left\{ \begin{array}{rcl} 0 & \to & q_0 \\ \mathsf{state}(q_0, q_0) & \to & q \\ s(q_0) & \to & q_1 \\ \mathsf{state}(q_0, q_1) & \to & q \\ \mathsf{state}(q_1, q_0) & \to & q \\ s(q_1) & \to & q_2 \\ \mathsf{state}(q_2, q_0) & \to & q \\ s(q_2) & \to & q_2 \end{array} \right\}$$

Tree Automata: Main Properties

- Boolean closure of regular tree languages
- membership $t \in \mathcal{L}(\mathcal{A})$ is decidable in PTIME
- emptiness $\mathcal{L}(\mathcal{A}) = \emptyset$ is decidable in PTIME
- finiteness of $\mathcal{L}(\mathcal{A})$ is decidable in PTIME
- universality $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$ is EXPTIME-complete
- inclusion $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$ is EXPTIME-complete
- equivalent to
  - parse trees of a context-free grammar
  - well-formed terms over a sorted signature
  - regular tree grammars
  - sentences of monadic second-order logic of the tree

Concurrent readers/writers: verification

Properties expected:

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

Set of excluded configurations = union of

$$E_1 = \big\{\mathsf{state}\big((q_1 \mid q_2), (q_1 \mid q_2)\big) \rightarrow e_1\big\}$$
$$E_2 = \big\{\mathsf{state}\big((q_0 \mid q_1 \mid q_2), q_2\big) \rightarrow e_2\big\}$$

with $0 \rightarrow q_0$, $s(q_0) \rightarrow q_1$, $s(q_1) \rightarrow q_2$, $s(q_2) \rightarrow q_2$.

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

Regular Model Checking

composition (Boolean closure)

decision procedures (emptiness)

$$R \cap (E_1 \cup E_2) = \varnothing$$

forward closure (term rewriting)
infinite (but regular) configuration set

Limitation: Non-Regular Configuration Set

two files, configurations of the form: $\mathsf{state}(x_1, y_1, x_2, y_2)$

- both files have the same number of readers

$$\mathsf{state}(x, y_1, x, y_2)$$

This set cannot be represented by tree automata
(pumping lemma)

Tree Automata with Local Constraints (Siblings)

- both files have the same number of readers

$$\mathsf{state}(x, y_1, x, y_2)$$

$$\begin{array}{rcl} 0 & \rightarrow & q_0 \\ \mathsf{state}(q_0, q_0, q_0, q_0) & \rightarrow & q \\ s(q_0) & \rightarrow & q_1 \\ \mathsf{state}(q_0, q_1, q_0, q_1) & \rightarrow & q \end{array} \qquad \begin{array}{rcl} \mathsf{state}(q_1, q_0, q_1, q_0) & \rightarrow & q \\ s(q_1) & \rightarrow & q_2 \\ \mathsf{state}(q_2, q_0, q_2, q_0) & \xrightarrow{1=3} & q \\ s(q_2) & \rightarrow & q_2 \end{array}$$

Tree automata with sibling $=$ and $\neq$ constraints
[Bogaert Tison 1992]

Tree Automata with Local Constraints

- both files have the same number of readers

$$\mathsf{pair}\big(\mathsf{state}(x, y_1), \mathsf{state}(x, y_2)\big)$$

$$\begin{array}{rcl} 0 & \to & q_0 \\ \mathsf{state}(q_0, q_0) & \to & p_1 \mid p_2 \\ s(q_0) & \to & q_1 \\ \mathsf{state}(q_0, q_1) & \to & p_1 \mid p_2 \end{array} \qquad \begin{array}{rcl} \mathsf{state}(q_1, q_0) & \to & p_1 \mid p_2 \\ s(q_1) & \to & q_2 \\ \mathsf{state}(q_2, q_0) & \to & p_1 \mid p_2 \\ s(q_2) & \to & q_2 \\ \mathsf{pair}(p_1, p_2) & \xrightarrow{1.1=2.1} & q \end{array}$$

Tree automata with $=$ and $\neq$ (path) constraints

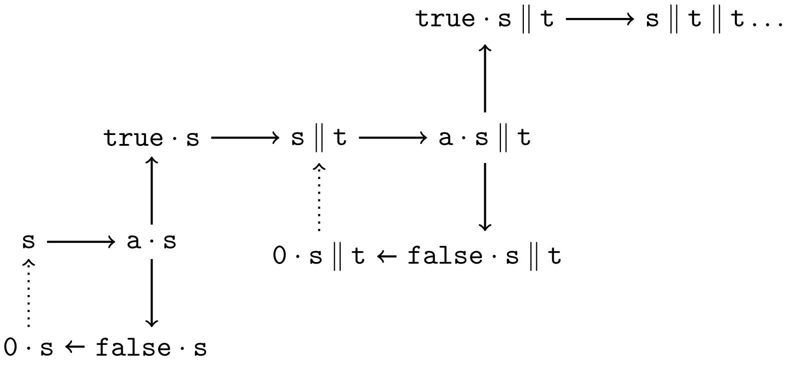
[Mongy 1981 PhD], [Caron 1993 PhD]

Imperative programs with procedure calls and thread creation

Concurrent server [Bouajjani, Touili 02]

```
void server() {
  while(true) {
    if accept() {
    thread_create(&t1,c)
    } else { return }}}
```

$$
\begin{array}{rcll}
\mathtt{s} & \to & \mathtt{a}\cdot\mathtt{s} & (r_1)\\
\mathtt{a} & \to & \mathtt{true} & (r_2)\\
\mathtt{a} & \to & \mathtt{false} & (r_3)\\
\mathtt{true}\cdot\mathtt{s} & \to & \mathtt{s}\parallel\mathtt{t} & (r_4)\\
\mathtt{false} & \to & 0 & (r_5)
\end{array}
$$



Program verification

$$p ::= 0 \mid \mathtt{a} \mid p \cdot p \mid p \parallel p$$

- 0: null process (termination)
- $\mathtt{a}$: program point
- $p \cdot p$: sequential composition
- $p \parallel p$: parallel composition

Transition rules:

- procedure call: $\mathtt{x} \to \mathtt{a} \cdot \mathtt{r}$ ($\mathtt{r}$ = return point)
- conditional continuation: $\mathtt{true} \cdot \mathtt{z} \to \mathtt{t}$
- dynamic thread creation: $\mathtt{x} \to \mathtt{y} \parallel \mathtt{z}$ ($\mathtt{z}$ = return point)
- handshake : $\mathtt{x} \parallel \mathtt{y} \to \mathtt{x}' \parallel \mathtt{y}'$

Algebraic Laws

$$0 \cdot \mathrm{x} = \mathrm{x}$$

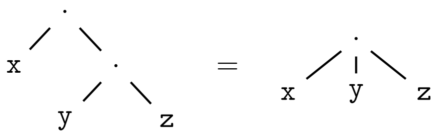
associativity of $\cdot$

$$\mathrm{x} \cdot (\mathrm{y} \cdot \mathrm{z}) = (\mathrm{x} \cdot \mathrm{y}) \cdot \mathrm{z}$$

associativity and commutativity of $\|$

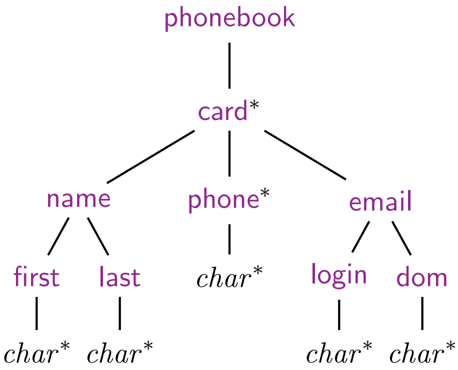
$$\mathrm{x} \,\|\, \mathrm{y} = \mathrm{y} \,\|\, \mathrm{x}$$

trees with $\cdot$ and $\|$ seen as unranked.



XML types and constraints

Type Definition for XML Data



Defines an unranked ordered tree automata language.

Tree automata capture all type formalisms in use for XML data.

XML Documents

ranked tree	unranked tree (XML)
tree automata	schemas (DTD, XML schema, Relax NG)
membership	validation
emptiness	query satisfiability
inclusion	schema entailment
tree transducers, rewrite systems	transformation languages (XSLT)
rewrite closure	type inference

Static Typechecking

[Milo Suciu Vianu 03]

forward closure
(T: tree transformation)

$$T(L_{\mathsf{in}}) \subseteq L_{\mathsf{out}}$$

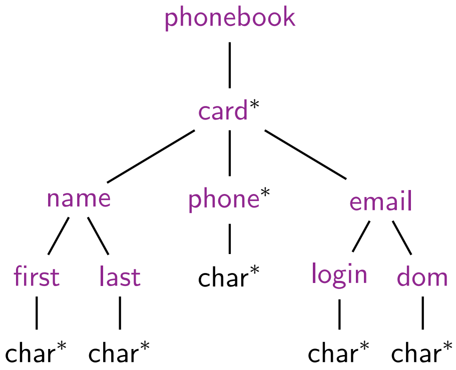
decision procedures

backward closure

$$L_{\mathsf{in}} \cap T^{-1}\left(\overline{L_{\mathsf{out}}}\right) = \varnothing$$

composition (Boolean closure)

Limitation: XML Integrity Constraints



email is a key (ID)

- type constraints can be expressed with tree automata
- integrity constraints cannot be expressed with tree automata

Unranked Ordered Trees & Hedges

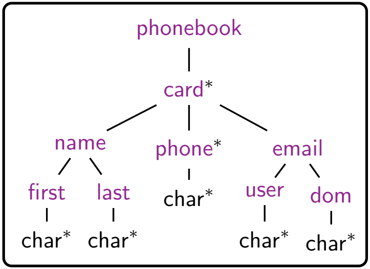
Σ unranked alphabet

$$\begin{array}{rcl} \text{hedge} & = & \text{finite sequence of unranked trees (possibly } \varepsilon) \\ \text{unranked tree} & = & \text{variable} \\ & & a(\text{hedge}) \text{ with } a \in \Sigma \end{array}$$

Unranked Tree Automata: Definition

A hedge automaton (HA [Murata 00])
is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

- Σ is an (unranked) alphabet
- Q is a finite set of states
- $F \subset Q$ is the subset of final states
- Δ is a set of transitions $a(L) \to q$ where $L \subseteq Q^*$ is regular



$$
\begin{array}{rcl}
\mathsf{phonebook}(p_{\mathsf{c}}{}^*) & \to & p_{\mathsf{b}} \\
\mathsf{card}(p_{\mathsf{n}}\ p_{\mathsf{h}}{}^*\ p_{\mathsf{m}}) & \to & p_{\mathsf{c}} \\
\mathsf{name}(p_{\mathsf{f}}\ p_{\mathsf{l}}) & \to & p_{\mathsf{n}} \\
\mathsf{first}(p^*) & \to & p_{\mathsf{f}} \\
\mathsf{last}(p^*) & \to & p_{\mathsf{l}} \\
\mathsf{phone}(p^*) & \to & p_{\mathsf{h}} \\
\mathsf{email}(p_{\mathsf{u}}\ p_{\mathsf{d}}) & \to & p_{\mathsf{m}} \\
\mathsf{user}(p^*) & \to & p_{\mathsf{u}} \\
\mathsf{dom}(p^*) & \to & p_{\mathsf{d}} \\
\mathsf{a} & \to & p \\
\mathsf{b} & \to & p \\
\vdots & &
\end{array}
$$

Unranked Tree Automata: Languages

A hedge automaton is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

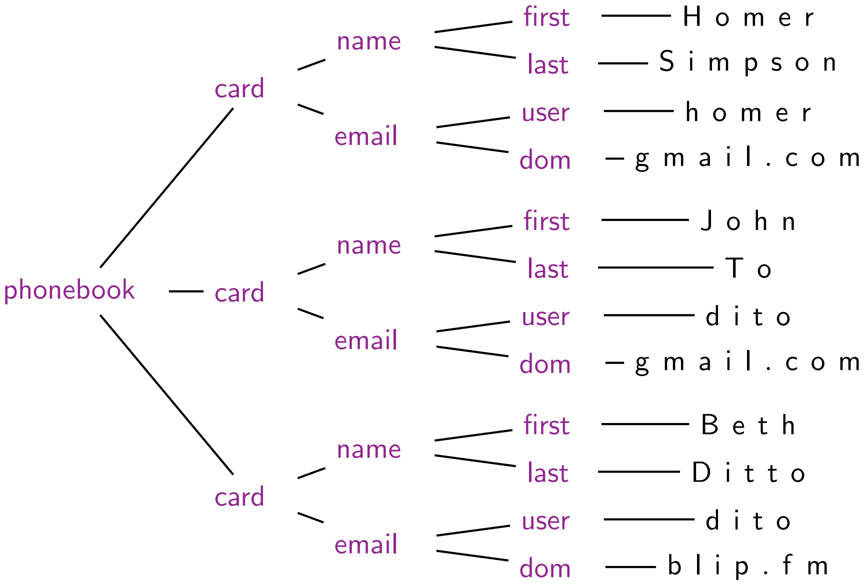
- Σ is an (unranked) alphabet,
- Q is a finite set of states,
- $F \subset Q$ is the subset of final states,
- Δ is a set of transitions $a(L) \to q$ where $L \subseteq Q^*$ is regular

Languages

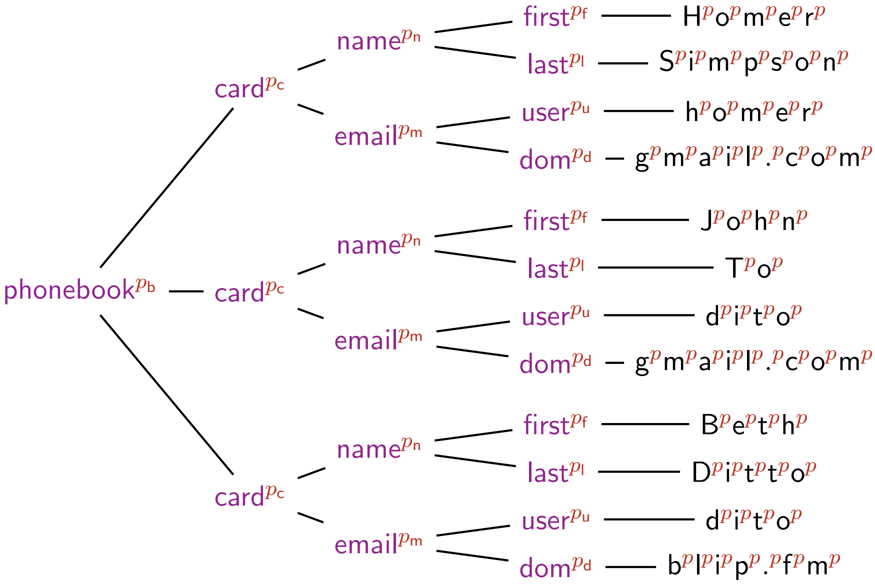
$$\begin{array}{rcl}\mathcal{L}(\mathcal{A}, q) & = & \left\{ a \in \Sigma_0 \;\middle|\; a \to q \in \Delta \right\} \\ & \cup & \left\{ f(t_1, \ldots, t_n) \;\middle|\; \begin{array}{l} f \in \Sigma_n \\ t_1 \in \mathcal{L}(\mathcal{A}, q_1), \ldots, t_n \in \mathcal{L}(\mathcal{A}, q_n) \\ f(L) \to q \in \Delta,\ q_1 \ldots q_n \in L \end{array} \right\} \\ \\ \mathcal{L}(\mathcal{A}) & = & \displaystyle\bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q) \end{array}$$

Equivalent to ranked tree automata via binary encodings

Hedge Automaton Run

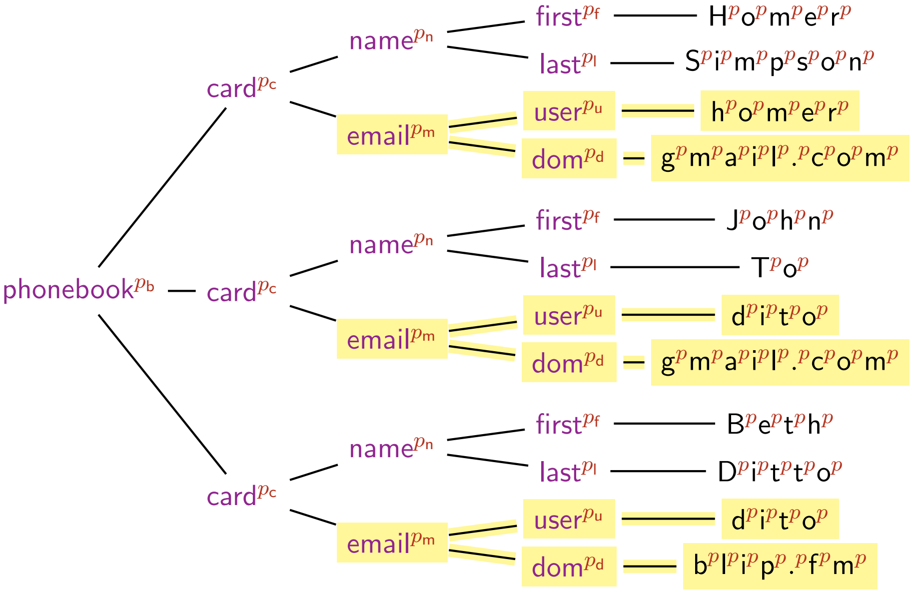


Hedge Automaton Run



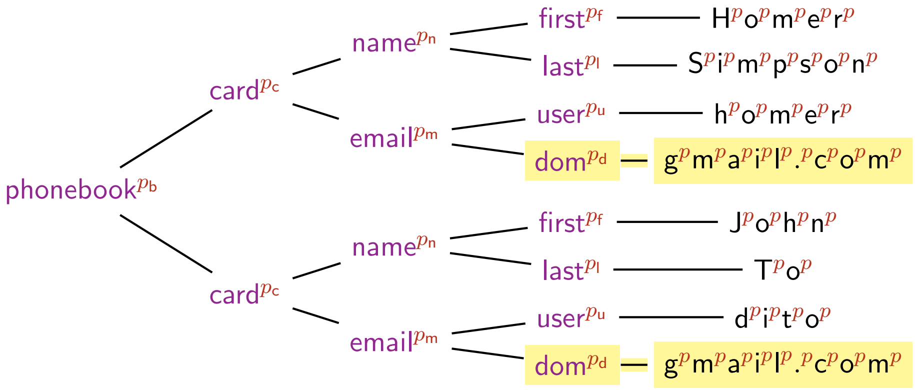
Key Constraint

email is a key: $p_{\mathsf{m}} \not\approx p_{\mathsf{m}}$ $\quad \forall x, y \; p_{\mathsf{m}}(x) \wedge p_{\mathsf{m}}(y) \wedge x \neq y \Rightarrow t|_x \neq t|_y$



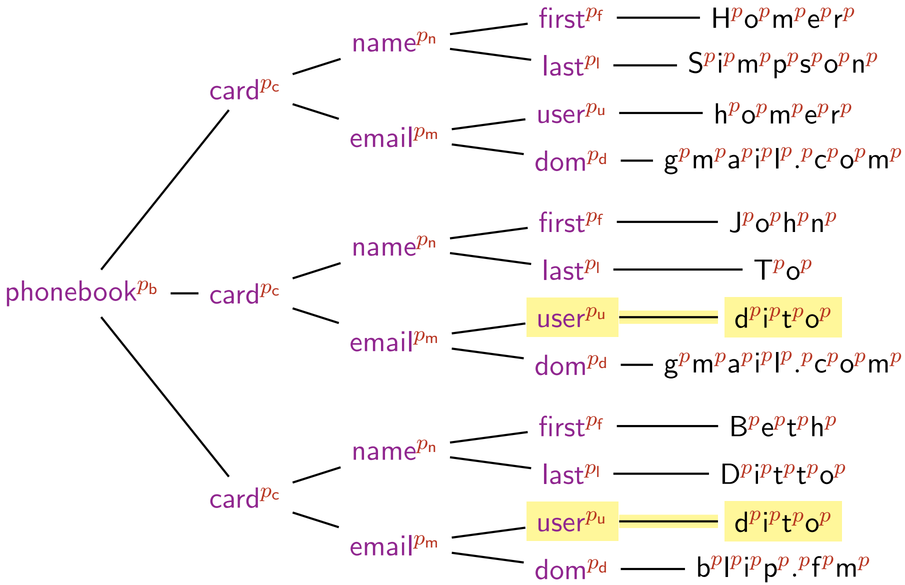
Global Equality Constraint

all domain's coincide $p_{\mathsf{d}} \approx p_{\mathsf{d}}$ $\quad \forall x, y\ p_{\mathsf{d}}(x) \wedge p_{\mathsf{d}}(y) \Rightarrow t|_x = t|_y$



Negation

user is a not a key $\neg p_{\mathsf{u}} \not\approx p_{\mathsf{u}} \quad \exists x, y\ p_{\mathsf{u}}(x) \wedge p_{\mathsf{u}}(y) \wedge x \neq y \wedge t|_x = t|_y$



Tree Automata with Global Constraints: Definition

A tree automaton with global constraints (TAGC[$\approx, \not\approx$])
is a tuple $\mathcal{A} = \langle \Sigma, Q, F, \Delta, C \rangle$ where

- $\langle \Sigma, Q, F, \Delta \rangle$ is a HA
- C is a Boolean combination of atomic constraints
  $C := q_1 \approx q_2 \mid q_1 \not\approx q_2 \mid \neg C \mid C \vee C \mid C \wedge C$ with $q_1, q_2 \in Q$

run r of $\mathcal{A}$ on t: function $dom(t) \to Q$ compatible with Δ,
successful if $r(root) \in F$
language: $\mathcal{L}(\mathcal{A}) = \{t \mid \exists r$ successful run of $\mathcal{A}$ on t, $\langle t, r \rangle \models C\}$

$\langle t, r \rangle \models q_1 \approx q_2$ iff $\forall x, y \in dom(t)\ q_1(x) \wedge q_2(y) \wedge x \neq y \Rightarrow t|_x = t|_y$
$\langle t, r \rangle \models q_1 \not\approx q_2$ iff $\forall x, y \in dom(t)\ q_1(x) \wedge q_2(y) \wedge x \neq y \Rightarrow t|_x \neq t|_y$

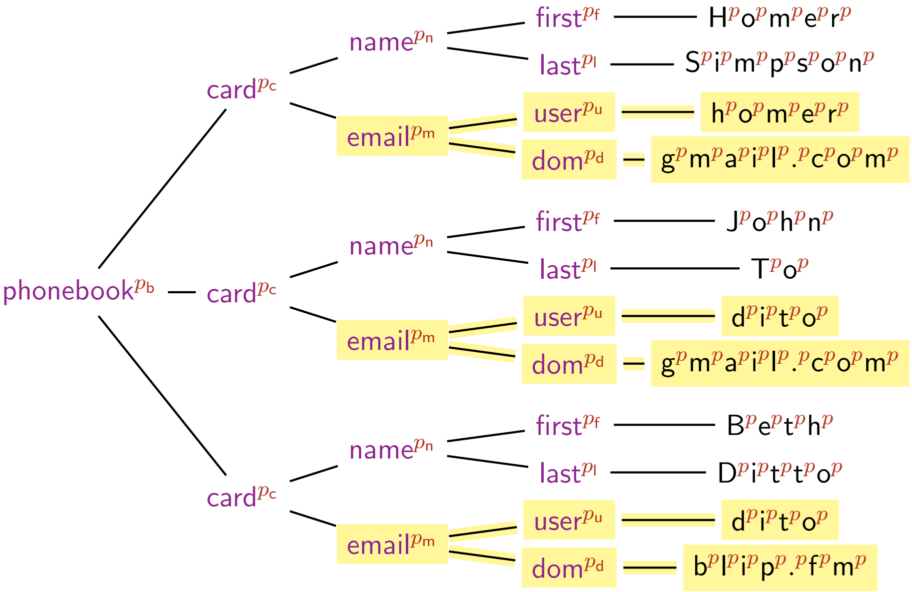
Emptiness Decision TAGC

emptiness is decidable for TAGC$[\approx, \not\approx]$

- One tree is accepted iff a tree of *"small"* height is accepted
- global pumping: replace all subtrees of height h by selected subtrees of height $< h$ while preserving all the relative $\approx$, $\not\approx$
- accepted tree $t \mapsto$ sequence of measures $e_0, e_1, \ldots, e_{h(t)}$ st if $e_i \preceq e_j$ for $i < j$ then there exists a global pumping
- Higman's Lemma, König's Lemma: exists a bound B on the maximal length of sequences (for any t) without $e_i \preceq e_j$, $i < j$
- every t of height $> B$ can be reduced by a global pumping.

Arithmetic Constraints

$$|p_{\mathsf{m}}| = 3,\ \|p_{\mathsf{m}}\| = 3, \quad |p_{\mathsf{u}}| = 3,\ \|p_{\mathsf{u}}\| = 2, \quad |p_{\mathsf{m}}| = \|p_{\mathsf{m}}\|:\ p_{\mathsf{m}} \not\approx p_{\mathsf{m}}$$



Arithmetic Constraints

for a run r on a tree t,
$$\begin{aligned} |q| &= |r^{-1}(q)| \\ \|q\| &= \big|\{t|_x \mid x \in dom(t), r(x) = q\}\big| \end{aligned}$$

linear inequalities $a_q, a \in \mathbb{Z}$
$$\sum_{q \in Q} a_q \cdot |q| \geq a \qquad \text{type } |.|_{\mathbb{Z}}$$
$$\sum_{q \in Q} a_q \cdot \|q\| \geq a \qquad \text{type } \|.\|_{\mathbb{Z}}$$

natural inequalities when all a_q, a have the same sign — types $|.|_{\mathbb{N}}$, $\|.\|_{\mathbb{N}}$

Presburger automata [Seidl et al 2003, 2008], [Dal Zilio Lugiez 2006]: count the siblings of unranked trees (*local cstr*).

Arithmetic Constraints: results

- emptiness is decidable in NPTIME for TAGC$[\,|.|_{\mathbb{Z}}]$
  e.g. [Klaedtke Ruess 2002], [Bojanczyk et al 2009]
- emptiness is undecidable for TAGC$[\approx, |.|_{\mathbb{Z}}]$
  [Godoy et al 2010]
- TAGC$[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}] \equiv$ positive TAGC$[\approx, \not\approx]$ [id]

Other Decidable Extensions

on ranked trees, emptiness is still decidable for

- TAGC$[\approx, \not\approx]$ extended with local $=$ and $\neq$ constraints between siblings, à la [Bogaert Tison]
- same combination for unranked trees?

- TAGC$[\approx, \not\approx]$ where $\approx$ and $\not\approx$ are interpreted modulo flat equational theories

Monadic Second Order Logic

MSO$[+1, \approx, \not\approx, |.|_{\mathbb{Z}}, \|.\|_{\mathbb{Z}}]$ monadic second-order logic

- first order variables x: position in a tree
- second order variables X: finite set of positions

with predicates

	$a(x)$ (x labeled by $a \in \Sigma$ in t)
$+1$	$S_{\downarrow}(x, y)$ (y child of x) and $S_{\rightarrow}(x, y)$ (y next sibling of x)
$\approx$	$X \approx Y$ (for all $x \in X$, $y \in Y$, $t\|_x = t\|_y$)
$\not\approx$	$X \approx Y$ (for all $x \in X$, $y \in Y$, $t\|_x \neq t\|_y$)
$\|.\|_{\mathbb{Z}}$	$\sum a_i \cdot \|X_i\| \geq a$, $a_i, a \in \mathbb{Z}$ ($\|X_i\|$ is cardinality of X_i)
$\|.\|_{\mathbb{N}}$	when a_i, a have same sign
$\\|.\\|_{\mathbb{Z}}$	$\sum a_i \cdot \\|X_i\\| \geq a$ ($\\|X_i\\|$ is cardinality of $\{t\|_x \mid x \in X_i\}$)
$\\|.\\|_{\mathbb{N}}$	when a_i, a have same sign

Monadic Second Order Logic: satisfiability

- MSO$[+1] \equiv$ tree automata [Thatcher Wright 1968]
- MSO$[+1, \approx]$ undecidable
- MSO$[+1, \mathbb{Z}]$ undecidable [Klaedtke Ruess 2002]

EMSO: $\exists X_1 \ldots \exists X_n \, \phi(X_1, \ldots, X_n) \wedge \psi(X_1, \ldots, X_n)$ where

- $\phi(X_1, \ldots, X_n)$ in MSO$[+1]$
- $\psi(X_1, \ldots, X_n)$ in MSO$[+1, \approx, \not\approx, |.|_{\mathbb{Z}}, \|.\|_{\mathbb{Z}}]$, free

- EMSO$[+1, \mathbb{Z}]$ decidable [Klaedtke Ruess 2002]
- fragment of EMSO$[+1, \approx, \not\approx]$ decidable [Filiot et al 2008]
- EMSO$[+1, \approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$ decidable [Godoy et al 2010]

Analysis of XML Transformations

XQuery Update Facility (XQUF)

[W3C recommandation 2011]

extension of XQuery with XML update primitives

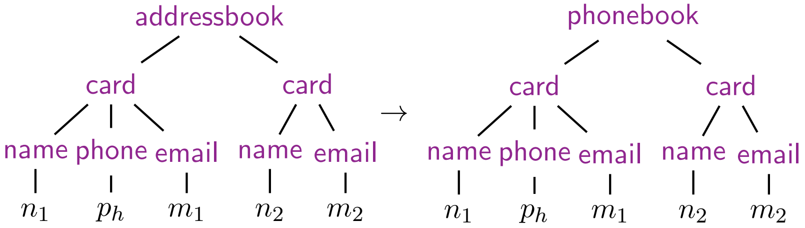
- model of update primitives as parametrized rewrite rules
- forward/backward closure
- application to the verification of read/write XML access control policies

Unranked Ordered Tree Rewriting Systems (HRS)

[Löding Spelten 07], [Touili 07]

$$\text{addressbook}(x) \rightarrow \text{phonebook}(x)$$

- the rule can be applied to any node labeled by addressbook
- the variable x represents a finite sequence of trees (hedge)



$\simeq$ term rewriting modulo A (via binary encoding)

XQUF Primitive Insert First

"insert a tree of type p_{c} (card) as the first children of phonebook"

$$\mathsf{phonebook}(x) \to \mathsf{phonebook}(p_{\mathsf{c}}, x)$$

- p_{c} is a state of a given HA $\mathcal{A}$
- it stands for an arbitrary tree in $\mathcal{L}(\mathcal{A}, p_{\mathsf{c}})$
- this parametrized rule represents an infinity of rules.
  see also [Gilleron 91], [Löding 02]

XQUF Primitive Insert Into

"insert a tree of type p_{c} as an arbitrary children of phonebook"

$$\text{phonebook}(x, y) \rightarrow \text{phonebook}(x, p_{\mathsf{c}}, y)$$

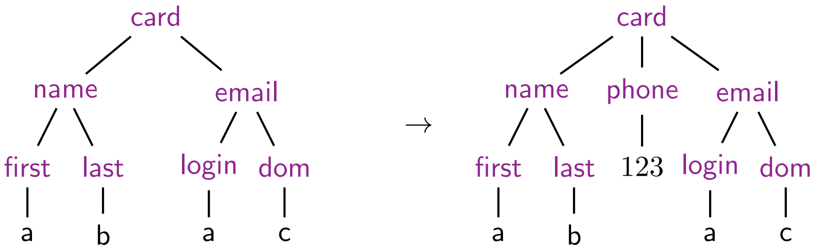
- each of the variables x and y represents an arbitrary hedge

XQUF Primitive Insert After

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

$$\mathsf{name}(x) \to \mathsf{name}(x), p_{\mathsf{h}}$$

- the right hand side of this rule is an hedge of length 2 (not a tree)



XQUF Primitive Replace

"replace a subtree (headed by) card by sequence of n trees of respective types $p_1,\ldots,p_n$"

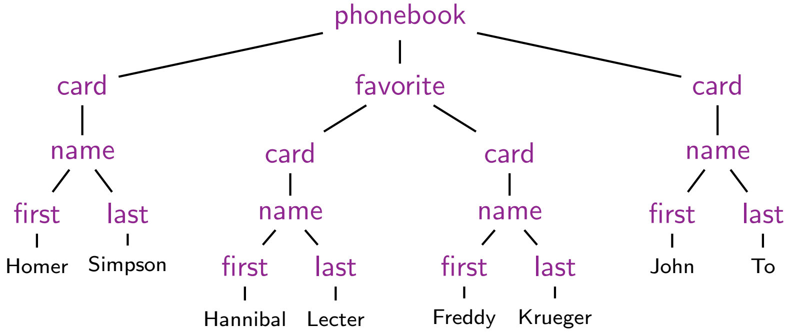
$$\mathsf{address}(x) \rightarrow p_1, \ldots, p_n$$

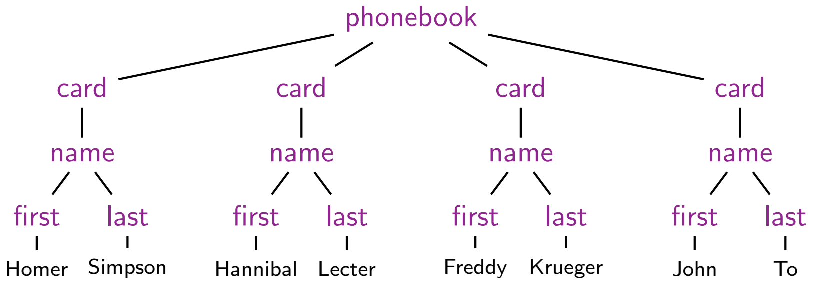
XQUF Primitive Delete

case $n = 0$
"delete a whole subtree headed by card"

$$\mathsf{card}(x) \rightarrow \varepsilon$$

Delete single node (not a XQUF Primitive)





Primitive Delete Single Node

"delete a single node labeled by favorite"

$$\text{favorite}(x) \rightarrow x$$

- the trees in the sequence of children x are moved up to the position of the deleted node.
- *collapsing rule*

- useful for constructing security views of documents

XQUF Primitives: Summary

$$
\begin{array}{rclllrcll}
a(x) & \rightarrow & b(x) & \mathsf{REN} & & & & & \\
a(x) & \rightarrow & a(p, x) & \mathsf{INS}_{\mathsf{first}} & & a(x) & \rightarrow & p, a(x) & \mathsf{INS}_{\mathsf{before}} \\
a(x) & \rightarrow & a(x, p) & \mathsf{INS}_{\mathsf{last}} & & a(x) & \rightarrow & a(x), p & \mathsf{INS}_{\mathsf{after}} \\
a(x, y) & \rightarrow & a(x, p, y) & \mathsf{INS}_{\mathsf{into}} & & & & & \\
a(x) & \rightarrow & p_1 & \mathsf{RPL}_1 & & a(x) & \rightarrow & p_1, \ldots, p_n & \mathsf{RPL} \\
a(x) & \rightarrow & () & \mathsf{DEL} & & a(x) & \rightarrow & x & \mathsf{DEL}_{\mathsf{s}}
\end{array}
$$

Forward Closure of XQUF Primitives

...does not preserve HA languages

e.g. $a(x) \to p_b, p_a, p_c$ (RPL)

$$\{d(a)\} \xrightarrow{*} \{d\big(b^n, a, c^n\big)\}$$

e.g. $a(x) \to x$ ($\mathsf{DEL_s}$)

$$\{a(b, a(b, \ldots, c), c)\} \xrightarrow{*} \{a\big(b^n, a, c^n\big)\}$$

- an extension of HA is needed

HA and CF-HA

[de la Higuera] [Ohsaki 01]

Variants of the hedge automata of [Murata 00]
A HA, resp. CF-HA is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

- Σ is an (unranked) alphabet,
- Q is a finite set of states,
- $F \subset Q$ is the subset of final states,
- Δ is a set of transitions of the form $a(L) \to q$ where
  - $L \subseteq Q^*$ is regular
  - $L \subseteq Q^*$ is context-free

HA $\equiv$ ranked tree automata

CF-HA $\equiv$ ranked tree automata modulo A

Forward and Backward Closure of XQUF Primitives

$a(x)$	$\rightarrow$	$b(x)$	REN
$a(x)$	$\rightarrow$	$a(p, x)$	$\text{INS}_{\text{first}}$
$a(x)$	$\rightarrow$	$a(x, p)$	INS_{last}
$a(x, y)$	$\rightarrow$	$a(x, p, y)$	INS_{into}
$a(x)$	$\rightarrow$	p_1	RPL_1
$a(x)$	$\rightarrow$	$()$	DEL

preserve HA

$a(x)$	$\rightarrow$	$p, a(x)$	$\text{INS}_{\text{before}}$
$a(x)$	$\rightarrow$	$a(x), p$	$\text{INS}_{\text{after}}$
$a(x)$	$\rightarrow$	$p_1, \ldots, p_n$	RPL
$a(x)$	$\rightarrow$	x	DEL_{s}

preserve CF-HA
polynomial construction

inverse-preserve HA
exponential construction

Rule Based Access Control Policies

Access Control Policy (ACP)

$\mathcal{R}_+$ authorized operations of eXQUF (HRS)

$\mathcal{R}_-$ forbidden operations of eXQUF (HRS)

example

$$\mathcal{R}_+ = \left\{ \begin{array}{rcl} \mathsf{addressbook}(x) & \to & \mathsf{addressbook}(p_{\mathsf{c}}, x) \\ \mathsf{card}(x) & \to & \varepsilon \end{array} \right\}$$

- user can insert card with name, delete card

$$\mathcal{R}_- = \{\mathsf{name}(x) \to p_{\mathsf{n}}\}$$

- user cannot change a name

Verification of ACP Inconsistency

Inconsistency

An ACP $\langle \mathcal{R}_+, \mathcal{R}_- \rangle$ is inconsistent
if there exists t, t' such that $t \xrightarrow[\mathcal{R}_-]{} t'$ and $t \xrightarrow[\mathcal{R}_+]{*} t'$.

example: changing name in a card is simulated by deleting and then inserting.

Inconsistency is undecidable for XQUF

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 $t \xrightarrow[\mathcal{R}_+]{*} u$.

Local inconsistency is decidable in PTIME for eXQUF

- using the forward closure construction

Conclusion

Tree automata techniques useful for reasoning tasks concerned with large tree sets.

(sets of configurations)

- Reachability analysis of programs

(sets of XML documents)

- consistency checking for combinations of type and integrity constraints
- forward closure of XQuery Update transformations

  verification of XML read/write access control policies

In computer music,

- verification of interactive realtime systems (mixed instrumental / electronic)
- trees as representation of time (rhythm, structure...), large tree sets represent music databases.