Tree Automata
and Symbolic Constraints Solving

Florent Jacquemard

IRCAM & INRIA
florent.jacquemard@ircam.fr
Automata are computation models.
Tree automata are computation models.
Tree automata for reasoning tasks

big

tree

set

inference

(finite model)

decision

procedure

answer
## Automata as Finite Representation of Infinite Tree Sets

<table>
<thead>
<tr>
<th>Tree Set $S$</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachable configurations of a system/program</td>
<td>Verification (infinite state model checking)</td>
</tr>
<tr>
<td>Forward/backward closure of a set by a transformation</td>
<td>$\text{err} \in S? \ S \cap S_{\text{err}} = \emptyset?$</td>
</tr>
<tr>
<td>Set of solutions</td>
<td>Type checking</td>
</tr>
</tbody>
</table>

### Verification

- Symbolic constraint solving
- Consistency
  
  $S = \emptyset?$ (i.e. $\exists$ solution?)
Reachability Analysis of Programs
Concurrent readers/writers

Example from [Clavel et al. 07 LNCS 4350]

1. state(0, 0) → state(0, s(0))
2. state(r, 0) → state(s(r), 0)
3. state(r, s(w)) → state(r, w)
4. state(s(r), w) → 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. state(0, 0) → state(0, s(0))
2. state(r, 0) → state(s(r), 0)
3. state(r, s(w)) → state(r, w)
4. state(s(r), w) → state(r, w)

initial configuration:

state(0, 0)
Concurrent readers/writers: reachable configurations

1. state(0, 0) $\rightarrow$ state(0, $s(0)$)
2. state($r$, 0) $\rightarrow$ state($s(r)$, 0)
3. state($r$, $s(w)$) $\rightarrow$ state($r$, $w$)
4. state($s(r)$, $w$) $\rightarrow$ state($r$, $w$)

reachable configurations:

state(0, 0)
Concurrent readers/writers: reachable configurations

1. state(0, 0) $\rightarrow$ state(0, $s(0)$)
2. state($r$, 0) $\rightarrow$ state($s(r)$, 0)
3. state($r$, $s(w)$) $\rightarrow$ state($r$, $w$)
4. state($s(r)$, $w$) $\rightarrow$ state($r$, $w$)

reachable configurations:

\[
\begin{array}{c}
\text{state}(0, 0) \quad \text{state}(0, s(0)) \\
\text{state}(0, s(0)) \quad \text{state}(0, 0)
\end{array}
\]
Concurrent readers/writers: reachable configurations

1. state(0, 0) \rightarrow state(0, s(0))
2. state(r, 0) \rightarrow state(s(r), 0)
3. state(r, s(w)) \rightarrow state(r, w)
4. state(s(r), w) \rightarrow state(r, w)

reachable configurations:

\[
\begin{array}{c}
\text{state}(0, 0) \\
\text{state}(s(0), 0)
\end{array}
\]

\[
\begin{array}{c}
\text{state}(0, s(0)) \\
\text{state}(s(s(0)), 0)
\end{array}
\]

\[
\begin{array}{c}
\text{state}(s(0), 0) \\
\text{state}(s(s(0)), 0)
\end{array}
\]

\[
\begin{array}{c}
\text{state}(0, 0) \\
\text{state}(0, s(0))
\end{array}
\]

System Timbuk

[Genet Tong 2004 JAR]

Automated construction, guess the acceleration: state(2) \rightarrow state(2)
Concurrent readers/writers: reachable configurations

1. \(\text{state}(0, 0) \rightarrow \text{state}(0, s(0))\)
2. \(\text{state}(r, 0) \rightarrow \text{state}(s(r), 0)\)
3. \(\text{state}(r, s(w)) \rightarrow \text{state}(r, w)\)
4. \(\text{state}(s(r), w) \rightarrow \text{state}(r, w)\)

reachable configurations:

\[\text{state}(0, 0)\]

tree automaton:

\[0 \rightarrow q_0\]
\[\text{state}(q_0, q_0) \rightarrow q\]
Concurrent readers/writers: reachable configurations

1. \( \text{state}(0, 0) \rightarrow \text{state}(0, s(0)) \)  
2. \( \text{state}(r, 0) \rightarrow \text{state}(s(r), 0) \)  
3. \( \text{state}(r, s(w)) \rightarrow \text{state}(r, w) \)  
4. \( \text{state}(s(r), w) \rightarrow \text{state}(r, w) \)

reachable configurations:

\[
\begin{array}{c}
\text{state}(0, 0) \quad \text{state}(0, s(0)) \\
\quad 1 \quad 3 \\
\end{array}
\]

tree automaton:

\[
\begin{align*}
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
\text{s}(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow q
\end{align*}
\]
Concurrent readers/writers: reachable configurations

1. \text{state}(0, 0) \rightarrow \text{state}(0, s(0))
2. \text{state}(r, 0) \rightarrow \text{state}(s(r), 0)
3. \text{state}(r, s(w)) \rightarrow \text{state}(r, w)
4. \text{state}(s(r), w) \rightarrow \text{state}(r, w)

reachable configurations:

\begin{align*}
\text{state}(0, 0) & \quad \text{state}(0, s(0)) \\
\text{state}(s(0), 0) & \quad \text{state}(s(s(0)), 0) \\
\vdots & \\
\end{align*}

tree automaton:

\begin{align*}
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
s(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow q \\
\text{state}(q_1, q_0) & \rightarrow q \\
s(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow q \\
s(q_2) & \rightarrow q_2 \\
\end{align*}

System Timbuk [Genet Tong 2004 JAR]. Automated construction, guess the acceleration \( s(q_2) \rightarrow q_2 \)
A (ranked) tree automaton is a tuple \( \langle \Sigma, Q, F, \Delta \rangle \) where

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

\[ \Sigma = \{ 0 : 0, s : 1, \text{state} : 2 \} \]
\[ Q = \{ q_0, q_1, q_2, q \} \]
\[ F = \{ q \} \]

\[ \Delta = \{ \]
\[ \quad 0 \to q_0 \]
\[ \quad \text{state}(q_0, q_0) \to q \]
\[ \quad s(q_0) \to q_1 \]
\[ \quad \text{state}(q_0, q_1) \to q \]
\[ \quad \text{state}(q_1, q_0) \to q \]
\[ \quad s(q_1) \to q_2 \]
\[ \quad \text{state}(q_2, q_0) \to q \]
\[ \quad s(q_2) \to q_2 \]
A (ranked) tree automaton is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where

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

**Regular tree languages**

$$
\mathcal{L}(\mathcal{A}, q) = \{ a \in \Sigma_0 \mid a \rightarrow q \in \Delta \}
\cup \left\{ a(t_1, \ldots, t_n) \mid a \in \Sigma_n, t_1 \in \mathcal{L}(\mathcal{A}, q_1), \ldots, t_n \in \mathcal{L}(\mathcal{A}, q_n), a(q_1, \ldots, q_n) \rightarrow q \in \Delta \right\}
$$

$$
\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)
$$
Tree Automata: Run

\[ \Sigma = \{0 : 0, s : 1, \text{state} : 2\} \]
\[ Q = \{q_0, q_1, q_2, q\} \]
\[ F = \{q\} \]

\[ \Delta = \begin{cases} 
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
s(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow q \\
\text{state}(q_1, q_0) & \rightarrow q \\
s(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow q \\
s(q_2) & \rightarrow q_2 
\end{cases} \]
Tree Automata: Run

\[ \begin{array}{c}
\text{state} \\
\downarrow \\
s \\
\downarrow \\
0 \ q_0 \\
\end{array} \]

\[\begin{align*}
\Sigma &= \{0 : 0, s : 1, \text{state} : 2\} \\
Q &= \{q_0, q_1, q_2, q\} \\
F &= \{q\} \\
\Delta &= \begin{cases}
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
s(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow q \\
\text{state}(q_1, q_0) & \rightarrow q \\
s(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow q \\
s(q_2) & \rightarrow q_2 \\
\end{cases}\end{align*}\]
Tree Automata: Run

\[
\begin{align*}
\Sigma &= \{0:0, s:1, \text{state}:2\} \\
Q &= \{q_0, q_1, q_2, q\} \\
F &= \{q\} \\
\Delta &= \left\{ \begin{array}{c@{\quad\rightarrow}c} 
0 & q_0 \\
\text{state}(q_0, q_0) & q \\
s(q_0) & q_1 \\
\text{state}(q_0, q_1) & q \\
\text{state}(q_1, q_0) & q \\
s(q_1) & q_2 \\
\text{state}(q_2, q_0) & q \\
s(q_2) & q_2 \end{array} \right\}
\end{align*}
\]
Tree Automata: Run

\[ \begin{align*}
\Sigma &= \{ 0: 0, s: 1, \text{state: 2} \} \\
Q &= \{ q_0, q_1, q_2, q \} \\
F &= \{ q \}
\end{align*} \]

\[ \Delta = \begin{cases} 
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
s(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow q \\
\text{state}(q_1, q_0) & \rightarrow q \\
s(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow q \\
s(q_2) & \rightarrow q_2 
\end{cases} \]
Tree Automata: Run

\[ \text{state } q \]

\[ \Sigma = \{ 0 : 0, s : 1, \text{state} : 2 \} \]

\[ Q = \{ q_0, q_1, q_2, q \} \]

\[ F = \{ q \} \]

\[ \Delta = \begin{cases} 
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow q \\
\text{state}(q_0, q_1) & \rightarrow q \\
\text{state}(q_1, q_0) & \rightarrow q \\
\text{state}(q_1, q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow q \\
\text{state}(q_2, q_2) & \rightarrow q_2 \\
s(q_0) & \rightarrow q_1 \\
s(q_1) & \rightarrow q_2 \\
s(q_2) & \rightarrow q_2 \\
\end{cases} \]
Tree Automata: Main Properties

- Boolean closure of regular tree languages
- membership $t \in \mathcal{L}(A)$ is decidable in PTIME
- emptiness $\mathcal{L}(A) = \emptyset$ is decidable in PTIME
- finiteness of $\mathcal{L}(A)$ is decidable in PTIME
- universality $\mathcal{L}(A) = \mathcal{T}(\Sigma)$ is EXPTIME-complete
- inclusion $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ 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: \( \text{state}(s(x), s(y)) \)

2. mutual exclusion between writers
   excluded pattern: \( \text{state}(x, s(s(y))) \)

Set of excluded configurations = union of

\[
E_1 = \{ \text{state}((q_1 \mid q_2), (q_1 \mid q_2)) \rightarrow e_1 \} \\
E_2 = \{ \text{state}((q_0 \mid q_1 \mid q_2), q_2) \rightarrow e_2 \}
\]

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)

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

decision procedures (emptiness)

forward closure (term rewriting)

infinite (but regular) configuration set
Limitation: Non-Regular Configuration Set

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

- both files have the same number of readers

\( \text{state}(x, y_1, x, y_2) \)

This set cannot be represented by tree automata (pumping lemma)
Tree Automata with Local Constraints (Siblings)

- both files have the same number of readers

\[
\text{state}(x, y_1, x, y_2)
\]

\[
\begin{align*}
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0, q_0, q_0) & \rightarrow q \\
\text{s}(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1, q_0, q_1) & \rightarrow q \\
\end{align*}
\]

\[
\begin{align*}
\text{state}(q_1, q_0, q_1, q_0) & \rightarrow q \\
\text{s}(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0, q_2, q_0) & \rightarrow q \\
\text{s}(q_2) & \rightarrow q_2 \\
\end{align*}
\]

Tree automata with sibling = and \(\neq\) constraints

[Bogaert Tison 1992]
Tree Automata with Local Constraints

- both files have the same number of readers

\[
\text{pair}(\text{state}(x, y_1), \text{state}(x, y_2))
\]

\[
\begin{align*}
0 & \rightarrow q_0 \\
\text{state}(q_0, q_0) & \rightarrow p_1 \mid p_2 \\
\text{s}(q_0) & \rightarrow q_1 \\
\text{state}(q_0, q_1) & \rightarrow p_1 \mid p_2 \\
\text{state}(q_1, q_0) & \rightarrow p_1 \mid p_2 \\
\text{s}(q_1) & \rightarrow q_2 \\
\text{state}(q_2, q_0) & \rightarrow p_1 \mid p_2 \\
\text{s}(q_2) & \rightarrow q_2 \\
\text{pair}(p_1, p_2) & \xrightarrow{1.1=2.1} q
\end{align*}
\]

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 }
    }
}

true \cdot s \rightarrow s \parallel t \quad (r_1)

a \rightarrow \text{true} \quad (r_2)

a \rightarrow \text{false} \quad (r_3)

true \cdot s \rightarrow s \parallel t \quad (r_4)

false \rightarrow 0 \quad (r_5)

true \cdot s \parallel t \rightarrow s \parallel t \parallel t \ldots
Program verification

\[ p ::= 0 \mid a \mid p \cdot p \mid p \parallel p \]

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

Transition rules:
- procedure call: \(x \rightarrow a \cdot r\) (\(r = \text{return point}\))
- conditional continuation: \(\text{true} \cdot z \rightarrow t\)
- dynamic thread creation: \(x \rightarrow y \parallel z\) (\(z = \text{return point}\))
- handshake: \(x \parallel y \rightarrow x' \parallel y'\)
Algebraic Laws

\[ 0 \cdot x = x \]

associativity of \( \cdot \)

\[ x \cdot (y \cdot z) = (x \cdot y) \cdot z \]

associativity and commutativity of \( \mid \mid \)

\[ x \mid\mid y = y \mid\mid x \]

trees with \( \cdot \) and \( \mid\mid \) 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

<table>
<thead>
<tr>
<th>ranked tree</th>
<th>unranked tree (XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree automata</td>
<td>schemas (DTD, XML schema, Relax NG)</td>
</tr>
<tr>
<td>membership</td>
<td>validation</td>
</tr>
<tr>
<td>emptiness</td>
<td>query satisfiability</td>
</tr>
<tr>
<td>inclusion</td>
<td>schema entailment</td>
</tr>
<tr>
<td>tree transducers, rewrite systems</td>
<td>transformation languages (XSLT)</td>
</tr>
<tr>
<td>rewrite closure</td>
<td>type inference</td>
</tr>
</tbody>
</table>
Static Typechecking

forward closure

\[ T(L_{in}) \subseteq L_{out} \]

backward closure

\[ L_{in} \cap T^{-1}(\overline{L_{out}}) = \emptyset \]

composition (Boolean closure)

decision procedures
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

hedge = finite sequence of unranked trees (possibly ε)
unranked tree = variable
a(hedge) with a ∈ Σ
Unranked Tree Automata: Definition

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

- \( \Sigma \) is an (unranked) alphabet
- \( Q \) is a finite set of states
- \( F \subseteq Q \) is the subset of final states
- \( \Delta \) is a set of transitions \( a(L) \rightarrow q \)

where \( L \subseteq Q^* \) is regular

\[
\begin{align*}
\text{phonebook}(p_c^*) & \rightarrow p_b \\
\text{card}(p_n p_h^* p_m) & \rightarrow p_c \\
\text{name}(p_f p_l) & \rightarrow p_n \\
\text{first}(p^*) & \rightarrow p_f \\
\text{last}(p^*) & \rightarrow p_l \\
\text{phone}(p^*) & \rightarrow p_h \\
\text{email}(p_u p_d) & \rightarrow p_m \\
\text{user}(p^*) & \rightarrow p_u \\
\text{dom}(p^*) & \rightarrow p_d \\
a & \rightarrow p \\
b & \rightarrow p \\
: & \\
\end{align*}
\]
Unranked Tree Automata: Languages

A hedge automaton is a tuple $\langle \Sigma, Q, F, \Delta \rangle$ where
- $\Sigma$ is an (unranked) alphabet,
- $Q$ is a finite set of states,
- $F \subset Q$ is the subset of final states,
- $\Delta$ is a set of transitions $a(L) \to q$ where $L \subseteq Q^*$ is regular.

Languages

$$\mathcal{L}(\mathcal{A}, q) = \left\{ a \in \Sigma_0 \mid a \to q \in \Delta \right\}$$
$$\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$$

Equivalent to ranked tree automata via binary encodings
Hedge Automaton Run

phonebook
  └── card
      └── name
          └── first
              └── Homer
          └── last
              └── Simpson
      └── email
          └── user
              └── homer
          └── dom
              └── gmail.com
      └── name
          └── first
              └── John
          └── last
              └── To
          └── user
              └── dito
          └── dom
              └── gmail.com
      └── email
          └── user
              └── Beth
          └── dom
              └── Ditto
Hedge Automaton Run

phonebook

- card
  - email
  - user
  - dom
  - first
  - last

- card
  - email
  - user
  - dom
  - first
  - last

- card
  - email
  - user
  - dom
  - first
  - last
Key Constraint

email is a key: $p_m \not\equiv p_m \quad \forall x, y \quad p_m(x) \land p_m(y) \land x \neq y \Rightarrow t|x \neq t|y$
Global Equality Constraint

all domain's coincide $p_d \approx p_d \quad \forall x, y \ p_d(x) \land p_d(y) \Rightarrow t|_x = t|_y$
Negation

\[ \neg \text{user is a not a key} \iff \exists x, y \ p_u(x) \wedge p_u(y) \wedge x \neq y \wedge t | x = t | y \]
A tree automaton with global constraints \((\text{TAGC}[\simeq, \neq])\) 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
  \[ C := q_1 \simeq q_2 \mid q_1 \not\simeq q_2 \mid \neg C \mid C \lor C \mid C \land C \] with \(q_1, q_2 \in Q\)

run \(r\) of \(\mathcal{A}\) on \(t\): function \(\text{dom}(t) \to Q\) compatible with \(\Delta\),

successful if \(r(\text{root}) \in F\)

language: \(\mathcal{L}(\mathcal{A}) = \{ t \mid \exists r \text{ successful run of } \mathcal{A} \text{ on } t, \langle t, r \rangle \models C \}\)

\[
\langle t, r \rangle \models q_1 \simeq q_2 \text{ iff } \forall x, y \in \text{dom}(t) \ q_1(x) \land q_2(y) \land x \neq y \Rightarrow t|_x = t|_y \\
\langle t, r \rangle \models q_1 \not\simeq q_2 \text{ iff } \forall x, y \in \text{dom}(t) \ q_1(x) \land q_2(y) \land x \neq y \Rightarrow t|_x \neq t|_y
\]
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 $e_i \leq e_j$ 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

\[|p_m| = 3, \|p_m\| = 3, \quad |p_u| = 3, \|p_u\| = 2, \quad |p_m| = \|p_m\|: \quad p_m \not\subseteq p_m\]
Arithmetic Constraints

for a run \( r \) on a tree \( t \),

\[
|q| = |r^{-1}(q)| \\
\|q\| = |\{t|x | x \in \text{dom}(t), r(x) = q\}|
\]

linear inequalities \( a_q, a \in \mathbb{Z} \)

\[
\sum_{q \in Q} a_q \cdot |q| \geq a \quad \text{type } |.|_{\mathbb{Z}}
\]

\[
\sum_{q \in Q} a_q \cdot \|q\| \geq a \quad \text{type } ||.||_{\mathbb{Z}}
\]

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 $\text{NPTIME}$ for $\text{TAGC}[\cdot, |_\mathbb{Z}]$

  e.g. [Klaedtke Ruess 2002], [Bojanczyk et al 2009]

- **emptiness** is undecidable for $\text{TAGC}[\approx, |_\mathbb{Z}]$  

  [Godoy et al 2010]

- $\text{TAGC}[\approx, \not\approx, |_\mathbb{N}, |.|_\mathbb{N}] \equiv \text{positive TAGC}[\approx, \not\approx]$  

  [id]
Other Decidable Extensions

on ranked trees, emptiness is still decidable for

- TAGC[≃, †] extended with local = and ≠ constraints between siblings, à la [Bogaert Tison]
- same combination for unranked trees?

- TAGC[≃, †] where ≃ and † are interpreted modulo flat equational theories
Monadic Second Order Logic

\[ \text{MSO}[+1, \approx, \not\approx, \cdot|\mathbb{Z}, \cdot||\mathbb{Z}] \] monadic second-order logic

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

with predicates

\[ a(x) \ (x \text{ labeled by } a \in \Sigma \text{ in } t) \]
\[ +1 \quad S_\downarrow(x, y) \ (y \text{ child of } x) \text{ and } S_\rightarrow(x, y) \ (y \text{ next sibling of } x) \]
\[ \approx \quad X \approx Y \ (\text{for all } x \in X, y \in Y, t|_x = t|_y) \]
\[ \not\approx \quad X \not\approx Y \ (\text{for all } x \in X, y \in Y, t|_x \neq t|_y) \]
\[ \cdot|\mathbb{Z} \quad \sum a_i \cdot |X_i| \geq a, \ a_i, a \in \mathbb{Z} \quad (|X_i| \text{ is cardinality of } X_i) \]
\[ \cdot|\mathbb{N} \quad \text{when } a_i, a \text{ have same sign} \]
\[ \cdot||\mathbb{Z} \quad \sum a_i \cdot ||X_i|| \geq a \ (||X_i|| \text{ is cardinality of } \{t|_x \mid x \in X_i\}) \]
\[ \cdot||\mathbb{N} \quad \text{when } a_i, a \text{ have same sign} \]
Monadic Second Order Logic: satisfiability

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

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

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

- $\text{EMSO}[+1, \mathbb{Z}]$ decidable [Klaedtke Ruess 2002]
- fragment of $\text{EMSO}[+1, \approx, \not\exists]$ decidable [Filiot et al 2008]
- $\text{EMSO}[+1, \approx, \not\exists, .|\mathbb{N}, |||\mathbb{N}]$ decidable [Godoy et al 2010]
Analysis of XML Transformations
XQuery Update Facility (XQUF)

[W3C recommendation 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)

[Łoeding Spelten 07], [Touili 07]

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)

\[ \begin{array}{ccc}
\text{addressbook} & \rightarrow & \text{phonebook} \\
\text{card} & \rightarrow & \text{card} \\
\text{name} & \rightarrow & \text{name} \\
\text{phone} & \rightarrow & \text{phone} \\
\text{email} & \rightarrow & \text{email} \\
\text{name} & \rightarrow & \text{name} \\
\text{phone} & \rightarrow & \text{phone} \\
\text{email} & \rightarrow & \text{email} \\
\end{array} \]

\( \simeq \) 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"

$$\text{phonebook}(x) \rightarrow \text{phonebook}(p_c, x)$$

- $p_c$ is a state of a given HA $A$
- it stands for an arbitrary tree in $\mathcal{L}(A, p_c)$
- this parametrized rule represents an infinity of rules.
  see also [Gilleron 91], [Löding 02]
"insert a tree of type $p_c$ as an arbitrary children of phonebook"

\[ \text{phonebook}(x, y) \rightarrow \text{phonebook}(x, p_c, y) \]

- 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"

\[
\text{name}(x) \rightarrow \text{name}(x), p_h
\]

- 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 $p_1, \ldots, p_n$"

\[
\text{address}(x) \rightarrow p_1, \ldots, p_n
\]

XQUF Primitive Delete

\[
\text{case } n = 0 \\
\text{"delete a whole subtree headed by card"}
\]

\[
\text{card}(x) \rightarrow \varepsilon
\]
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

\[\begin{align*}
    a(x) & \rightarrow b(x) \\
    a(x) & \rightarrow a(p, x) \\
    a(x) & \rightarrow a(x, p) \\
    a(x, y) & \rightarrow a(x, p, y) \\
    a(x) & \rightarrow p_1 \\
    a(x) & \rightarrow () \\
    \end{align*}\]

- **REN**
- **INS\text{first}**
- **INS\text{last}**
- **INS\text{into}**
- **INS\text{before}**
- **INS\text{after}**
- **RPL\text{1}**
- **RPL**
- **DEL**
- **DEL\text{s}**

\[\begin{align*}
    a(x) & \rightarrow p, a(x) \\
    a(x) & \rightarrow a(x), p \\
    a(x) & \rightarrow p_1, \ldots, p_n \\
    a(x) & \rightarrow x \\
    \end{align*}\]
Forward Closure of XQUF Primitives

...does not preserve HA languages

e.g. \( a(x) \rightarrow p_b, p_a, p_c \) \hspace{1cm} \text{(RPL)}

\[ \{ d(a) \} \xrightarrow{*} \{ d(b^n, a, c^n) \} \]

e.g. \( a(x) \rightarrow x \) \hspace{1cm} \text{(DEL_s)}

\[ \{ a(b, a(b, \ldots, c), c) \} \xrightarrow{*} \{ a(b^n, a, c^n) \} \]

- an extension of HA is needed
HA and CF-HA

Variants of the hedge automata of [Murata 00]

A HA, resp. CF-HA is a tuple \( \langle \Sigma, Q, F, \Delta \rangle \) where

- \( \Sigma \) is an (unranked) alphabet,
- \( Q \) is a finite set of states,
- \( F \subseteq Q \) is the subset of final states,
- \( \Delta \) 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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(x) \rightarrow b(x) )</td>
<td>REN</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow a(p, x) )</td>
<td>INS_first</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow a(x, p) )</td>
<td>INS_last</td>
<td></td>
</tr>
<tr>
<td>( a(x, y) \rightarrow a(x, p, y) )</td>
<td>INS_into</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow p_1 )</td>
<td>RPL_1</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow () )</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>

preserve HA

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(x) \rightarrow p, a(x) )</td>
<td>INS_before</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow a(x), p )</td>
<td>INS_after</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow p_1, \ldots, p_n )</td>
<td>RPL</td>
<td></td>
</tr>
<tr>
<td>( a(x) \rightarrow x )</td>
<td>DEL_s</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\mathcal{R}_+ = \begin{cases} 
\text{addressbook}(x) \rightarrow \text{addressbook}(p_c, x) \\
\text{card}(x) \rightarrow \varepsilon 
\end{cases}
\]

- user can insert card with name, delete card

\[
\mathcal{R}_- = \{ \text{name}(x) \rightarrow p_n \}
\]

- user cannot change a name
Verification of ACP Inconsistency

**Inconsistency**

An ACP $\langle R_+, R_- \rangle$ is inconsistent if there exists $t, t'$ such that $t \xrightarrow{R_-} t'$ and $t \xrightarrow{R_+} t'$.

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

Inconsistency is undecidable for XQUF

**Local Inconsistency**

An ACP $\langle R_+, R_- \rangle$ is locally inconsistent for $t$ if there exists $u$ such that $t \xrightarrow{R_-} u$ and $t \xrightarrow{R_+} u$.

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.