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

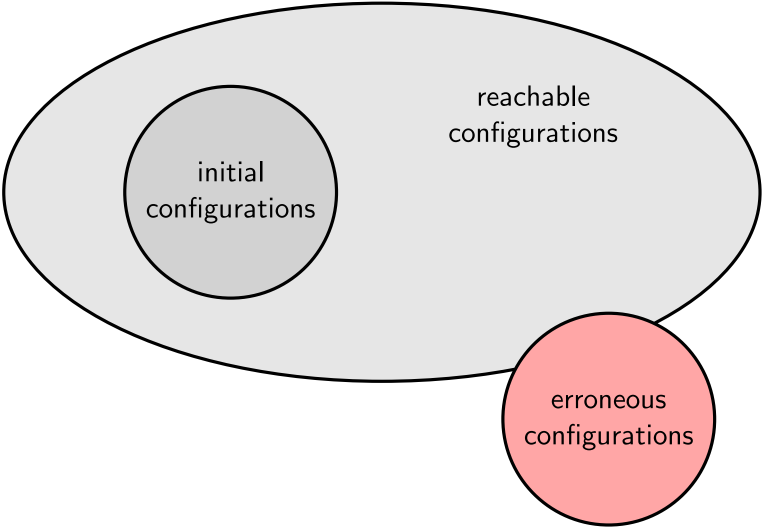
= finite representations of infinite set of labeled trees

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

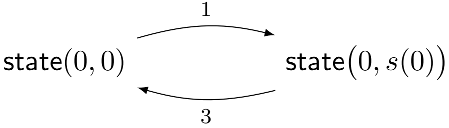
$$\begin{array}{lrcl}
1. & \mathsf{state}(0,0) & = & \mathsf{state}(0,s(0)) \\
2. & \mathsf{state}(r,0) & = & \mathsf{state}(s(r),0) \\
3. & \mathsf{state}(r,s(w)) & = & \mathsf{state}(r,w) \\
4. & \mathsf{state}(s(r),w) & = & \mathsf{state}(r,w)
\end{array}$$

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

Concurrent readers/writers: reachable configurations

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

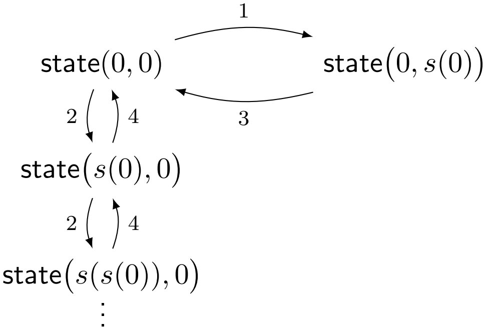
Reachable configurations:



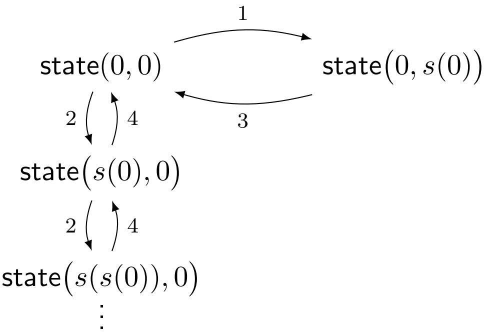
Concurrent readers/writers: reachable configurations

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

Reachable configurations:



Concurrent readers/writers: finite representation



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

Concurrent readers/writers: automata construction

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

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

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

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

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

Concurrent readers/writers: automata construction

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

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

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

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

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

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

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

$$\mathsf{state}(s(q_0 \mid q_1 \mid q_2), q_0) \in q \Rightarrow \mathsf{state}(q_0 \mid q_1 \mid q_2, q_0) \in q$$

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

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: state$(s(x), s(y))$
2. mutual exclusion between writers
   forbidden pattern: state$(x, s(s(y)))$

The red set: union of

1. state$\big((q_1 \mid q_2), (q_1 \mid q_2)\big)$
2. 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.

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

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.

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

set of initial config.:

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

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

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

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: $X \to Y \cdot Z \quad (Z = \text{return point})$
- 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 }
    }
  }
```

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

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

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

- Process rewrite systems (PRS) [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

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 $t(p) \in \Sigma_n$ $(n \geq 1)$, $\big\{pj \in \mathcal{P}os(t) \bigm| j \in \mathbb{N}\big\} = \{p1, ..., pn\}$,
- 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

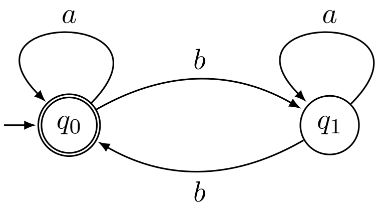
A *context* is a linear term.

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

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



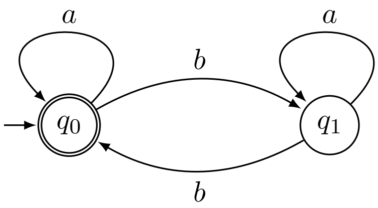
word. run on $aabba$: $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_0$.

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

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

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

with $f(L_1, \ldots, L_n) := \{f(t_1, \ldots, t_n) \mid t_1 \in L_1, \ldots, t_n \in L_n\}$.

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 :

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

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

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

Tree Automata: example 2

Example :

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

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

Example :

Ground terms embedding the pattern $\neg(\neg(x))$: $\mathcal{A} \cup \{\neg(q_\mathsf{f}) \rightarrow q_\mathsf{f}, \vee(q_\mathsf{f}, q_*) \rightarrow q_\mathsf{f}, \vee(q_*, q_\mathsf{f}) \rightarrow q_\mathsf{f}, \ldots\}$ (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: $L(\mathcal{A}) = \{t\sigma \mid \sigma : \mathcal{X} \to \mathcal{T}(\Sigma)\}$.

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

TA = 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 $\neg(\neg(x))$):

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

Top-Down Tree Automata and Determinism

Definition : Determinism

A top-down tree automaton $(\Sigma, Q, Q^{\text{init}}, \Delta)$ is *deterministic* if $|Q^{\text{init}}| = 1$ 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 $(\Sigma, Q, Q^{\text{init}}, \Delta)$ is *deterministic* if $|Q^{\text{init}}| = 1$ 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

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 $k \leq n$,

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

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

$v \rhd_k \langle v_1, v_2 \rangle$ 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 left$(t_2) \in T$, right$(t_2) \in T$

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 :

$$h : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma')$$
$$h\big(f(t_1, \ldots, t_n)\big) := t_f\big\{x_1 \leftarrow h(t_1), \ldots, x_n \leftarrow h(t_n)\big\}$$

for $f \in \Sigma_n$, with $t_f \in \mathcal{T}\big(\Sigma', \{x_1, \ldots, x_n\}\big)$.

h is called

- *linear* if for all $f \in \Sigma$, t_f is linear,
- *complete* if for all $f \in \Sigma_n$, $vars(t_f) = \{x_1, \ldots, x_n\}$,
- *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 $r = f(q_1, \ldots, q_n) \rightarrow q \in \Delta$, 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 $g(q^r_{p_1}, \ldots, q^r_{p_m}) \rightarrow q^r_p \in \Delta_r$,
- 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 $L = \{f(g^n(a)) \mid n \geq 0\}$,
$h(L) = \{f'(g^n(a), g^n(a)) \mid n \geq 0\}$ 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

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

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

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

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 $\{q_2(t) \mid t \in \mathcal{T}\big(\{\mathsf{t}, \mathsf{n}, \underline{\mathsf{t}}, \underline{\mathsf{n}}\}\big),\ t \text{ contains at least } 2 \text{ tokens}\}$ (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

Term Rewriting

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

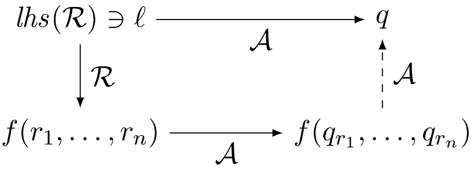
Theorem : [Brainerd 69]

Ground TRS are preserving regularity.

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

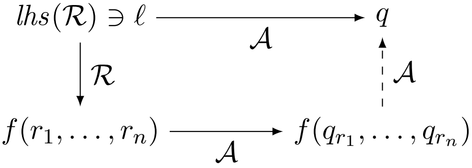
$$\mathcal{A}_{\mathsf{in}} \cup (\Sigma, Q_{\mathcal{R}}, \emptyset, \{f(q_{r_1}, \ldots, q_{r_n}) \to q_r \mid r = f(r_1, \ldots r_n) \in Q_{\mathcal{R}}\})$$

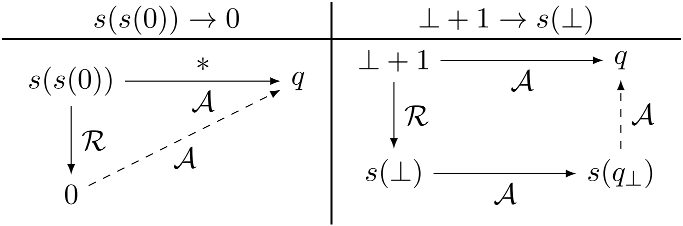
where $Q_{\mathcal{R}} = \mathit{strict\ subterms}(\mathit{rhs}(\mathcal{R}))$,
and add transitions according to the schema:



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

Ground TRS (examples)





Linear and right-shallow TRS

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

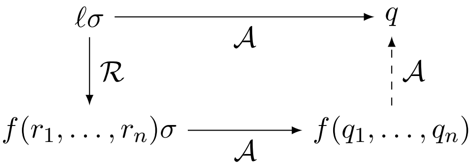
Theorem : [Salomaa 88]

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

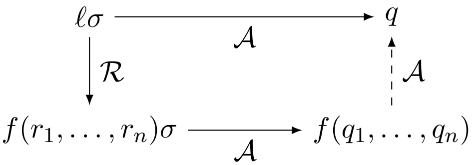
$$\mathcal{A}_{\mathsf{in}} \cup (\Sigma, Q_{\mathcal{R}}, \emptyset, \{f(q_{r_1}, \ldots, q_{r_n}) \to q_r \mid r = f(r_1, \ldots r_n) \in Q_{\mathcal{R}}\})$$

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

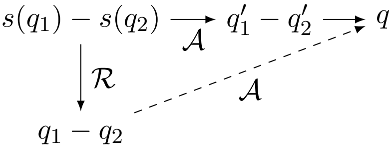
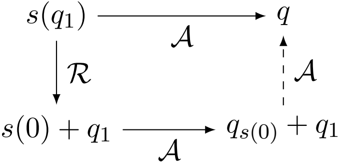


where $\ell \in \mathit{lhs}(\mathcal{R})$, substitution $\sigma : \mathit{vars}(\ell) \to Q$, for all $i \leq n$, if $r_i \notin \mathcal{X}$ then $q_i = q_{r_i}$ and $q_i = r_i\sigma$ otherwise.

Linear and right-shallow TRS (examples)



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.

$s(x) - s(y) \to x - y$	$s(x) \to s(0) + x$
	

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), $L_{\text{in}} = \{f(\dots f(c))\}$.
$\mathcal{R}^*(L_{\text{in}}) \cap \mathcal{T}\big(\{g,c\}\big)$ 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: $\text{app}(\text{nil}, y) \to y$, $\text{app}\big(\text{cons}(x,y), z\big) \to \text{cons}\big(x, \text{app}(y,z)\big)$.
- [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

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

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]

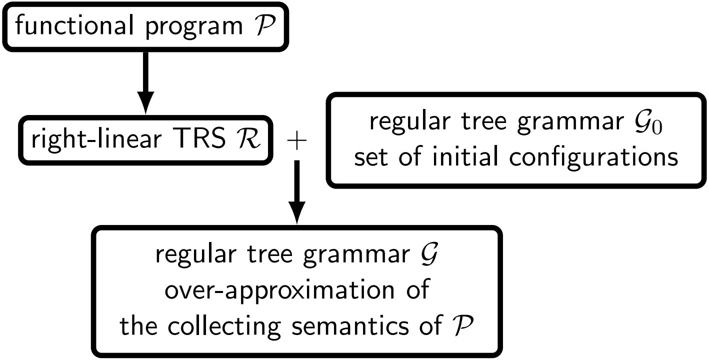
$$X_1 \cdot \ldots \cdot X_n \to Y_1 \cdot \ldots \cdot Y_m, \quad X_1 \to Y_1 \parallel \ldots \parallel Y_m$$

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 :

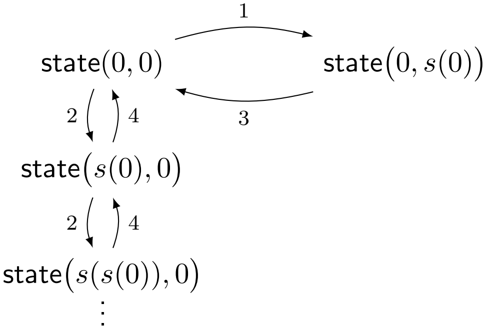
$\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$, $\mathcal{G} = (\{X_0, X_1\}, X_1, \Sigma, P)$.

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

Approximations of Collecting Semantics: Example

Concurrent readers/writers: reachable configurations

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



Approximations of Collecting Semantics: Example

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

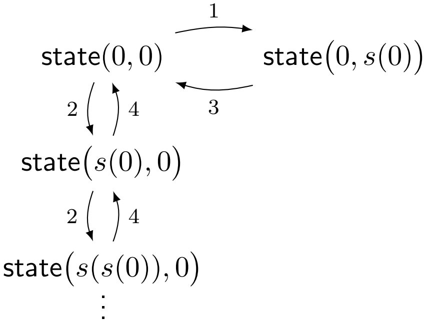
$R_0 := \mathsf{state}(0,0)$

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

Approximations of Collecting Semantics: Example

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

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



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);
```

$$R_2 : \quad \text{first}(\text{nil}, X_s) \quad \to \quad \text{nil}$$
$$R_3 : \quad \text{first}(\text{cons}(1, M), \text{cons}(X, X_s)) \quad \to \quad \text{cons}(X, \text{first}(M, X_s))$$

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

$$R_4 : \quad \text{sequence}(Y) \quad \to \quad \text{cons}(Y, \text{sequence}(\text{cons}(1, Y)))$$

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

$$R_1 : \quad \text{g}(N) \quad \to \quad \text{first}(N, \text{sequence}(\text{nil}))$$

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.

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

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

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

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),
- $L_{\underline{a}}^{t} = \{p \in \mathcal{P}os(t) \mid t(p) = a\}$.

FOL with k successors

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

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

WSkS: syntax

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

Notation: $\phi(x_1, \ldots, x_m, X_1, \ldots, X_n)$,
where $x_1, \ldots, x_m$, $X_1, \ldots, X_n$ 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 $\mathcal{L}_\Sigma$,

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

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: 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: 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:
  $\mathsf{sing}(X) \equiv X \neq \emptyset \wedge \forall Y \, \big(Y \subseteq X \Rightarrow (Y = X \vee Y = \emptyset)\big)$
- $\leq$ (without $<$)

WSkS: examples (2)

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

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

or

$$x \leq y \quad \equiv \quad \exists X \, \left(\forall z \; z \in X \Rightarrow (\exists z' \bigvee_{i \leq k} S_i(z', z) \wedge z' \in X) \vee z = x\right) \\ \wedge \, y \in X$$

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 → 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 $t \in \mathcal{T}(\Sigma)$.

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

regular languages → 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.

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

regular languages $\rightarrow$ WSkS languages

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

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

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

regular languages $\to$ WSkS languages

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

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

$\phi_{\mathsf{acc}}(\overline{Y})$: 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

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

regular languages $\to$ 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

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

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

Plan

WSkS: Definition

Automata $\rightarrow$ Logic

Logic $\rightarrow$ Automata

Fragments and Extensions of WSkS

Theorem Thatcher & Wright

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 $\mathcal{A}_\phi$.

Theorem Thatcher & Wright

$\mathcal{A}_\phi$ 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.

Theorem Thatcher & Wright

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.

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

WSkS$_0$

We consider a simplified language (wlog).

- no first order variables,
- only second order variables $X, Y \ldots$,
- form $::=$ $X \subseteq Y \mid Y = X \cdot 1 \mid \ldots \mid Y = X \cdot k$
  $\mid X \subseteq L_a \quad a \in \Sigma$
  $\mid$ form $\wedge$ form $\mid$ form $\vee$ form $\mid \neg$form
  $\mid \exists X$ form $\mid \forall X$ form

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

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

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:

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

For $m \geq 1$,

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

compilation: Boolean connectors

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

Theorem Thatcher & Wright

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 $\alpha_1, \dots, \alpha_n \in dom(\mathfrak{A})$, $\beta_1, \dots, \beta_n \in dom(\mathfrak{B})$, $m \geq 0$,

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

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

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

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

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

FO $\subsetneq$ WSkS

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

Lemma :

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

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,

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

Grid $G : \mathbb{N} \times \mathbb{N} \rightarrow \Sigma$; 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