Tree Automata and Tree Constraints

Ralf Treinen
Laboratoire de Recherche en Informatique
Université Paris-Sud
Orsay, France

In parts joint works with: Alexander Aiken, Martin Müller, Joachim Niehren, Tim Priesnitz, Zhendong Su.

Plan of this Talk

1. Tree Constraints

2. Solving Tree Constraints by Syntactic Methods

3. Solving Monadic Tree Constraints by Automata

4. Solving Tree Constraints by Tree Automata

(Tree) Constraints

A constraint system is given by

- a first-order signature $\langle \Sigma, P \rangle$
- a class of 1st-order formulas over this signature
- a fixed interpretation (with universe $= T(\Sigma)$)

Questions:

- Validity of constraints
- How to simplify constraints
- $\ldots$

Usage of Tree Constraints

- Constraint Logic Programming, Constraint Programming
- Computational Linguistics
- Constrained-Based Deduction
- Constraint-Based Program Analysis
- Constraint-Based Anything
- ...

Constraint-Based Theorem Proving

One example: ordered rewriting.

$$\left\{\begin{array}{rcl} (x \times y) \times z & = & x \times (y \times z) \\ x \times 1 & = & x \\ x \times x^{-1} & = & 1 \end{array}\right.$$

Is an equation like

$$(x \times y)^{-1} = y^{-1} \times x^{-1}$$

a consequence of these axioms?

Proof be rewriting

Idea: Turn the equations into a terminating rewrite system, like

$$x \times 1 \quad \rightarrow \quad x$$

Normalize both sides of an equation and test for syntactic equality.

Problem: In general, completeness is lost but can be regained by adding new rewrite rules (Knuth-Bendix Completion).

Non-orientable equations

$$x \times y = y \times x$$

The corresponding rewrite rule is looping:

$$x \times y \to y \times x$$

Constrained rewrite rule

$$x \times y \to y \times x \mid x > y$$

To be read: Rewrite ground term $s \times t$ into $t \times s$ only if $s > t$.

[Kirchner & Kirchner & Rusinowitch '90, ...]

In general, an equation $l = r$ yields two constrained rewrite rules:

$$\begin{aligned} l &\rightarrow r \mid l > r \\ r &\rightarrow l \mid r > l \end{aligned}$$

where $>$ is an ordering relation that is

1. total on ground terms

2. well-founded

3. monotonic

Examples of such orderings: recursive path ordering, lexicographic path ordering, Knuth-Bendix Ordering, ...

We may use a constraint system with

- Signature $\langle \Sigma, \{\leq\} \rangle$
- Interpretation of $\leq$: The chosen ordering (lexicographic path ordering, Knuth-Bendix Ordering, $\ldots$)
- Class of constraints: depends on the kind of problems one wants to solve (see next slides), includes at least atomic inequations $s < t$, closed under $\exists$ and $\wedge$

Consider constrained rewrite rules

$$l \to r \mid c(\bar{x})$$

where $\bar{x} = \mathit{free}(l) \cup \mathit{free}(r) \cup \mathit{free}(c)$.

$\exists^*$-Formulas as Constraints

To complete the rewrite system, we have to know whether two rules can overlap:

$$\begin{array}{ll} (R1) & s \to t \mid c(\bar{x}) \\ (R2) & u \to v \mid d(\bar{y}) \end{array}$$

overlap at the top when the following constraint is satisfiable:

$$s = u \wedge c(\bar{x}) \wedge d(\bar{y})$$

that is if

$$\exists \bar{x}, \bar{y}\; s = u \wedge c(\bar{x}) \wedge d(\bar{y})$$

is valid.

$\forall^*\exists^*$-Formulas as Constraints

$$\begin{array}{ll} (R1) & s \to t \mid c(\bar{x}) \\ (R2) & u \to v \mid d(\bar{y}) \end{array}$$

$(R2)$ subsumes $(R1)$ iff

$$\forall \bar{x}\Big(c(\bar{x}) \to \exists \bar{y}(s = u \wedge t = v \wedge d(\bar{y}))\Big)$$

Consequence: We need $\forall^*\exists^*$-formulas to decide subsumption.

Arbitrary Formulas as Constraints

$$\begin{array}{ll} (R1) & s \to t \mid c(\bar{x}) \\ (R2) & u \to v \mid d(\bar{y}) \end{array}$$

To make both rules disjoint replace $d(\bar{y})$ by

$$d(\bar{y}) \wedge \neg \exists \bar{x} \Big(c(\bar{x}) \wedge s = u \wedge t = v \Big)$$

Consequence: Constraints can be any FOL formula

Solving Tree Constraints by Syntactic Methods

Design a set of transformation rules that

- preserves the semantics of formulas
- is terminating
- has normal forms which allow to read off the answer

Example: Solving equational constraints

$$\frac{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)}{s_1 = t_1 \wedge \ldots \wedge s_n = t_n}$$

Example: Quantifier elimination

$$\frac{\exists x \; x = t \wedge \phi}{\phi[x \mapsto t]}$$

Problem: Sometimes difficult to find.

Using Automata to Solve Tree Constraints

Running example: Non-Structural Subtyping

- Signature: $\langle \Sigma, \leq \rangle$
- Interpretation of $s \leq t$: For all $\pi \in O(s) \cap O(t)$ we have $s(\pi) \leq t(\pi)$ for some fixed order on Σ.
- Class of constraints: Let's see how far we get ...

Example Σ: $\bot/0 < f/2 < g/2 < \top/0$

Properties of Non-Structural Subtyping

- complete lattice
- minimum: $\perp$
- maximum: $\top$
- $f(s_1, s_2) \leq f(t_1, t_2) \Leftrightarrow s_1 \leq t_1 \wedge s_2 \leq t_2$

Old and New Results on NSS

- Satisfiability of constraints $\exists \bar{x}\ c_1 \wedge \ldots \wedge c_n$

  Decidable in n^3

  [Rehoff; Palsberg & Wand & O'Keefe]

- Entailment of constraints

  $\forall \bar{x}\ c_1 \wedge \ldots \wedge c_n \to c$

  (Still) open question

- Full first-order theory
  - Monadic signature: decidable
  - In general: undecidable

  [Aiken et al. POPL'02]

Simplification: Monadic Signatures

Monadic signature: only constants and unary function symbols.

A tree over a monadic signature Σ can be represented as a word over Σ.

Example: Σ: $\perp/0 < f/1 < g/1 < \top/0$

$$\begin{array}{c} f \\ | \\ g \\ | \\ \perp \end{array}$$

is represented by the word

$$f g \perp$$

An n-tuple of trees over a monadic signature Σ can be represented as a word over $(\Sigma \cup \{\#\})^n$

Example: Σ: $\bot/0 < f/1 < g/1 < \top/0$

$$\begin{array}{ccc} f & \top & g \\ | & & | \\ g & & \top \\ | & & \\ \bot & & \end{array}$$

is represented by the word

$$\begin{pmatrix} f \\ \top \\ g \end{pmatrix} \begin{pmatrix} g \\ \# \\ \top \end{pmatrix} \begin{pmatrix} \bot \\ \# \\ \# \end{pmatrix}$$

Coding function $\gamma : T(\Sigma) \mapsto (\Sigma \cup \{\#\})^n$

Basic Idea

A solution of a constraint $\phi(x_1, \ldots, x_n)$ is a n-tuple of trees over Σ

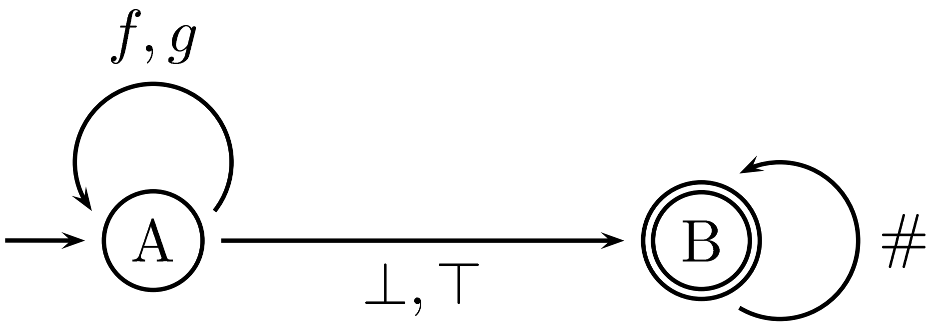
Try to build, for every formula $\phi(x_1, \ldots, x_n)$, an automaton A_ϕ with

$$L(A_\phi) = \{\gamma(t_1, \ldots, t_n) \mid\models \phi[t_1, \ldots, t_n]\}$$

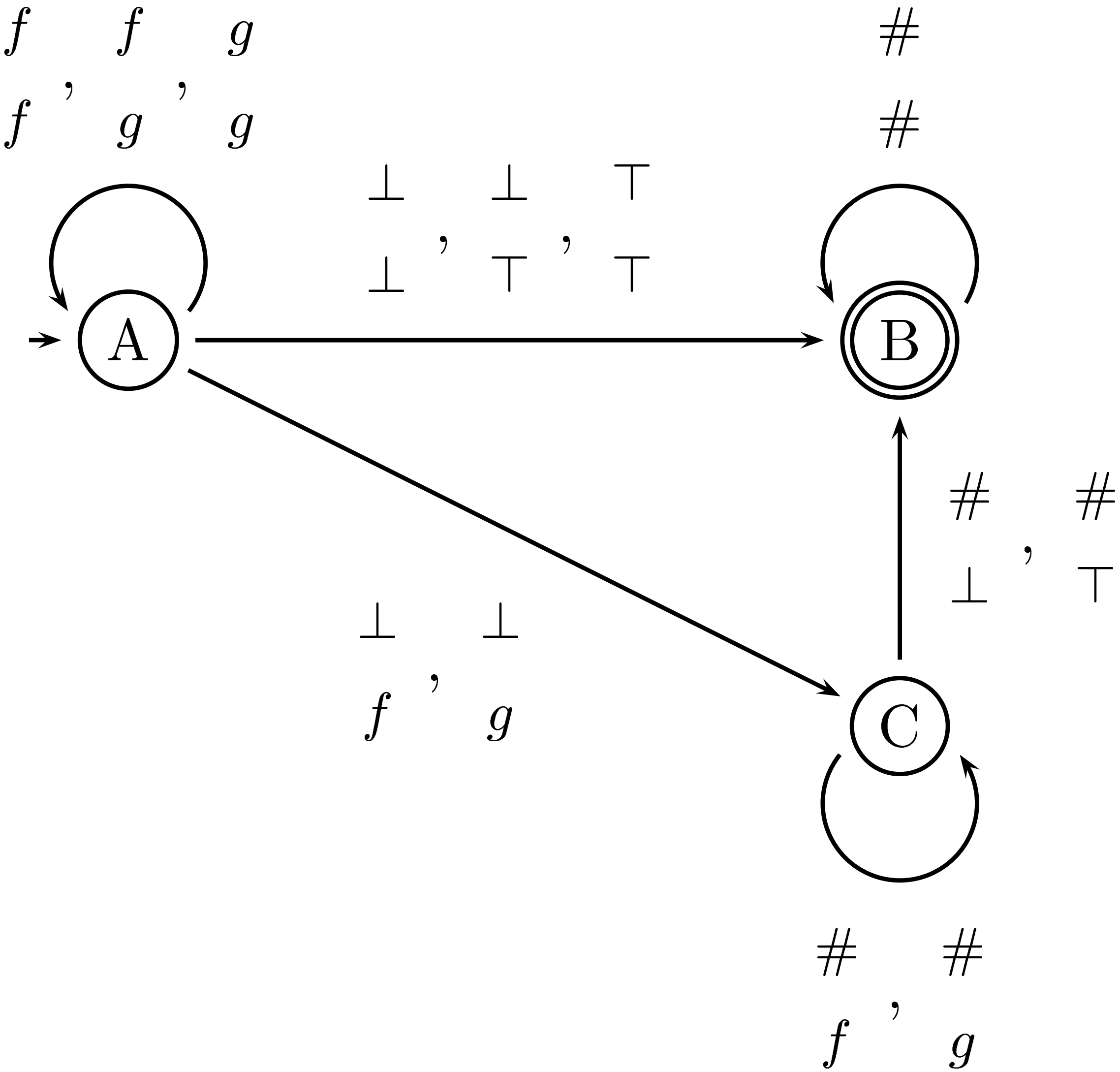
If ϕ is closed: ϕ is valid iff $L(A_\phi) \neq \emptyset$.

Idea goes back to [Buechi '60]

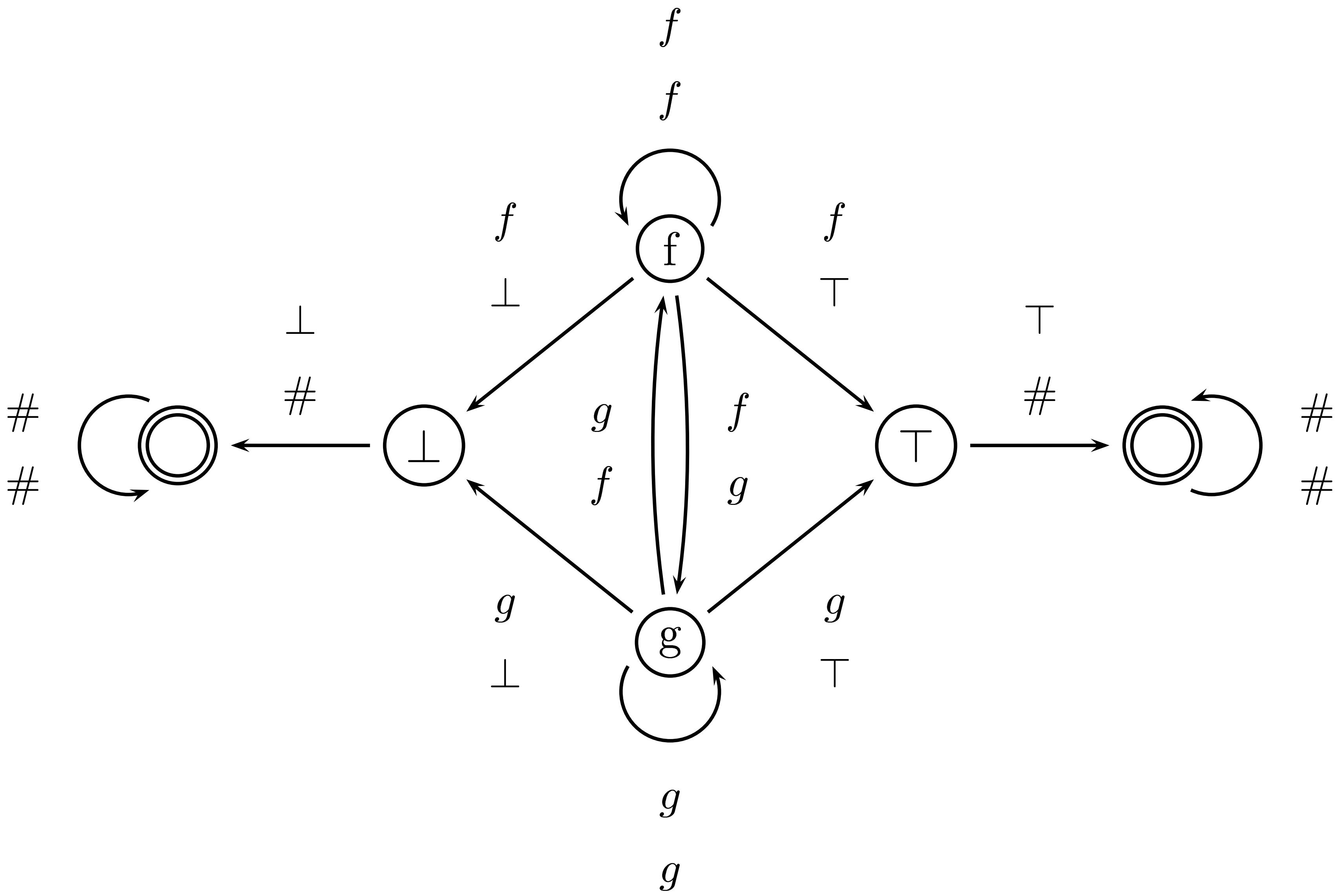
An automaton that recognizes $\gamma(T(\Sigma))$



An automaton for the solutions of $x_1 \leq x_2$



An automaton for the solutions of $x_1 = f(x_2)$



Building automata for composite formulas

Inductive construction using the closure properties of automata:

- conjunction $\Rightarrow$ intersection
- negation $\Rightarrow$ complement
  (using determinisation)
- existential quantification $\Rightarrow$ projection

Plus cylindrification:

To build the conjunction of $\Phi(X_1, X_2)$ and $\Psi(X_2, X_3)$

- Build A_Φ and A_Ψ
- Cylindrify to the variables (X_1, X_2, X_3)
- Intersection automaton

Tree Automata: Definition

[Doner, Thatcher, Wright: 1965-1970]

A tree automaton (TA) is given by

- a tree signature Σ
- a finite set Q of states
- a set $F \subseteq Q$ of final states
- a set Δ of transition rules of the form

$$f(q_1, q_2) \rightarrow q$$

Tree Automata: Example

- $\Sigma = \{0/0, s/1, nil/0, cons/2\}$
- $Q = \{N, L\}$
- $F = \{L\}$
- $\Delta = \left\{ \begin{array}{llll} 0 & \to N & s(N) & \to N \\ nil & \to L & cons(L, N) & \to L \end{array} \right\}$

The move relation $\to_A$ of an automaton A is the smallest relation with

- $a \to_A q(a)$ if $q \to a \in \Delta$
- if $t_1 \to_A q_1(t_1)$

  $t_2 \to_A q_2(t_2)$

  $f(q_1, q_2) \to q \in \Delta$

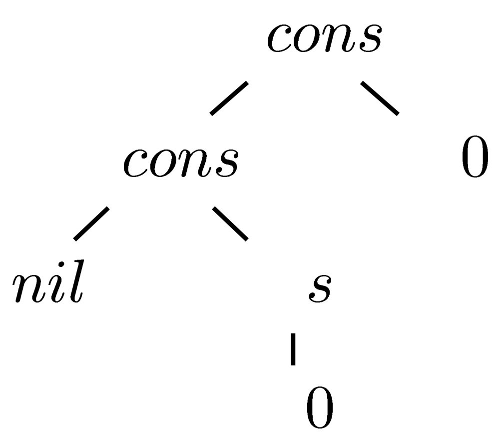
  then $f(t_1, t_2) \to_A q(f(t_1, t_2))$

The automaton A accepts the tree t if $t \to_A q(t)$ for some $q \in F$.

The language recognized by A is

$$L(A) = \{t \mid A \text{ accepts } t\}$$

Run of a Tree Automaton: Example



Transition rules:

$$\begin{array}{llll} 0 & \to N & s(N) & \to N \\ nil & \to L & cons(L,N) & \to L \end{array}$$

$$L(\quad\quad)$$

$$L(\quad\quad) \qquad N(\ )$$

$$L(\quad) \qquad N(\ )$$

$$N(\ )$$

Properties of Tree Automata

Tree automata enjoy all nice properties of word automata:

- closure under $\cup$, $\cap$, complement, linear homomorphisms, inverse homomorphisms, cylindrification, projection.
- decidability of emptiness
- determinisation
- minimisation

A Tree Automaton for $\gamma(T(\Sigma))$

$\Sigma = \{\perp/0, \top/0, f/2\}$

$$\begin{array}{rcl}
\# & \rightarrow & B \\
\#(B, B) & \rightarrow & B \\
\perp & \rightarrow & A \\
\perp(B, B) & \rightarrow & A \\
\top & \rightarrow & A \\
\top(B, B) & \rightarrow & A \\
f(A, A) & \rightarrow & A
\end{array}$$

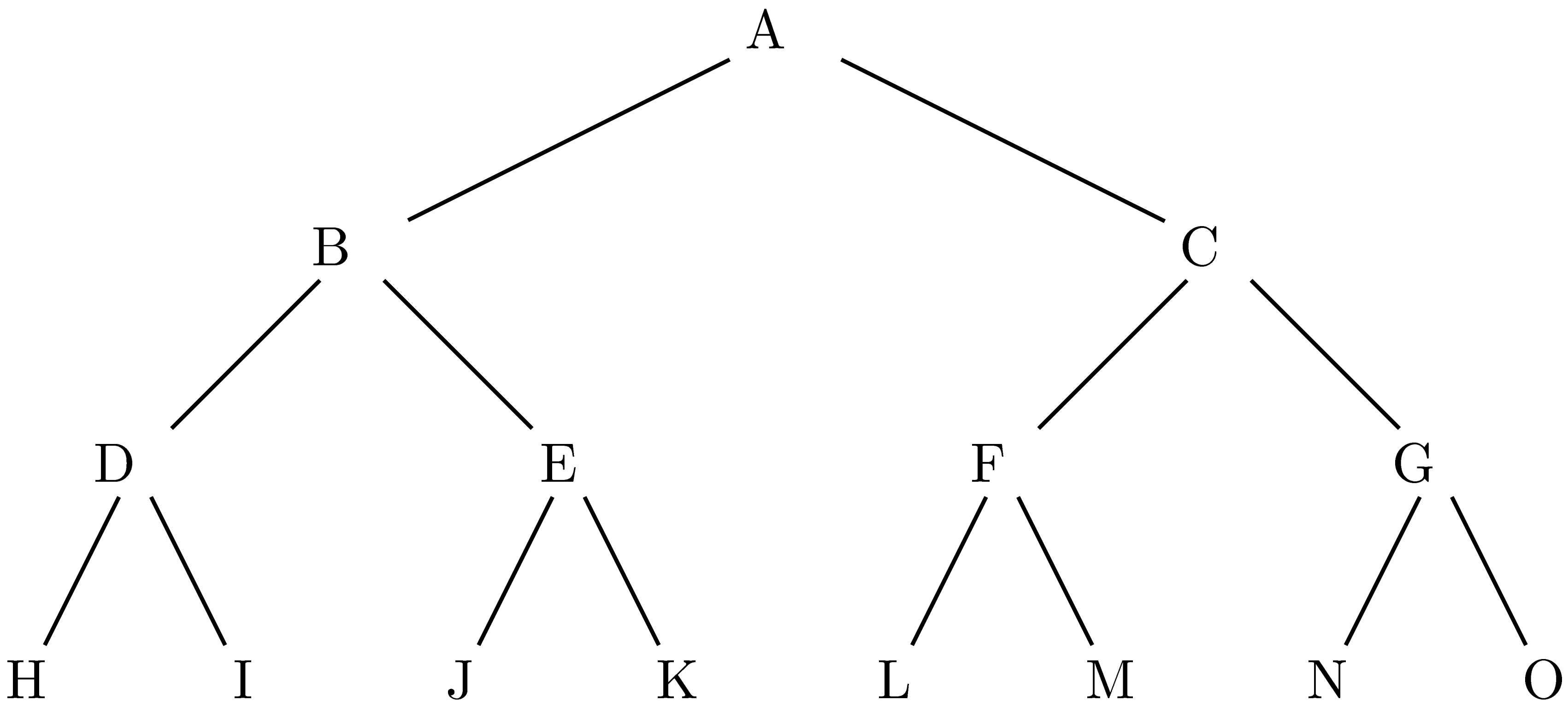
Accepting states: $\{A\}$.

A Tree Automaton for the Solutions of $x_1 \leq x_2$

Intersection of three automata:

1. check whether first component in $\gamma(T(\Sigma))$
2. check whether second component in $\gamma(T(\Sigma))$
3. check that no "bad" symbol occurs: $(f, \perp)$, $(\top, f)$, $(\top, \perp)$

A Tree Automaton for $x_1 = f(x_2, \_)$?



B

D E

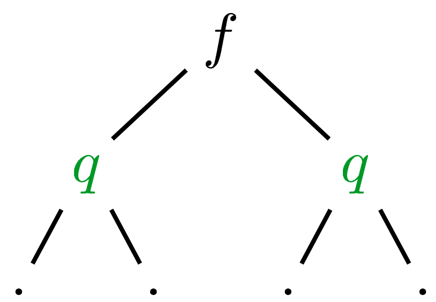
H I J K

Tree Automata with Tests (TAT)

[Mongy and Dauchet, early 80's]: Have transitions with tests.

Example:

$$f(q,q) \rightarrow^{12=21} q$$



Emptiness undecidable even when only comparisons between cousin positions [Seynhaeve et. al. 97]

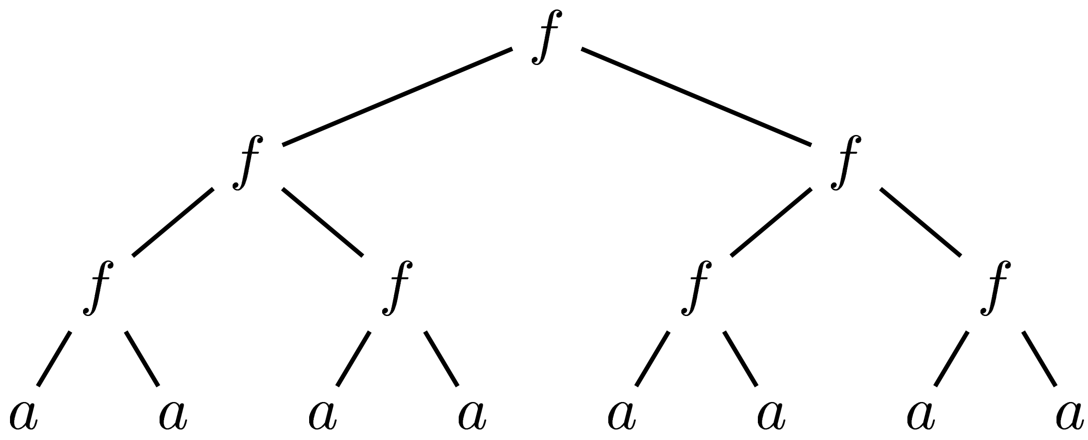
Restriction to tree automata with tests between brothers [Bogaert & Tison 92]:

- Closure under Boolean operations.
- Emptiness decidable.

Example:

$$a \to q \quad f(q,q) \to^{1=2} q$$

defines the complete binary trees.



Restriction to tree automata with a bounded number of tests on any branch (Reduction Automata) [Caron & Comon & Coquidé & Dauchet & Jacquemard '94]

- Closure under $\cup$, $\cap$ (but not under complement).
- Emptiness decidable.

Example: Reduction automaton for the set of terms that are reducible by some (non-linear) rewrite system.

Tree Automata with Component-wise Tests (TACT)

If we want to translate any logical formula of a logic into automata then we need closure under cylindrification.

TACT: Tests on Components:

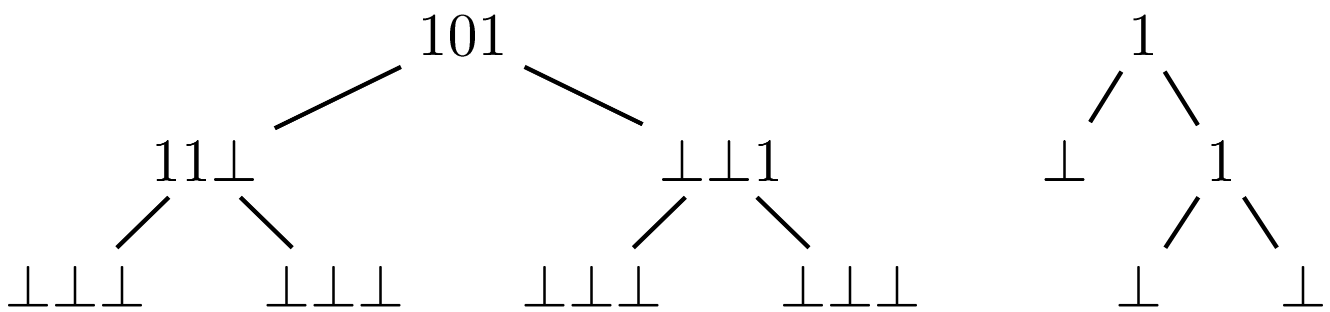
$$f(q_1, q_2) \rightarrow^{1.i=2.i} q$$

In general: any boolean combination of tests:

$$f(q_1, q_2) \rightarrow^{1.i=2.i \vee (1.j \neq 2.j \wedge 1.k=2.k)} q$$

Component of a Tree

Example 3rd component:



Results on Tree Automata with Component-Wise Tests

- Emptiness undecidable in case of
  - Tests between brothers [RT, FOSSACS'00],
  - One test, and this test is at the root.
- Not closed under arbitrary projection.

Conclusion

- Monadic case: Automata work well.
- General case: Problem for automata to represent equations.

Possible solution?

- Find a tree automata model which
  - is strong enough to model constraints
  - emptiness decidable
- Translate problems on constraints into Word automata [Müller et. al. LICS'98]