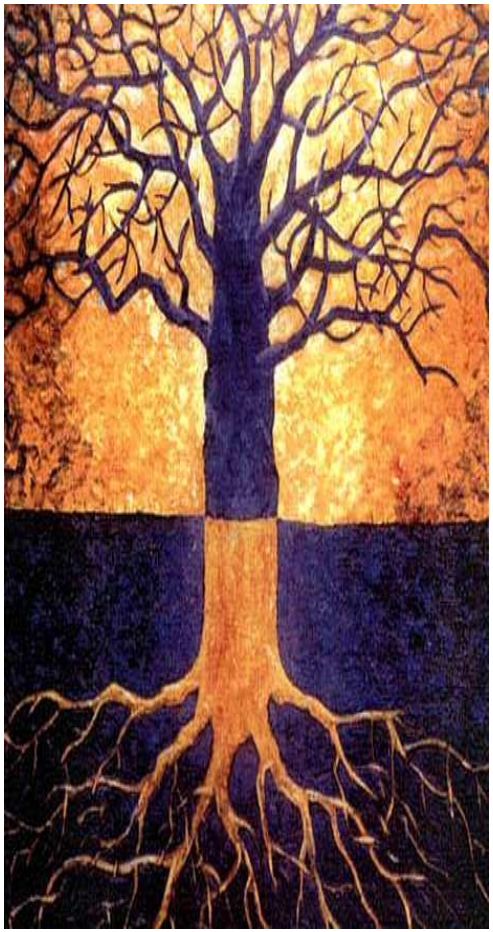


Tree Automata Techniques and Applications

Emmanuel Filiot University of Lille1, LIFL, Mostrare Project, now ULB
Jean-Marc Talbot University of Provence, LIF, Marseille
Sophie Tison University of Lille1, LIFL, Mostrare Project

Workshop in Honour of Hubert Comon-Lundh

Why this title?

Tree Automata Techniques and Applications

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

Why this title?

Without Hubert, this book wouldn't exist.

Why this title?

Without Hubert, this book wouldn't exist.

Most important, Hubert brought many answers to:

Why this title?

Without Hubert, this book wouldn't exist.

Most important, Hubert brought many answers to:

> How tree automata can be applied (to rewriting, verification, ...)?

Why this title?

Without Hubert, this book wouldn't exist.

Most important, Hubert brought many answers to:

> How tree automata can be applied (to rewriting, verification, ...)?
>
> How applications can make evolving tree automata, or how to develop an "application-driven automata theory"?

A more convenient title to this talk:

One aspect of Tree Automata Techniques :
How to ensure Equality and respect Differences in Tree Automata?

A more convenient title to this talk:

One aspect of Tree Automata Techniques :
How to ensure Equality and respect Differences in Tree Automata?

A short and not complete survey,
A few words about Tree Automata with Global Equalities and Differences,
TAGED

A more convenient title to this talk:

One aspect of Tree Automata Techniques :
How to ensure Equality and respect Differences in Tree Automata?

A short and not complete survey,
A few words about Tree Automata with Global Equalities and Differences,
TAGED

Emmanuel Filiot — University of Lille1, LIFL, INRIA LNE, now ULB (Brussels)
Jean-Marc Talbot — University of Provence, LIF, Marseille
Sophie Tison — University of Lille1, LIFL, INRIA LNE

Outline

Objective: Add to tree automata constraints to ensure Equality and Differences of subterms?
Two approaches:

1. LOCAL tests : only close subterms will be compared

Outline

Objective: Add to tree automata constraints to ensure Equality and Differences of subterms?
Two approaches:

1. LOCAL tests : only close subterms will be compared
2. GLOBAL tests: any pair of subterms can be compared

Bottom-up Tree Automata (binary case)

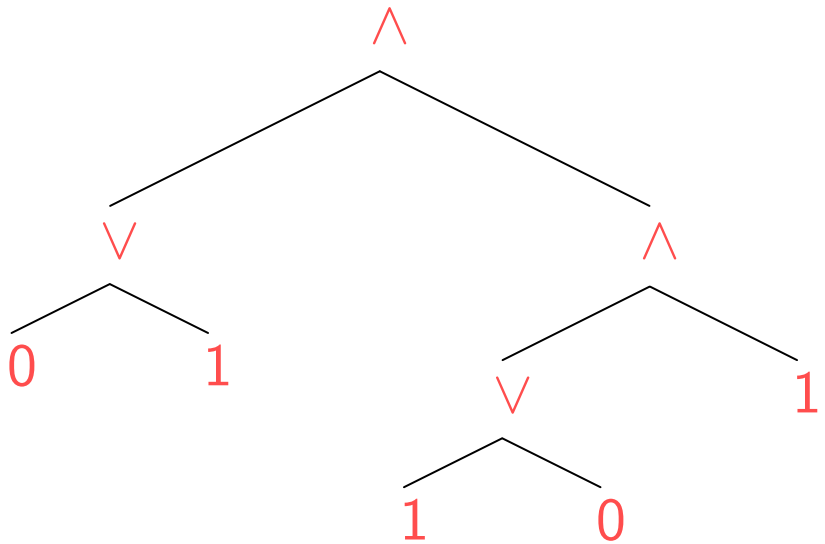
- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

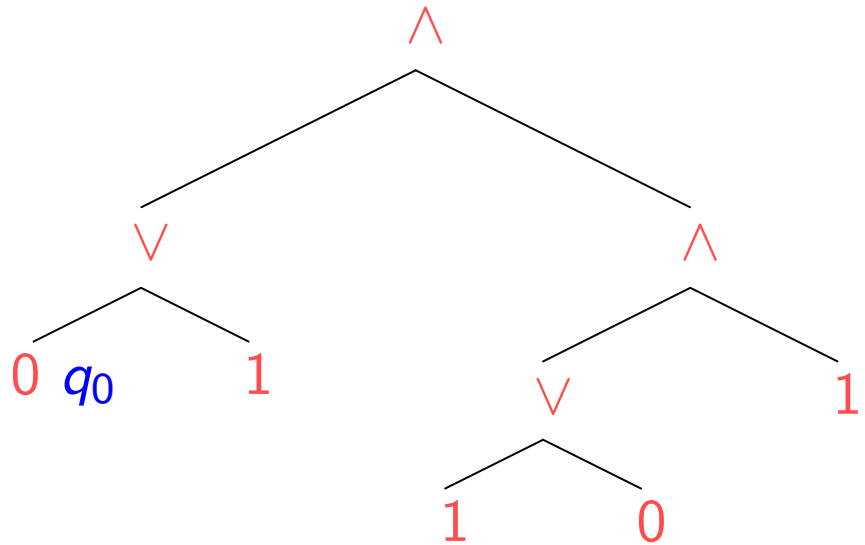
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

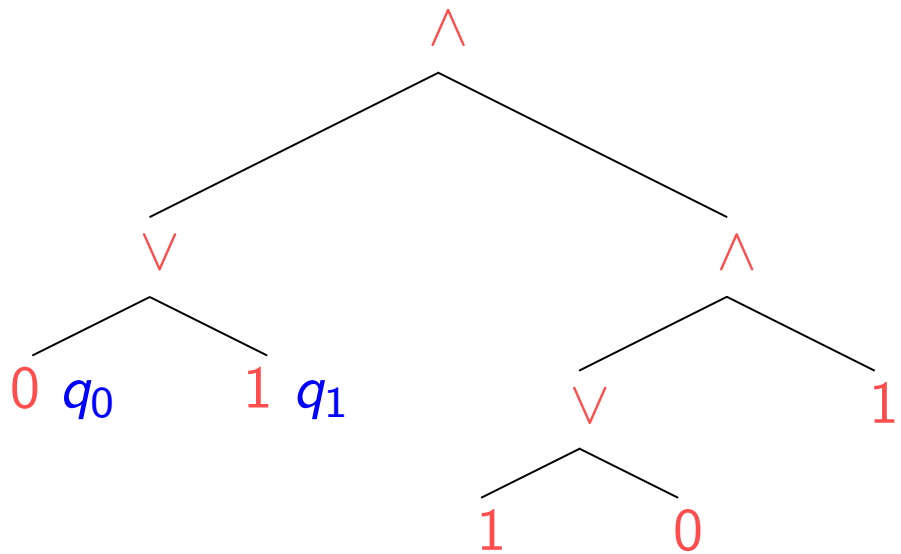
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$$
$$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$$
$$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$$

final states

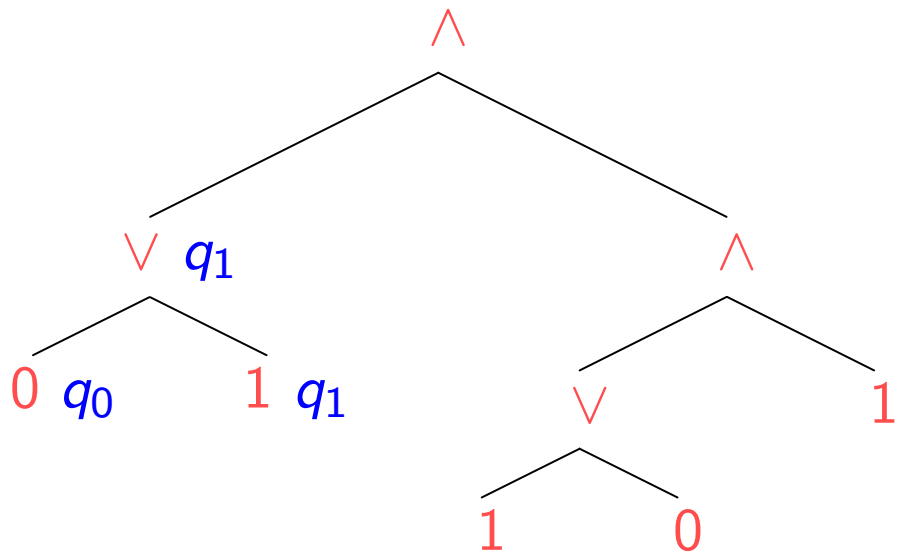
$$F = \{q_1\}$$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$$
$$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$$
$$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$$

final states

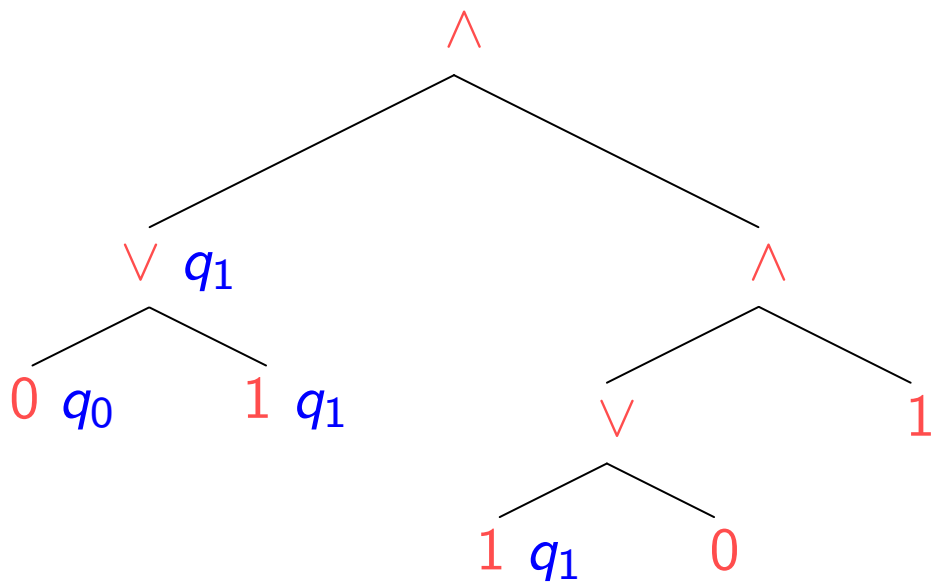
$$F = \{q_1\}$$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

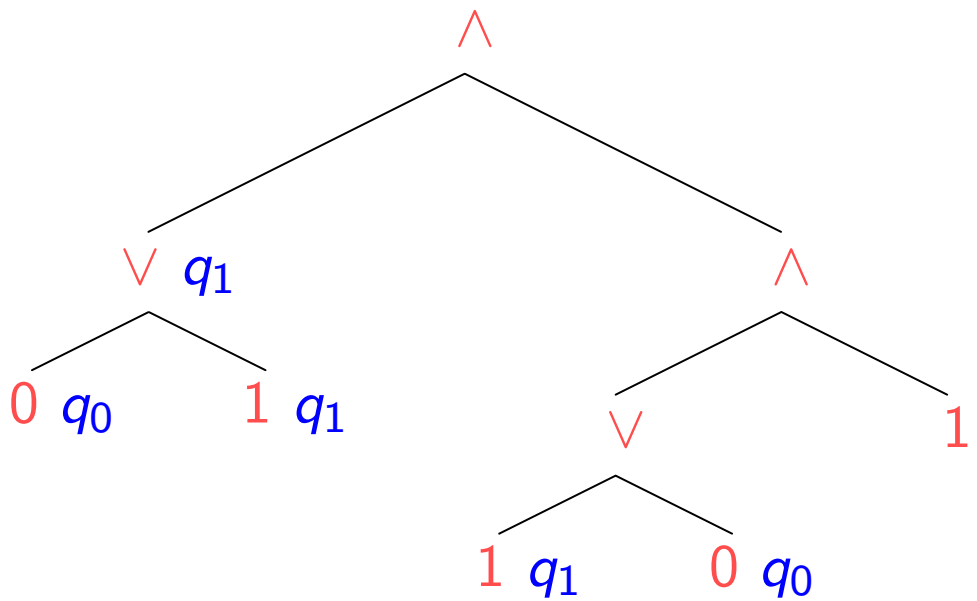
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

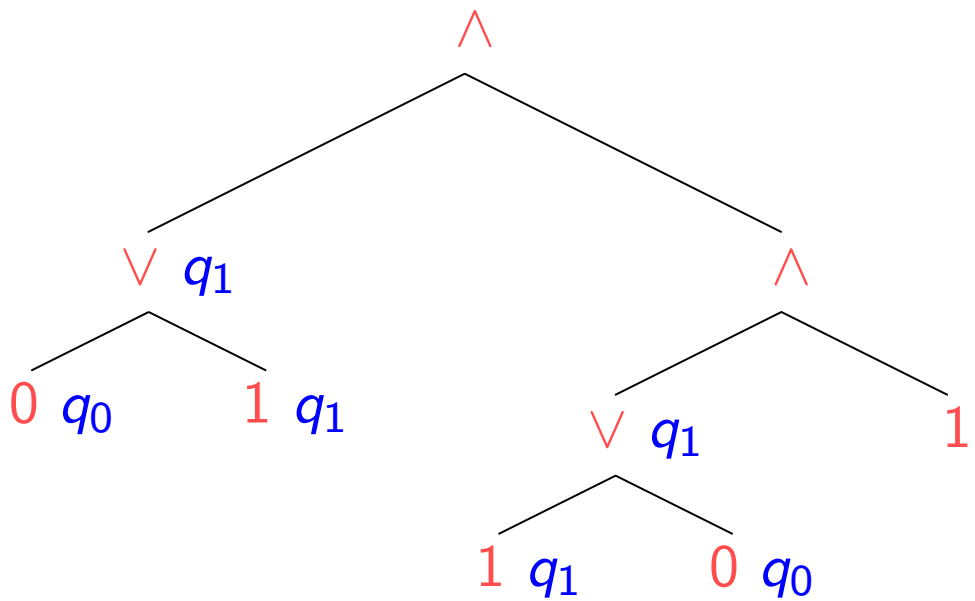
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

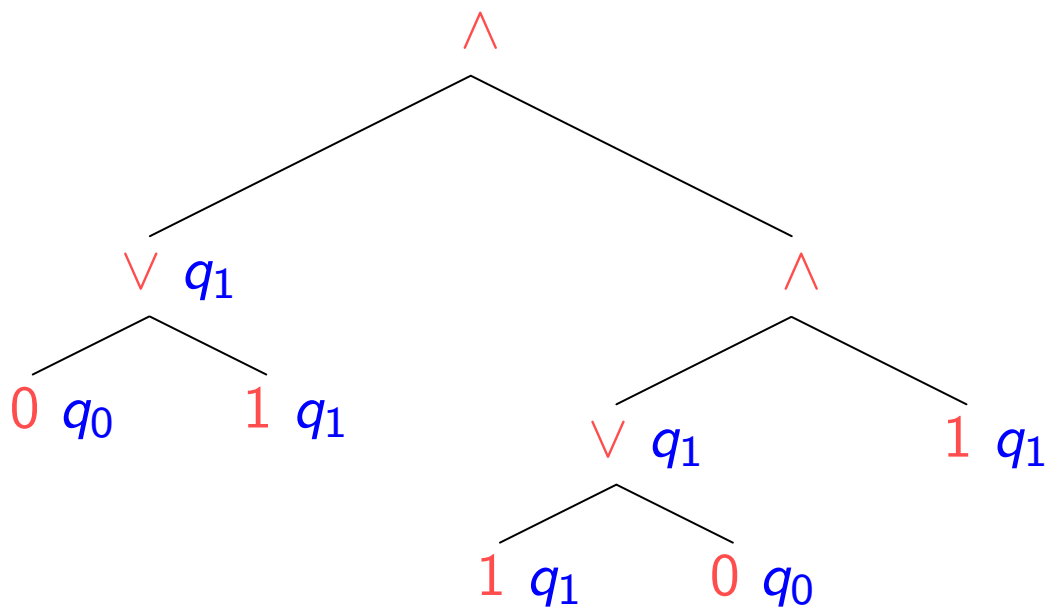
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

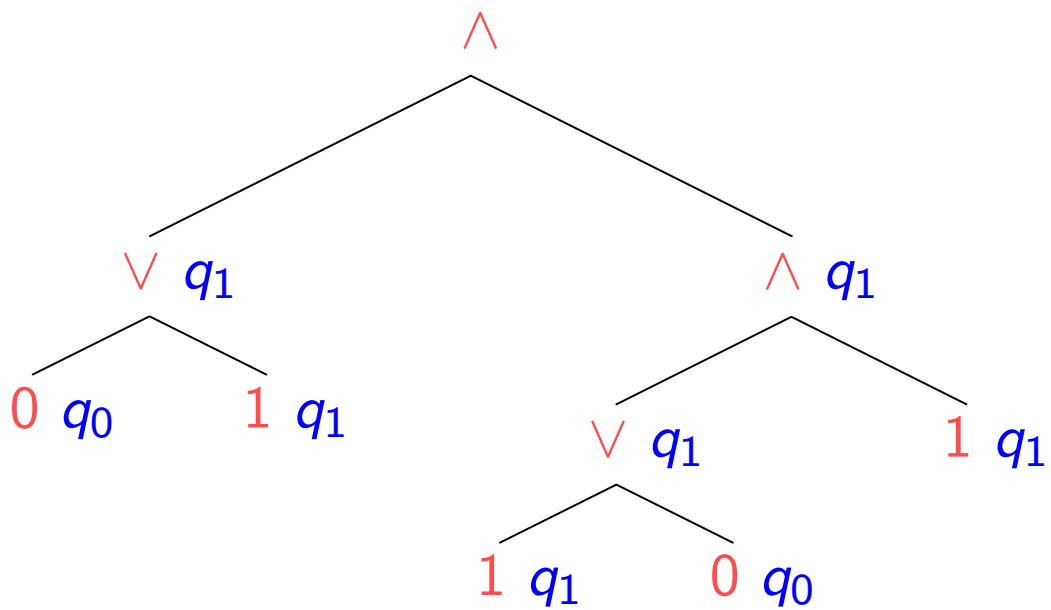
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

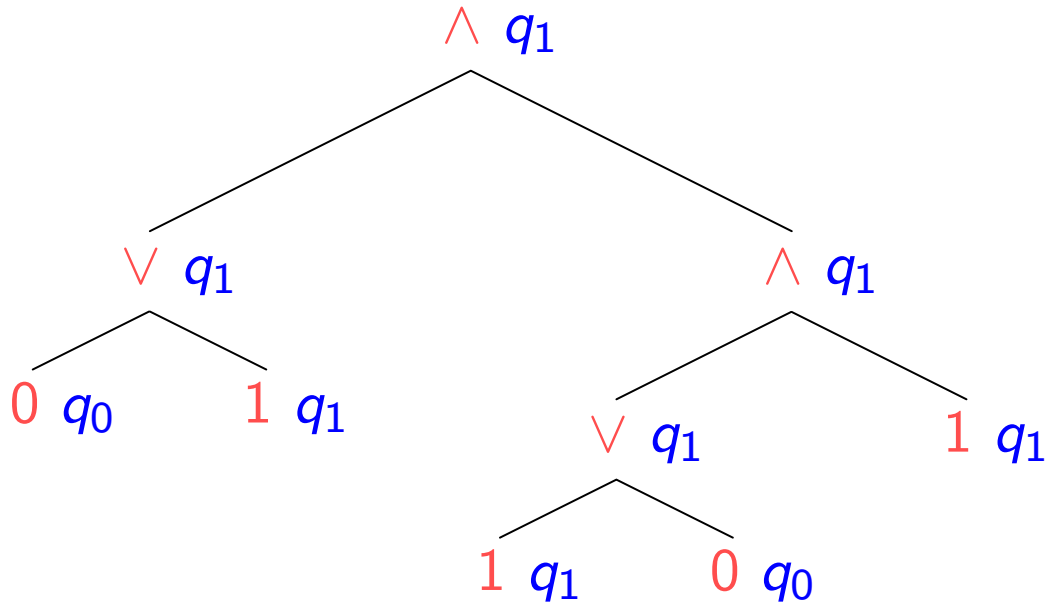
$F = \{q_1\}$

Bottom-up Tree Automata (binary case)

- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

$F = \{q_1\}$

Automata and equality and difference tests: the beginning of the story

A typical example of a language which is not recognized by a finite tree automaton is the set of terms $\{f(t,t)\}$, i.e. the set of ground instances of the term $f(x,x)$.

Automata and equality and difference tests: the beginning of the story

A typical example of a language which is not recognized by a finite tree automaton is the set of terms $\{f(t,t)\}$, i.e. the set of ground instances of the term $f(x,x)$.

How add expressivity to automata while keeping good properties?

Automata and equality and difference tests: the beginning of the story

A typical example of a language which is not recognized by a finite tree automaton is the set of terms $\{f(t,t)\}$, i.e. the set of ground instances of the term $f(x,x)$.

How add expressivity to automata while keeping good properties?

At least, we would like to be able to test equality or disequality of subterms.

Automata and equality and difference tests: the beginning of the story

A typical example of a language which is not recognized by a finite tree automaton is the set of terms $\{f(t,t)\}$, i.e. the set of ground instances of the term $f(x,x)$.

How add expressivity to automata while keeping good properties?

At least, we would like to be able to test equality or disequality of subterms.

Of course, we would like to have good decidabilty and closure properties.

Automata and tests: a first attempt

AWEDC, Tree Automata With Equality and Difference Constraints have been introduced by Max Dauchet and Jocelyne Mongy (Mongy's PhD 81):

Automata and tests: a first attempt

AWEDC, Tree Automata With Equality and Difference Constraints have been introduced by Max Dauchet and Jocelyne Mongy (Mongy's PhD 81):

Rules have the following form:

$$f(q_1, q_2) \rightarrow_{1.2=2,\ 1.2\neq 1.3} q$$

Automata and tests: a first attempt

AWEDC, Tree Automata With Equality and Difference Constraints have been introduced by Max Dauchet and Jocelyne Mongy (Mongy's PhD 81):

Rules have the following form:

$$f(q_1, q_2) \rightarrow_{1.2=2,\ 1.2\neq 1.3} q$$

The class *AWEDC* is rather expressive, has good closure properties but emptiness is undecidable.

Automata and Equality tests: Ten Years After

Rewriting gave new motivation for adding constraints to tree automata, e.g. in order to describe the set of ground irreducible terms.

Automata and Equality tests: Ten Years After

Rewriting gave new motivation for adding constraints to tree automata, e.g. in order to describe the set of ground irreducible terms.

Two main subclasses of *AWEDC* are defined

Automata and Equality tests: Ten Years After

Rewriting gave new motivation for adding constraints to tree automata, e.g. in order to describe the set of ground irreducible terms.

Two main subclasses of *AWEDC* are defined

A very simple one: Automata With Constraints Between Brothers

A more powerful one: Reduction Automata

A very simple class: Automata With Constraints Between Brothers

Constraints are reduced to constraints between brothers (siblings).
$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1\neq 3} q$ is allowed.

A very simple class: Automata With Constraints Between Brothers

Constraints are reduced to constraints between brothers (siblings).

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1\neq 3} q$ is allowed.

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1=3.1} q$ is forbidden.

A very simple class: Automata With Constraints Between Brothers

Constraints are reduced to constraints between brothers (siblings).

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1\neq 3} q$ is allowed.

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1=3.1} q$ is forbidden.

E.g., to recognize the set of well-balanced trees on $f(,), a$:

$$\begin{array}{rcl} a & \rightarrow & q \\ f(q,q) & \rightarrow_{1=2} & q \end{array}$$

Everything goes rather well.

A very simple class: Automata With Constraints Between Brothers

Constraints are reduced to constraints between brothers (siblings).

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1\neq 3} q$ is allowed.

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1=3.1} q$ is forbidden.

E.g., to recognize the set of well-balanced trees on $f(,), a$:

$$\begin{array}{rcl} a & \rightarrow & q \\ f(q,q) & \rightarrow_{1=2} & q \end{array}$$

Everything goes rather well.

Emptiness is decidable but EXPTIME-complete for non deterministic automata.

A very simple class: Automata With Constraints Between Brothers

Constraints are reduced to constraints between brothers (siblings).

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1\neq 3} q$ is allowed.

$f(q_1, q_2, q_3) \rightarrow_{1=2,\ 1=3.1} q$ is forbidden.

E.g., to recognize the set of well-balanced trees on $f(,), a$:

$$\begin{array}{rcl} a & \rightarrow & q \\ f(q,q) & \rightarrow_{1=2} & q \end{array}$$

Everything goes rather well.

Emptiness is decidable but EXPTIME-complete for non deterministic automata.

The class is rather restricted!

Some examples of applications to rewriting

If a term rewrite system is shallow and left-linear, its sets of normal forms is recognizable .

Some examples of applications to rewriting

If a term rewrite system is shallow , its sets of normal forms is recognizable by an Automata With Constraints Between Brothers.

Some examples of applications to rewriting

If a term rewrite system is shallow , its sets of normal forms is recognizable by an Automata With Constraints Between Brothers. E.g., it can be used to prove decidability of normalization for shallow right linear trs, (Godoy and alt. 2007)

Some examples of applications to rewriting

If a term rewrite system is shallow , its sets of normal forms is recognizable by an Automata With Constraints Between Brothers. E.g., it can be used to prove decidability of normalization for shallow right linear trs, (Godoy and alt. 2007)

The set of innermost-reachable terms from a TA language by a shallow TRS is not necessarily regular, but it can be recognized by a tree automaton with equality and disequality constraints between brothers. (Godoy and oth.)

Some examples of applications to rewriting

If a term rewrite system is shallow , its sets of normal forms is recognizable by an Automata With Constraints Between Brothers. E.g., it can be used to prove decidability of normalization for shallow right linear trs, (Godoy and alt. 2007)

The set of innermost-reachable terms from a TA language by a shallow TRS is not necessarily regular, but it can be recognized by a tree automaton with equality and disequality constraints between brothers. (Godoy and oth.) : it can be used to prove decidability of some properties of innermost rewriting for shallow TRS.

A more powerful class: Reduction Automata

Idea: Add a restriction in order to bound the number of equalities along a path.

A more powerful class: Reduction Automata

Idea: Add a restriction in order to bound the number of equalities along a path.

A *reduction automaton* $\mathcal{A}$ is a *AWEDC* such that there is an ordering on the states of $\mathcal{A}$ such that,

> *for each rule* $f(q_1, \ldots, q_n) \rightarrow_{\pi_1=\pi_2 \wedge c} q$, q *is strictly smaller than each* q_i.

Reduction Automata

Emptiness is decidable for deterministic ones and an efficient cleaning algorithm is given for emptiness decision test.(CaronComonCoquidéDauchetJacquemard, ICALP94).

Reduction Automata

Emptiness is decidable for deterministic ones and an efficient cleaning algorithm is given for emptiness decision test.(CaronComonCoquidéDauchetJacquemard, ICALP94).

E.g., this class allows to express set of ground normal forms of t.r.s. and leads to decidability of encompassment theory. This gives an other proof of decidability of ground reducibility.

Reduction Automata

Emptiness is decidable for deterministic ones and an efficient cleaning algorithm is given for emptiness decision test.(CaronComonCoquidéDauchetJacquemard, ICALP94).

E.g., this class allows to express set of ground normal forms of t.r.s. and leads to decidability of encompassment theory. This gives an other proof of decidability of ground reducibility.

Emptiness is undecidable for non deterministic ones (JacquemardRusinowitchVigneron IJCAR2006)

One extension: Tree automata with equality test modulo equational theories

In IJCAR2006, F. Jacquemard, M. Rusinowitch and L.Vigneron present new classes of tree automata combining automata with equality test modulo equational theories.

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

Wong Karianto and Christof Löding (ICALP 2007) use MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees. Example: The set of well-balanced trees over the alphabet a can be recognized by

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

Wong Karianto and Christof Löding (ICALP 2007) use MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees. Example: The set of well-balanced trees over the alphabet a can be recognized by $Q = F = \{q\}, a \rightarrow q(leaf), (a, Q^+, q, \forall_= true)$

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

Wong Karianto and Christof Löding (ICALP 2007) use MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees.
Example: The set of well-balanced trees over the alphabet a can be recognized by $Q = F = \{q\}, a \to q(\mathit{leaf}), (a, Q^+, q, \forall_= \mathit{true})$

Emptiness is decidable for the deterministic case

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

Wong Karianto and Christof Löding (ICALP 2007) use MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees.
Example: The set of well-balanced trees over the alphabet a can be recognized by $Q = F = \{q\}, a \rightarrow q(\mathit{leaf}), (a, Q^+, q, \forall_= \mathit{true})$

Emptiness is decidable for the deterministic case

Non determinism can't be reduced

One other extension: The Unranked Case

Unranked trees are essential for modeling XML and tree automata extend to the unranked case (hedge automata, stepwise automata, ...)

How extend tree automata with equality and disequality tests to the unranked case?

Wong Karianto and Christof Löding (ICALP 2007) use MSO-formulas to capture the possibility of comparing unboundedly many direct subtrees.
Example: The set of well-balanced trees over the alphabet a can be recognized by $Q = F = \{q\}, a \rightarrow q(leaf), (a, Q^+, q, \forall_= true)$

Emptiness is decidable for the deterministic case

Non determinism can't be reduced

Decidability of emptiness for the non deterministic case ?

Can we get rid of equality and differences constraints?

Problem (Reg-problem)

Is the language accepted by an automaton with equality and difference constraints recognizable?

Can we get rid of equality and differences constraints?

Problem (Reg-problem)

Is the language accepted by an automaton with equality and difference constraints recognizable?

In the general case: undecidable

Can we get rid of equality and differences constraints?

Problem (Reg-problem)

Is the language accepted by an automaton with equality and difference constraints recognizable?

In the general case: undecidable

In the case of tests between brothers: decidable

Can we get rid of equality and differences constraints?

Problem (Reg-problem)

Is the language accepted by an automaton with equality and difference constraints recognizable?

In the general case: undecidable

In the case of tests between brothers: decidable

For reduction automata?

Can we get rid of equality and differences constraints? A related problem

Problem (HOM-problem)

Can we decide whether the homomorphic image of a regular tree language is regular?

Can we get rid of equality and differences constraints? A related problem

Problem (HOM-problem)

Can we decide whether the homomorphic image of a regular tree language is regular?

Decidable for shallow homorphisms by using automata with tests between brothers.

Can we get rid of equality and differences constraints? A related problem

Problem (HOM-problem)

Can we decide whether the homomorphic image of a regular tree language is regular?

Decidable for shallow homorphisms by using automata with tests between brothers.

Unsolved in the general case

Can we get rid of equality and differences constraints? A related problem

Problem (HOM-problem)

Can we decide whether the homomorphic image of a regular tree language is regular?

Decidable for shallow homorphisms by using automata with tests between brothers.

Unsolved in the general case

Decidable for some restricted cases by reduction to the next problem:

Can we get rid of equality and differences constraints? An other related problem

Problem (FS-problem)

Is the language defined by a finite set of terms with regular constraints regular?

$L_1 = g^+(a), L_2 = g^*(a)$
$C : x_1 \longrightarrow L_1, x_2 \longrightarrow L_2$

Can we get rid of equality and differences constraints? An other related problem

Problem (FS-problem)

Is the language defined by a finite set of terms with regular constraints regular?

$L_1 = g^+(a), L_2 = g^*(a)$
$C : x_1 \rightarrow L_1, x_2 \rightarrow L_2$

$S_0 = \{f(x_1, x_2)\}$: $L(\langle S_0, C \rangle)$ is regular

Can we get rid of equality and differences constraints? An other related problem

Problem (FS-problem)

Is the language defined by a finite set of terms with regular constraints regular?

$L_1 = g^+(a), L_2 = g^*(a)$
$C : x_1 \rightarrow L_1, x_2 \rightarrow L_2$

$S_0 = \{f(x_1, x_2)\}$: $L(\langle S_0, C\rangle)$ is regular
$S_1 = \{f(x_2, x_2)\}$: $L(\langle S_1, C\rangle)$ is not regular

Can we get rid of equality and differences constraints? An other related problem

Problem (FS-problem)

Is the language defined by a finite set of terms with regular constraints regular?

$L_1 = g^+(a), L_2 = g^*(a)$
$C : x_1 \to L_1, x_2 \to L_2$

$S_0 = \{f(x_1, x_2)\}$: $L(\langle S_0, C\rangle)$ is regular

$S_1 = \{f(x_2, x_2)\}$: $L(\langle S_1, C\rangle)$ is not regular

$S_1 = \{f(x_1, x_2), f(x_2, x_2)\}$: $L(\langle S_1, C\rangle)$ is regular

Without regular constraints, decidability has been proved by Lassez and Marriot (87).

Can we get rid of equality and differences constraints? An other related problem

Problem (FS-problem)

Is the language defined by a finite set of terms with regular constraints regular?

$L_1 = g^+(a), L_2 = g^*(a)$
$C : x_1 \rightarrow L_1, x_2 \rightarrow L_2$

$S_0 = \{f(x_1, x_2)\}$: $L(\langle S_0, C\rangle)$ is regular

$S_1 = \{f(x_2, x_2)\}$: $L(\langle S_1, C\rangle)$ is not regular

$S_1 = \{f(x_1, x_2), f(x_2, x_2)\}$: $L(\langle S_1, C\rangle)$ is regular

Without regular constraints, decidability has been proved by Lassez and Marriot (87).

With regular constraints , decidability can be obtained by using equational formulae with membership constraints (Comon & Delor 94).

Automata and Equality tests: the global case

Automata and Equality tests: different approaches

Objective here: don't restrict equality/difference tests to "close" subterms.

- Tree automata with memory (X & Comon & Y)

Automata and Equality tests: different approaches

Objective here: don't restrict equality/difference tests to "close" subterms.

- Tree automata with memory (X & Comon & Y)
- Rigid Tree Automata (Jacquemard, Klay, Vacher)

Automata and Equality tests: different approaches

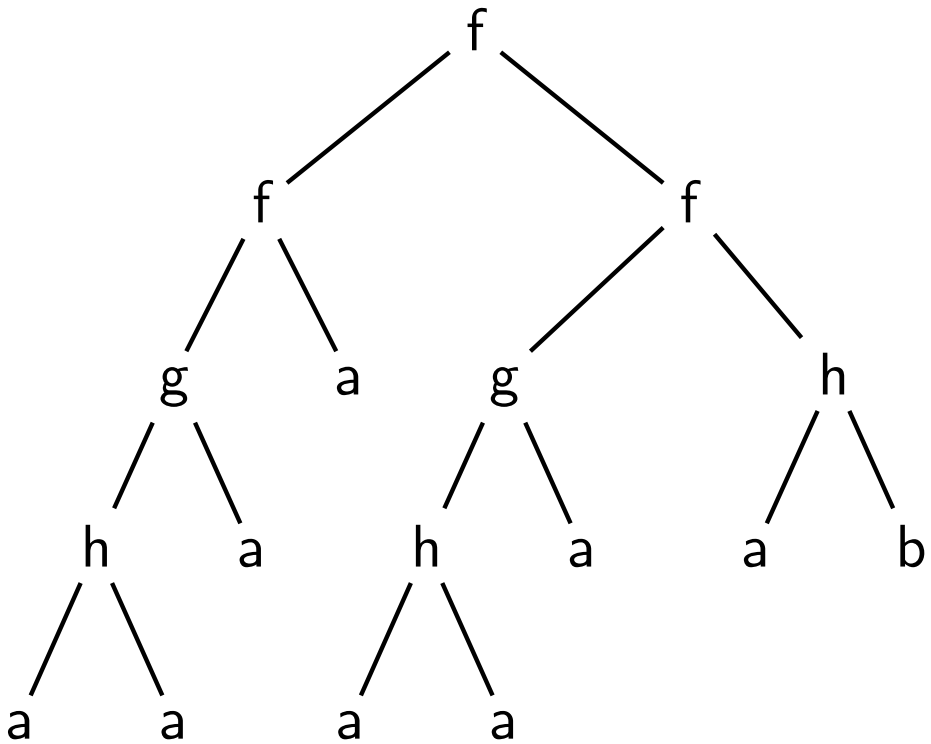
Objective here: don't restrict equality/difference tests to "close" subterms.

- Tree automata with memory (X & Comon & Y)
- Rigid Tree Automata (Jacquemard, Klay, Vacher)
- TAGED (Tree Automata with Global Equality and Disequality tests)

TAGED: Motivations

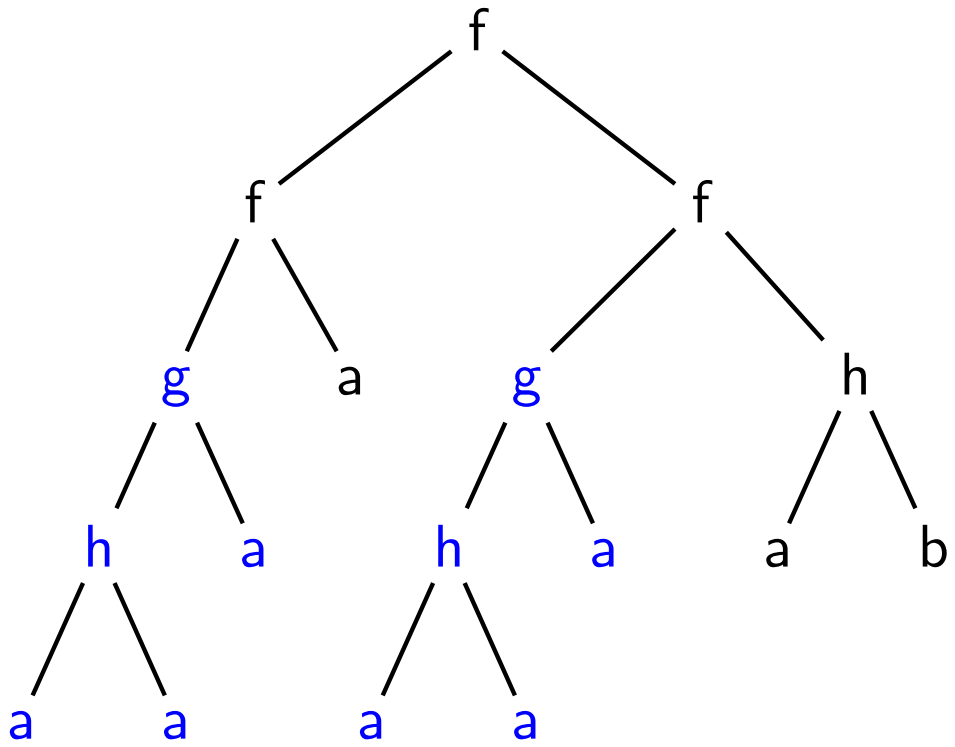
- the context: query languages for XML
- more precisely: TQL spatial logic (Cardelli, Ghelli, ICALP'02) (Filiot, Talbot,T, CSL'07)
- express tree patterns with tree (dis)equality tests
- equivalent automaton model?
- equality tests are **not** local

Examples of TQL-definable languages



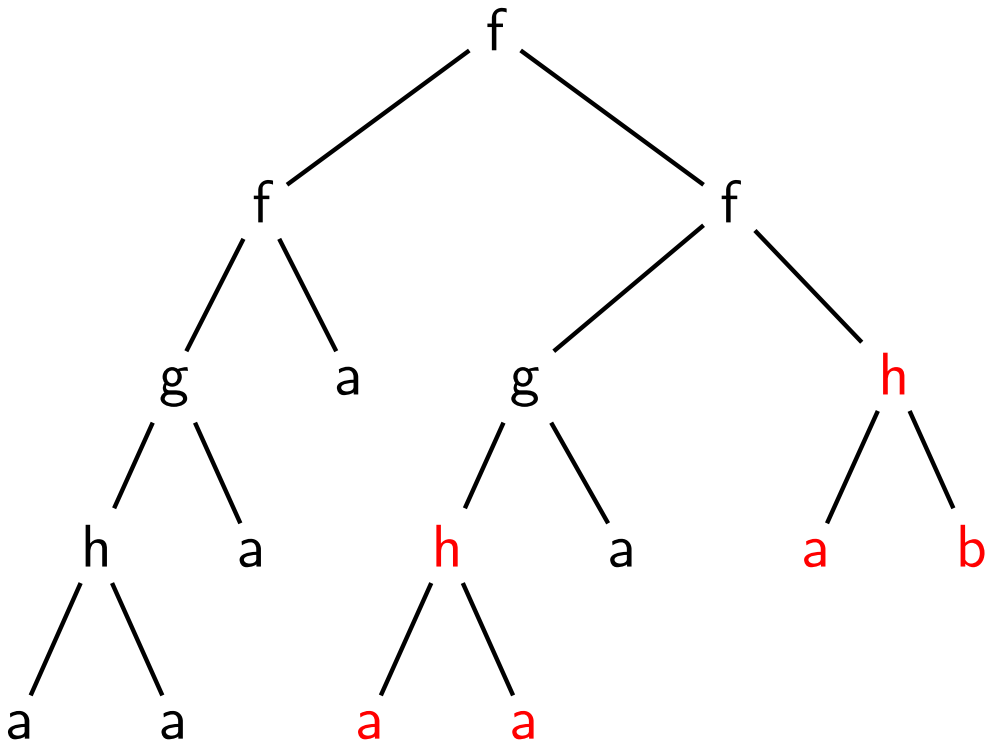
1. all regular languages

Examples of TQL-definable languages



1. all regular languages
2. all subtrees rooted at g are equal

Examples of TQL-definable languages



1. all regular languages
2. all subtrees rooted at g are equal
3. there are at least two different subtrees rooted at h

Tree Automata with Global Equalities and Disequalities

A **tree automata A with global equalities and disequalities** (TAGED) is given by:

Σ alphabet
Q set of states
F set of final states
Δ set of rules

} tree automaton

Tree Automata with Global Equalities and Disequalities

A **tree automata A with global equalities and disequalities** (TAGED) is given by:

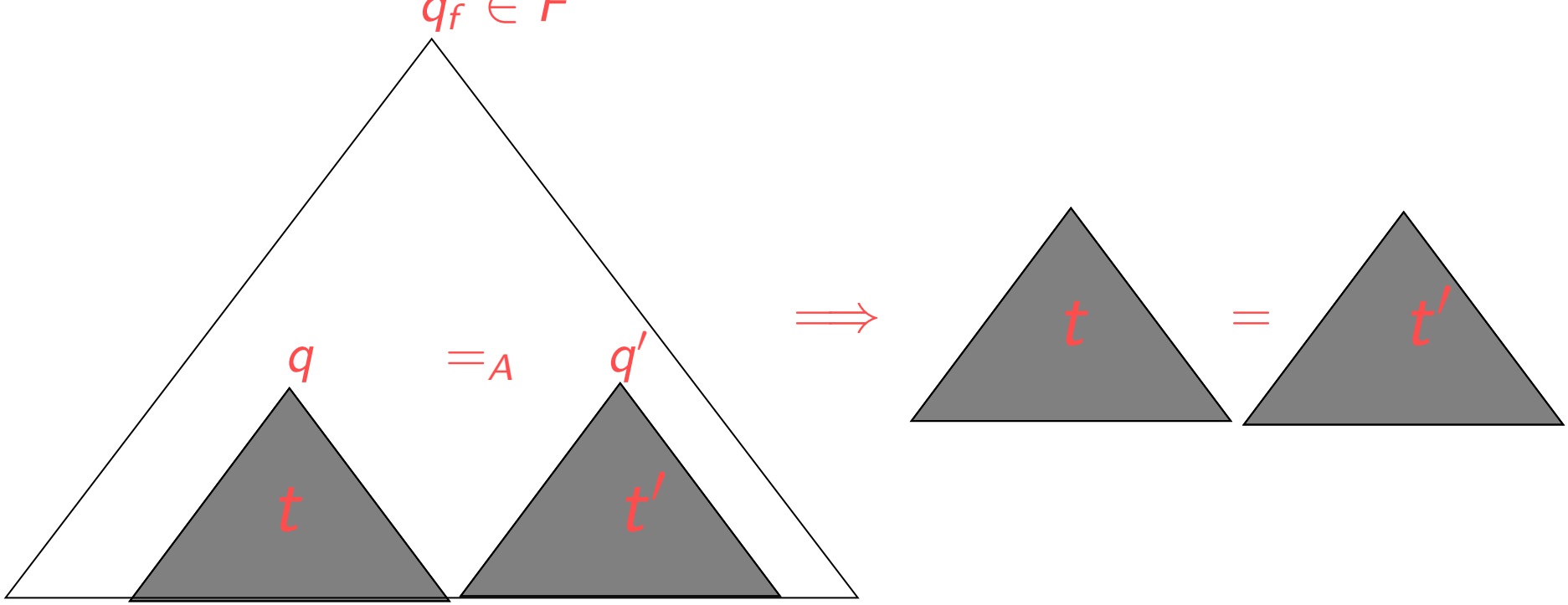
Σ alphabet
Q set of states
F set of final states
Δ set of rules

} tree automaton

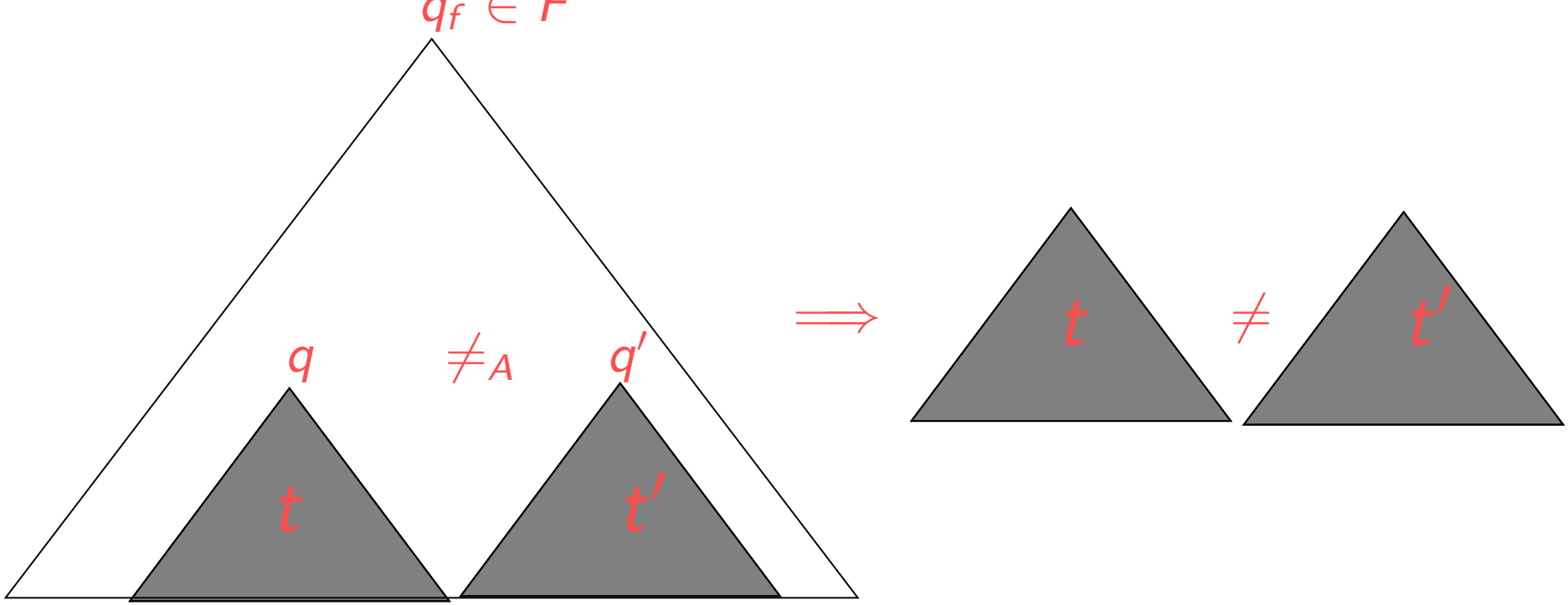
$=_A \subseteq Q^2$ reflexive and symmetric relation on a **subset** of Q

$\neq_A \subseteq Q^2$ irreflexive and symmetric relation

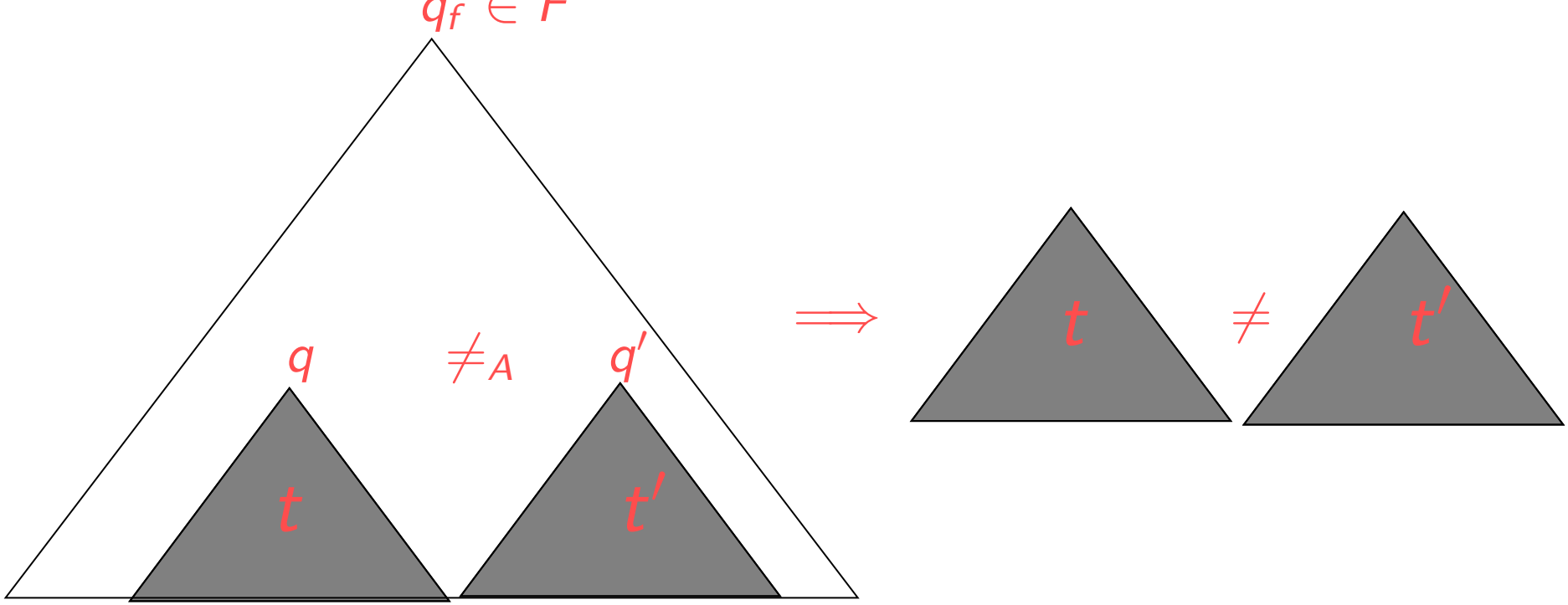
Successful Runs



Successful Runs



Successful Runs

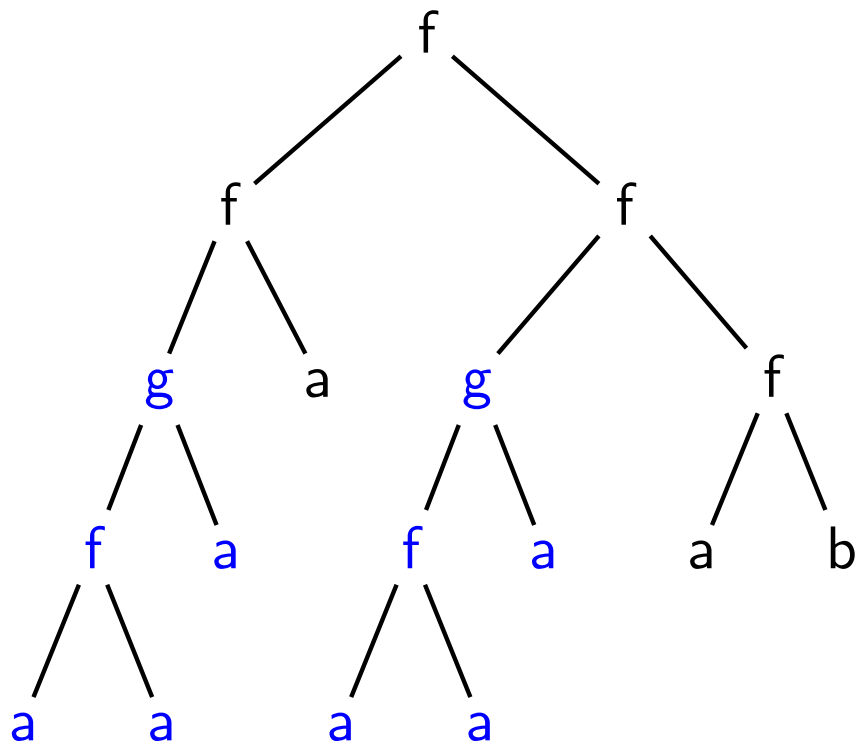


TAGEDs and automata with local constraints are orthogonal

Given a successful run, two nodes are equivalent if they have been "successfully tested" to be equal in the run

	number of eq. classes	cardinality of eq. classes
local constraints	unbounded	bounded
TAGEDs	bounded	unbounded

Example: all subtrees rooted at g are equal

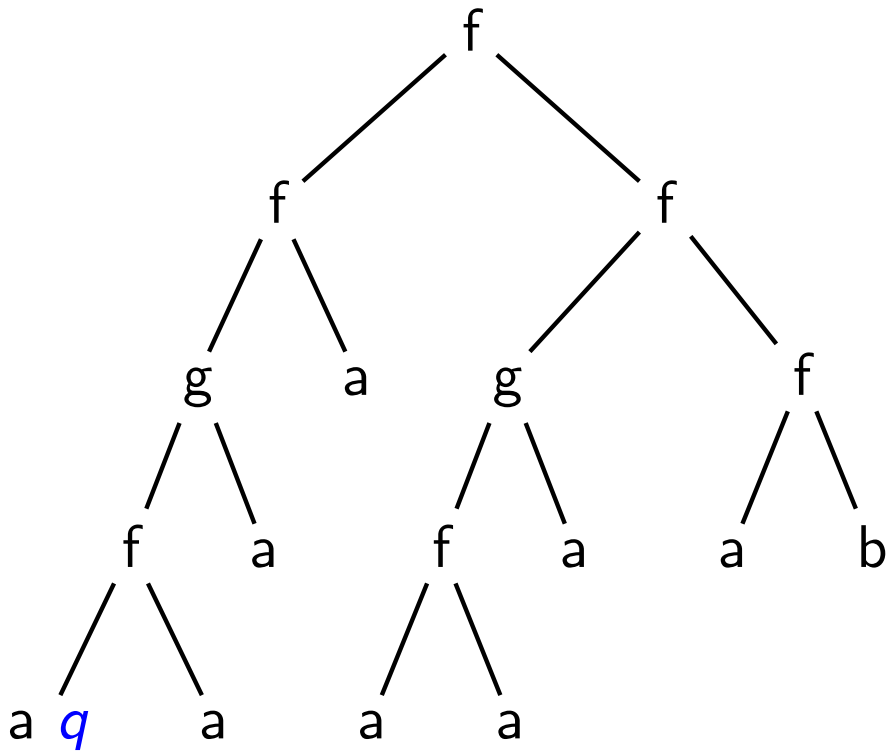


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
a &\to q \\
g(q,q) &\to q_g \\
f(q,q) &\to q \\
f(q_g,q) &\to q_g^{\downarrow} \\
f(q,q_g) &\to q_g^{\downarrow} \\
f(q_g,q_g) &\to q_g^{\downarrow} \\
f(q_g^{\downarrow},q_g) &\to q_g^{\downarrow} \\
&\dots
\end{aligned}
$$

- $q_g =_A q_g$

Example: all subtrees rooted at g are equal

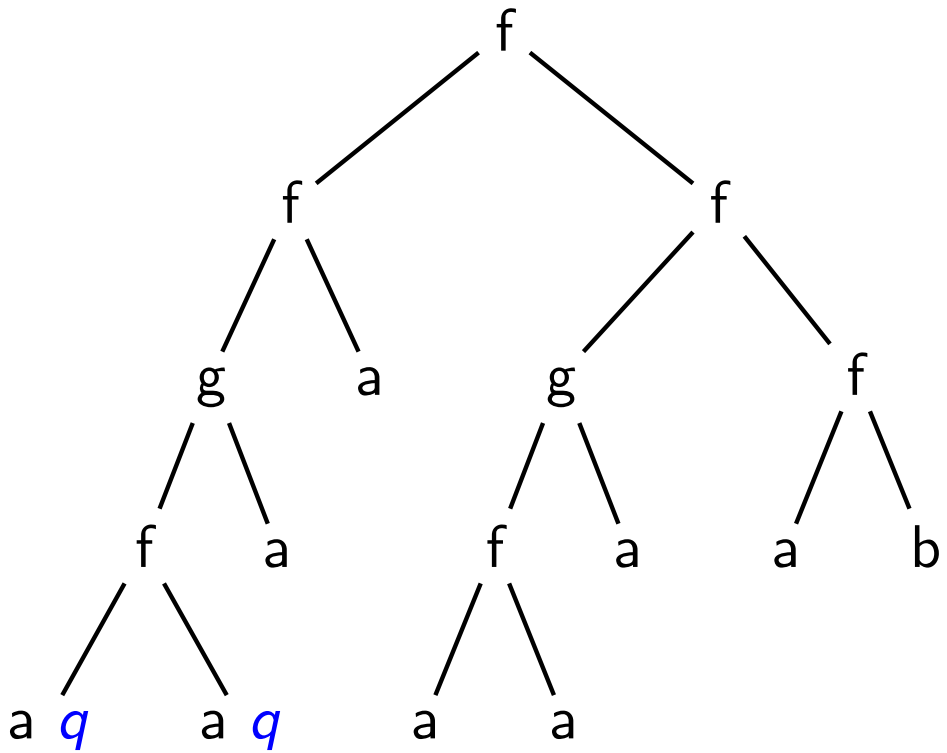


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
a &\to q \\
g(q,q) &\to q_g \\
f(q,q) &\to q \\
f(q_g,q) &\to q_g^{\downarrow} \\
f(q,q_g) &\to q_g^{\downarrow} \\
f(q_g,q_g) &\to q_g^{\downarrow} \\
f(q_g^{\downarrow},q_g) &\to q_g^{\downarrow} \\
&\dots
\end{aligned}
$$

- $q_g =_A q_g$

Example: all subtrees rooted at g are equal

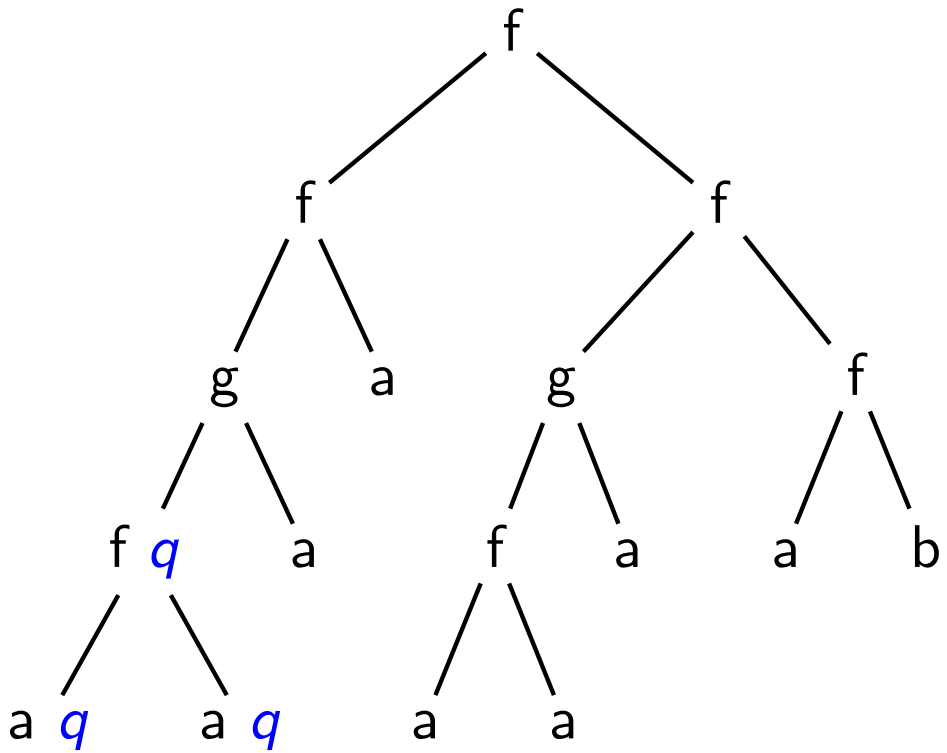


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
a &\to q \\
g(q,q) &\to q_g \\
f(q,q) &\to q \\
f(q_g,q) &\to q_g^{\downarrow} \\
f(q,q_g) &\to q_g^{\downarrow} \\
f(q_g,q_g) &\to q_g^{\downarrow} \\
f(q_g^{\downarrow},q_g) &\to q_g^{\downarrow} \\
&\dots
\end{aligned}
$$

- $q_g =_A q_g$

Example: all subtrees rooted at g are equal

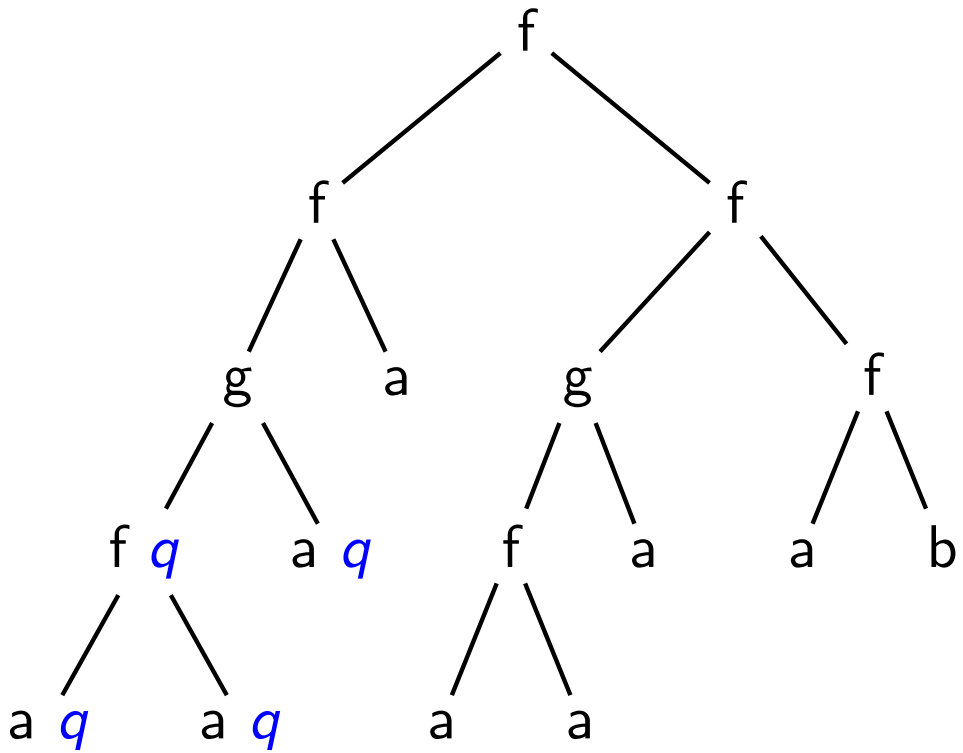


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
a &\to q \\
g(q,q) &\to q_g \\
f(q,q) &\to q \\
f(q_g,q) &\to q_g^{\downarrow} \\
f(q,q_g) &\to q_g^{\downarrow} \\
f(q_g,q_g) &\to q_g^{\downarrow} \\
f(q_g^{\downarrow},q_g) &\to q_g^{\downarrow} \\
&\dots
\end{aligned}
$$

- $q_g \ =_A \ q_g$

Example: all subtrees rooted at g are equal

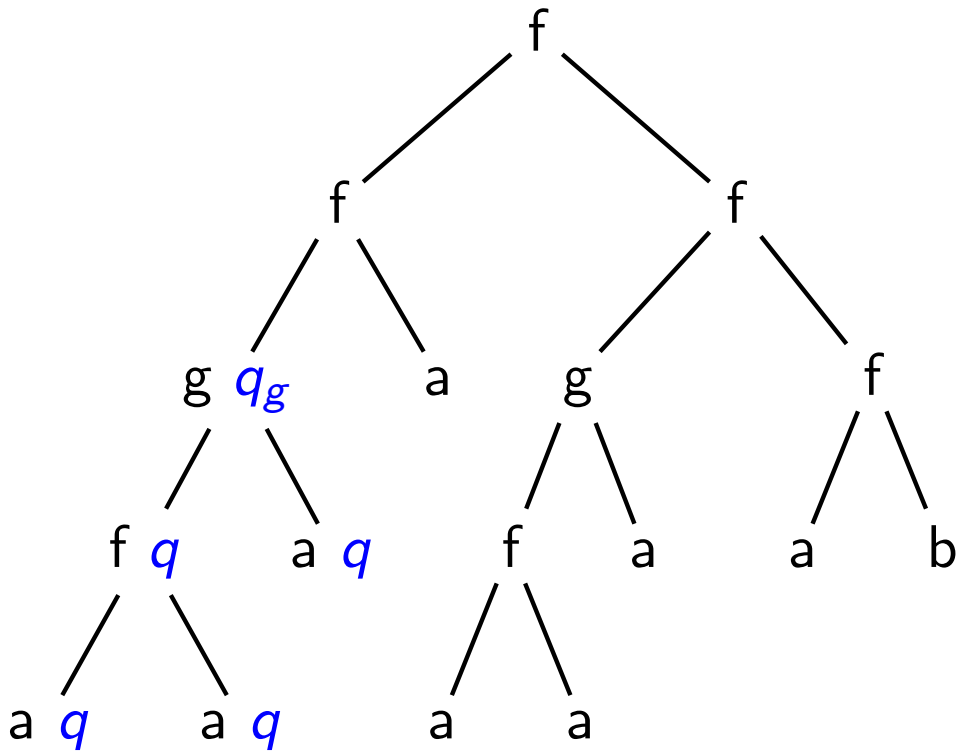


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
a &\to q \\
g(q,q) &\to q_g \\
f(q,q) &\to q \\
f(q_g,q) &\to q_g^{\downarrow} \\
f(q,q_g) &\to q_g^{\downarrow} \\
f(q_g,q_g) &\to q_g^{\downarrow} \\
f(q_g^{\downarrow},q_g) &\to q_g^{\downarrow} \\
&\dots
\end{aligned}
$$

- $q_g =_A q_g$

Example: all subtrees rooted at g are equal

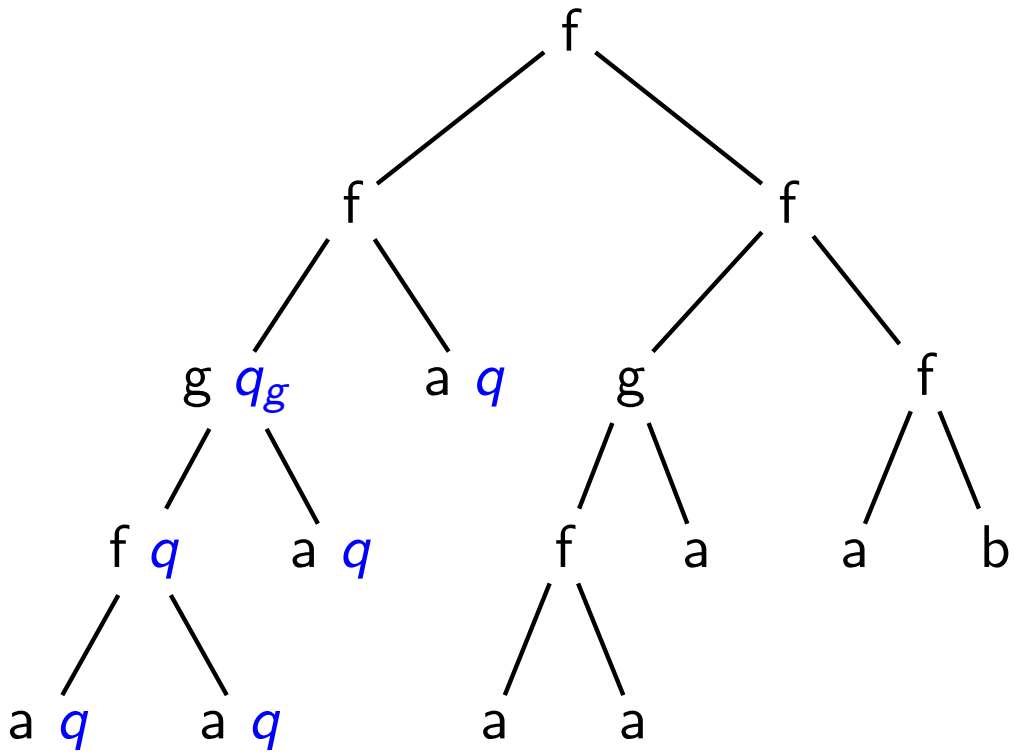


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
& a \longrightarrow q \\
& g(q,q) \longrightarrow q_g \\
& f(q,q) \longrightarrow q \\
& f(q_g,q) \longrightarrow q_g^{\downarrow} \\
& f(q,q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g,q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g^{\downarrow},q_g) \longrightarrow q_g^{\downarrow} \\
& \ldots
\end{aligned}
$$

- $q_g \ =_A \ q_g$

Example: all subtrees rooted at g are equal

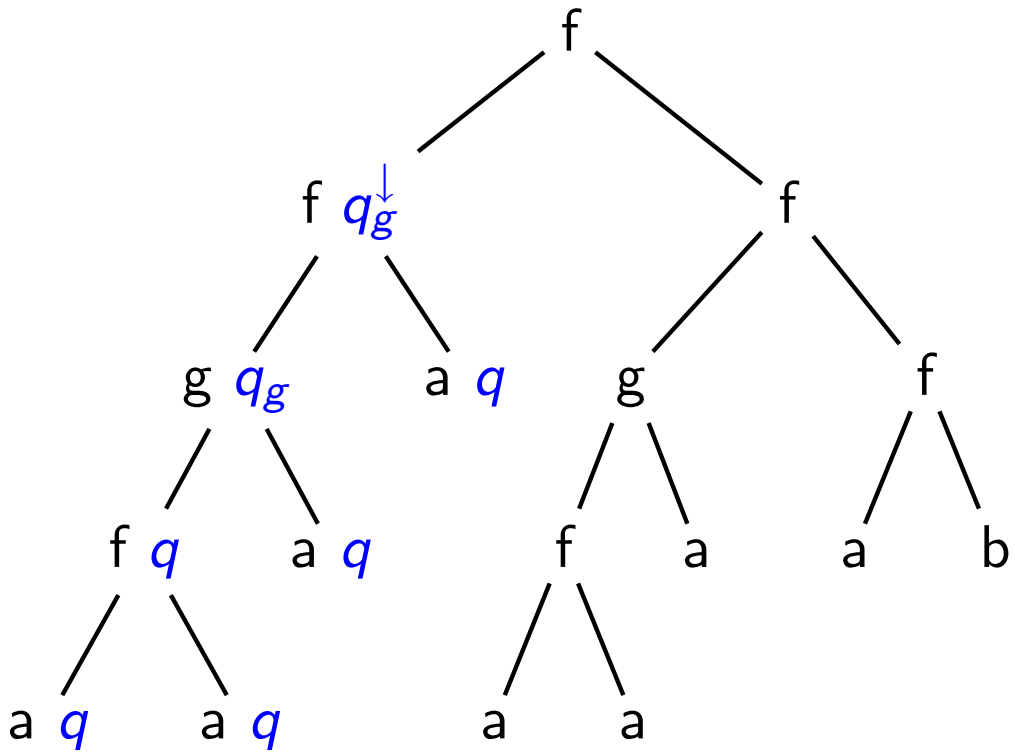


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
& a \longrightarrow q \\
& g(q, q) \longrightarrow q_g \\
& f(q, q) \longrightarrow q \\
& f(q_g, q) \longrightarrow q_g^{\downarrow} \\
& f(q, q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g, q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g^{\downarrow}, q_g) \longrightarrow q_g^{\downarrow} \\
& \ldots
\end{aligned}
$$

- $q_g \ =_A \ q_g$

Example: all subtrees rooted at g are equal

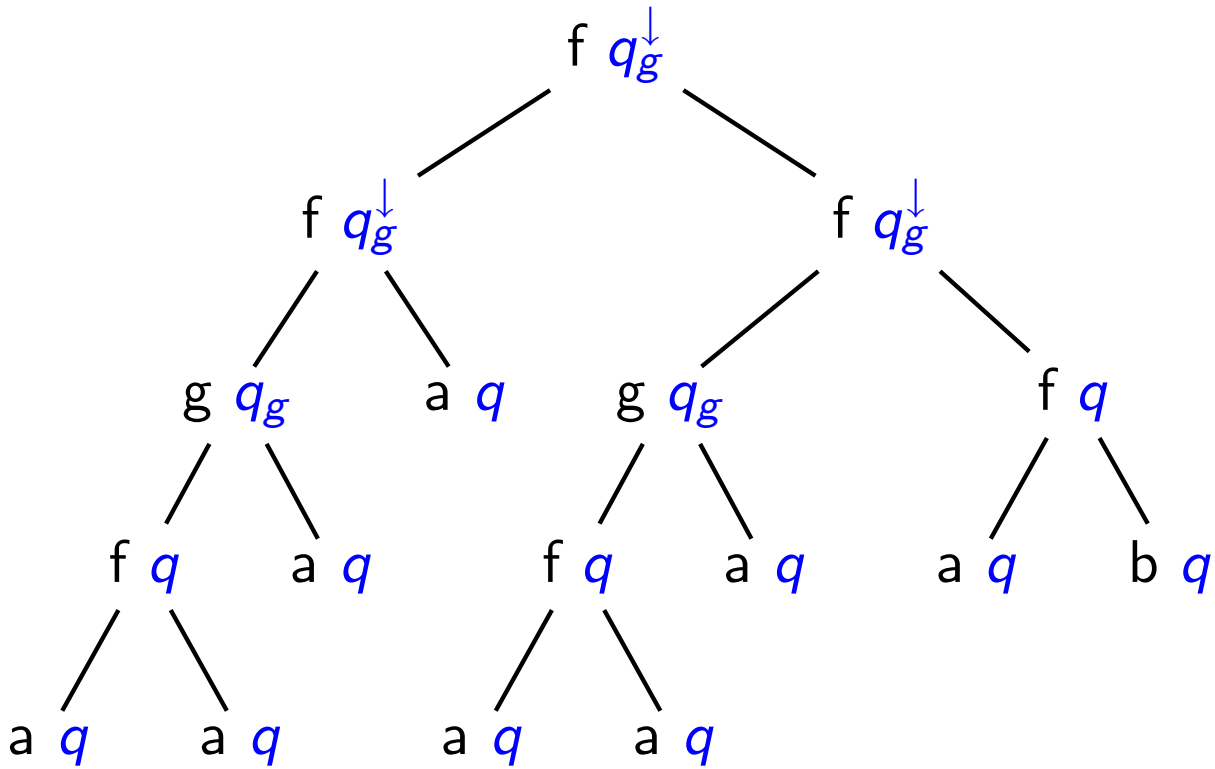


- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
& a \longrightarrow q \\
& g(q,q) \longrightarrow q_g \\
& f(q,q) \longrightarrow q \\
& f(q_g,q) \longrightarrow q_g^{\downarrow} \\
& f(q,q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g,q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g^{\downarrow},q_g) \longrightarrow q_g^{\downarrow} \\
& \ldots
\end{aligned}
$$

- $q_g \ =_A \ q_g$

Example: all subtrees rooted at g are equal



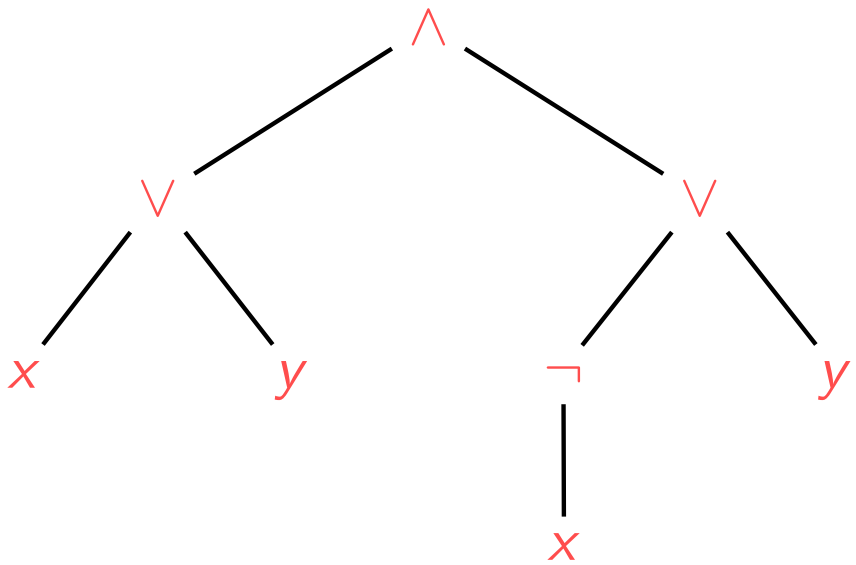
- $\Sigma = \{f, a, g\}$
- $Q = F = \{q, q_g, q_g^{\downarrow}\}$
- $\Delta =$

$$
\begin{aligned}
& a \longrightarrow q \\
& g(q, q) \longrightarrow q_g \\
& f(q, q) \longrightarrow q \\
& f(q_g, q) \longrightarrow q_g^{\downarrow} \\
& f(q, q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g, q_g) \longrightarrow q_g^{\downarrow} \\
& f(q_g^{\downarrow}, q_g) \longrightarrow q_g^{\downarrow} \\
& \ldots
\end{aligned}
$$

- $q_g \ =_A \ q_g$

Example: satisfiable CNF formulas

$\Phi = (x \vee y) \wedge (\neg x \vee y)$



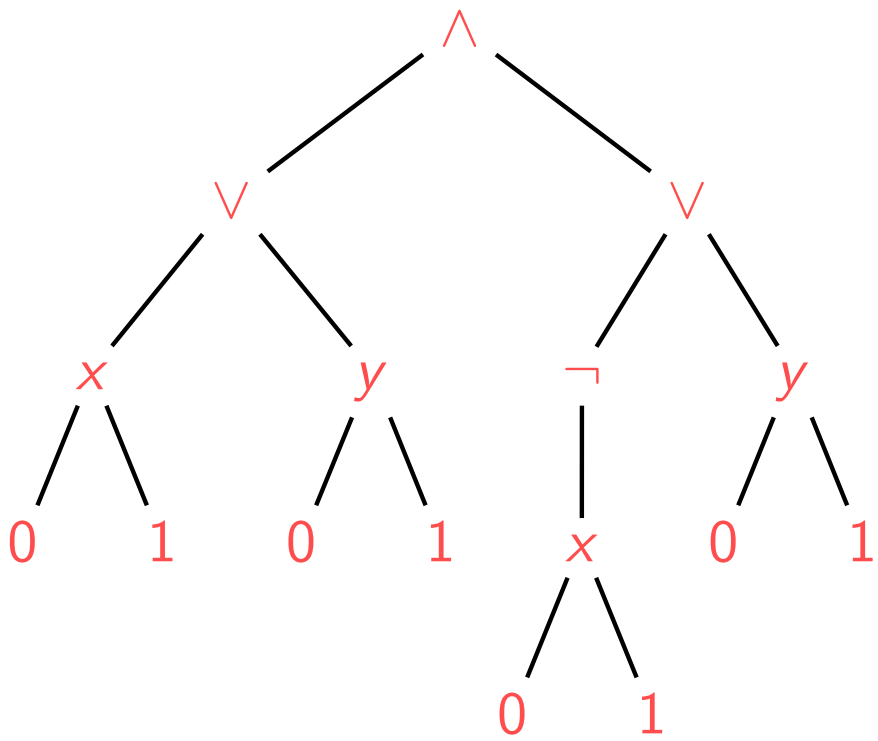
- $\Sigma = \{\vee, \wedge, x, y\}$
- $Q = \{q_0, q_1\}$
  $F = \{q_1\}$
- $\Delta =$

  $x \longrightarrow q_0$ $x \longrightarrow q_1$

  ...

Example: satisfiable CNF formulas

$\Phi = (x \vee y) \wedge (\neg x \vee y)$



- $\Sigma = \{\vee, \wedge, x, y\}$
- $Q = \{q_0, q_1, q, q_x, q_y\}$
  $F = \{q_1\}$
- $\Delta =$

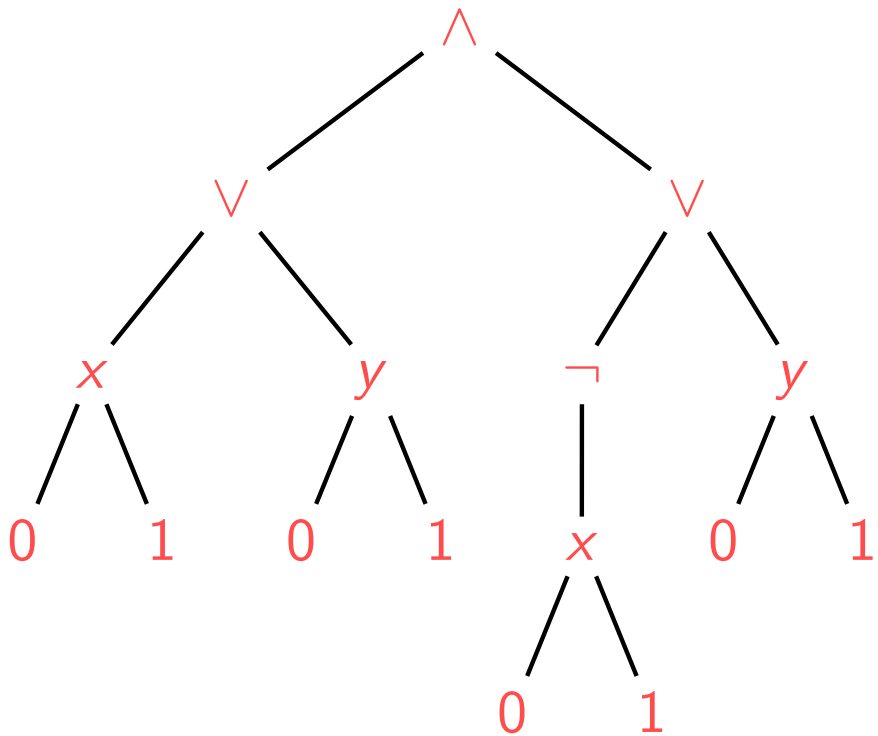
  $0 \to q_x \quad 0 \to q$
  $1 \to q_x \quad 1 \to q$
  $x(q, q_x) \to q_1$
  $x(q_x, q) \to q_0$

  $0 \to q_y \quad 0 \to q$
  ...

- $q_x =_A q_x \; q_y =_A q_y$

Example: satisfiable CNF formulas

$\Phi = (x \vee y) \wedge (\neg x \vee y)$



Prop. The uniform membership problem, given A and t, decide whether $t \in L(A)$? is NP-complete.

- $\Sigma = \{\vee, \wedge, x, y\}$
- $Q = \{q_0, q_1, q, q_x, q_y\}$
  $F = \{q_1\}$
- $\Delta =$

  $0 \to q_x \quad 0 \to q$
  $1 \to q_x \quad 1 \to q$
  $x(q, q_x) \to q_1$
  $x(q_x, q) \to q_0$

  $0 \to q_y \quad 0 \to q$
  ...

- $q_x =_A q_x \; q_y =_A q_y$

Determinization and Complement

Proposition

TAGEDs are not determinizable.

Proposition

TAGED-definable languages are not closed by complement.

Idea. The set of trees such that there is a subtree of the form $g(t, t')$ with $t \neq t'$ is TAGED-definable, but not its complement.

Closure by Union and Intersection

Proposition

TAGED-languages are closed by union and intersection.

Closure by Union and Intersection

Proposition

TAGED-languages are closed by union and intersection.

Sketch.

- union: take the disjoint union of the two TAGEDs

Closure by Union and Intersection

Proposition

TAGED-languages are closed by union and intersection.

Sketch.

- union: take the disjoint union of the two TAGEDs
- intersection: product automaton $A_1 \times A_2$ with

$$(q_1, q_2) =_{A_1 \times A_2} (p_1, p_2) \quad \text{iff} \quad q_1 =_{A_1} p_1 \text{ or } q_2 =_{A_1} p_2$$

$$(q_1, q_2) \neq_{A_1 \times A_2} (p_1, p_2) \quad \text{iff} \quad q_1 \neq_{A_1} p_1 \text{ or } q_2 \neq_{A_1} p_2$$

Emptiness Problem

- **Input**: a TAGED A
- **Output**: is there a tree accepted by A?

Theorem

Emptiness is decidable for positive, negative and bounded TAGEDs.

Simplification: Testing equalities can be done with the same state

Lemma

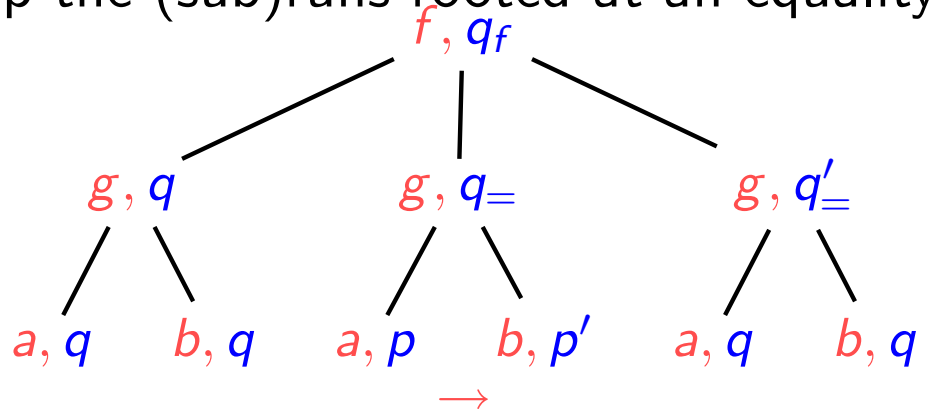
In a successful run, the same state can be used to test equalities: every TAGED A is equivalent to $A' = (Q', F', \Delta', =_{A'}, \neq_{A'})$ such that $=_{A'} \subseteq id_{Q'}$

Simplification: Testing equalities can be done with the same state

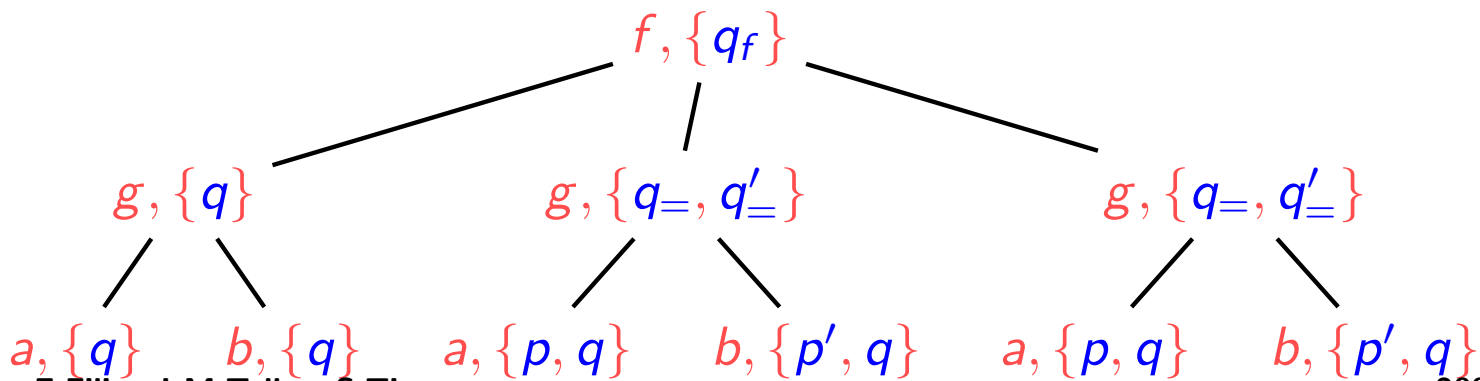
Lemma

In a successful run, the same state can be used to test equalities: every TAGED A is equivalent to $A' = (Q', F', \Delta', =_{A'}, \neq_{A'})$ such that $=_{A'} \subseteq id_{Q'}$

Idea: overlap the (sub)runs rooted at an equality state



$\longrightarrow$



Simplification: Testing equalities can be done with the same state

Lemma

In a successful run, the same state can be used to test equalities: every TAGED A *is equivalent to* $A' = (Q', F', \Delta', =_{A'}, \neq_{A'})$ *such that* $=_{A'} \subseteq id_{Q'}$

Idea: overlap the (sub)runs rooted at an equality state

Simplification: Testing equalities can be done with the same state

Lemma

In a successful run, the same state can be used to test equalities: every TAGED A *is equivalent to* $A' = (Q', F', \Delta', =_{A'}, \neq_{A'})$ *such that* $=_{A'} \subseteq id_{Q'}$

Idea: overlap the (sub)runs rooted at an equality state

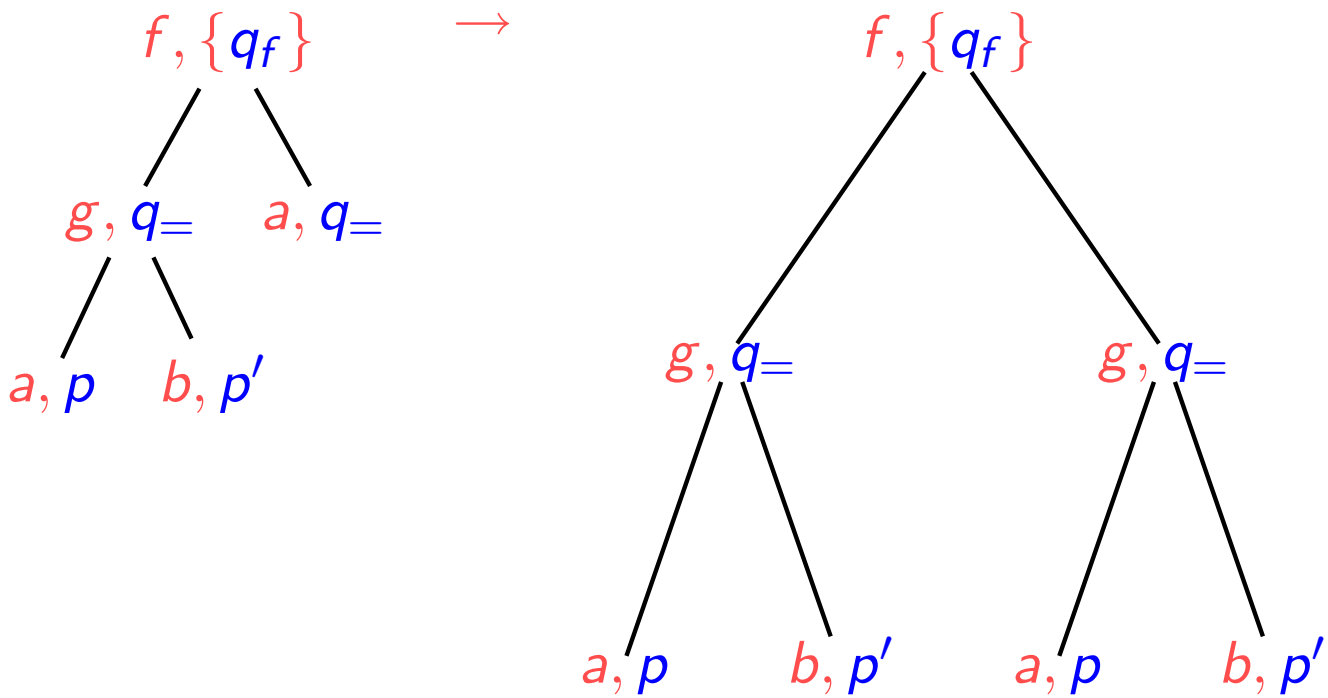
1. guess which states are used to test equalities
2. subset construction
3. $\rightarrow |A'| = 2^{O(|A|)}$

Emptiness of Positive TAGED (Upper Bound)

Theorem

Emptiness of positive TAGED is decidable in EXPTIME, and in linear time if $=_A \subseteq id_Q$.

1. transform A into an equivalent TAGED A' such that $=_{A'} \subseteq id_{Q'}$;
2. test emptiness of A' as for classical tree automata:



Emptiness of Positive TAGEDs (Lower Bound)

Theorem

Emptiness of positive TAGEDs is EXPTIME-hard.

Sketch. Reduction from the emptiness test of the intersection of n tree automata $A_1, \ldots, A_n$.

1. let A define the set of trees $\#(t_1, \ldots, t_n)$ such that $\forall i, t_i \in L(A_i)$
2. let $A_=$ define the set of trees $\#(t, \ldots, t)$
3. let $A_\cap$ define the set of trees $L(A) \cap L(A_=)$
4. test emptiness of $A_\cap$

Emptiness of Negative TAGEDs

Theorem

Emptiness of negative TAGEDs is decidable in NEXPTIME.

- Encoding into a system of set constraints

$$\begin{cases} X_q \subseteq \bigcup_{f(q_1,q_2)\to q\in\Delta} f(X_{q_1}, X_{q_2}) \cup \bigcup_{a\to q\in\Delta} a \\ \bigcup_{q\in F} X_q \not\subseteq \varnothing \\ X_q \cap X_p = \varnothing \text{ for all } q, p \in Q \text{ such that } q \neq_A p \end{cases}$$

- can be solved in NEXPTIME (Aiken,Kozen,Wimmers,95) (Charatonik,Pacholski,94)

Main Result: Bounded TAGEDs

Definition

A bounded TAGED is a pair (A, k) where A is a TAGED and $k \in \mathbb{N}$ is a natural number.

Definition (Successful Runs)

Additional condition: every state in the domain of $\neq_A$ must occur at most k times along any root-to-leaves path.

Main Result: Bounded TAGEDs

Definition

A bounded TAGED is a pair (A, k) where A is a TAGED and $k \in \mathbb{N}$ is a natural number.

Definition (Successful Runs)

Additional condition: every state in the domain of $\neq_A$ must occur at most k times along any root-to-leaves path.

Theorem

Emptiness of bounded TAGEDs is decidable in 2NEXPTIME.

- assume that $=_A \subseteq id_Q$
- put the same run below equality states
- pump in parallel below equality states
- repair the unsatisfied disequality constraints

Back to the first motivation: Querying XML

- TAGED need to be lifted to the unranked case.
- We can define hedge automata with equality and disequality tests.
- The "vertically bounded" restriction will be the same: a bounded number of difference are tested along a path.
- By an encoding of unranked trees to ranked ones, we can use TAGED for binary trees.
- We get decidabilty for a fragment of TQL.

MSO with Tree Isomorphism Tests: MSO($\sim$)

- first-order variables x, y denote **nodes**
- second-order variables X, Y denote **set of nodes**
- a new predicate $x \sim y$ to test tree isomorphisms between subtrees rooted at x and y respectively

Theorem

Satisfiability of MSO($\sim$) is undecidable.

Existential Fragment: $MSO^{\exists}_{=}$

- remove $\sim$ and add new predicates:

$$\begin{array}{lcl} eq(X) & = & \forall x, y \in X,\ x \sim y \\ diff_k(X, Y) & = & \forall x \in X \forall y \in Y, x \not\sim y \wedge \\ & & DescChain(X) \leq k \wedge DescChain(Y) \leq k \end{array}$$

- formulas of the form:

$$\exists X_1 \ldots \exists X_n \psi(X_1, \ldots, X_n)$$

- tests $eq(X_i)$ or $diff_k(X_i, X_j)$ allowed **only** on $X_1, \ldots, X_n$ in ψ

Theorem

- **expressivity**: *$MSO^{\exists}_{=}$ sentences and bounded TAGEDs effectively define the same tree languages;*
- **satisfiability**: *decidable for $MSO^{\exists}_{=}$.*

Unification with membership constraints

- atoms of the form $s = s'$ or $x \in L$ (true if $\exists \sigma, \sigma(s) = \sigma(s')$)
- s, s' are terms with variables
- FO over these atoms is decidable (Comon, Delor, ICALP'90)

Unification with membership constraints

- atoms of the form $s = s'$ or $x \in L$ (true if $\exists \sigma, \sigma(s) = \sigma(s')$)
- s, s' are terms with variables
- FO over these atoms is decidable (Comon, Delor, ICALP'90)

- add context variables C and atoms $C \in L$
- **restriction:** cannot use the same context variable in two different terms
- full FO is undecidable (even with the restriction)
- decidable for the existential fragment (by using bounded TAGEDs)

Conclusion for TAGED

Summary

- automata with global constraints orthogonal to classical automata with local constraints
- emptiness for three subclasses
- correspondence with a super MSO fragment
- useful for decidability of a fragment of TQL

To do?

- emptiness of full TAGEDs?
- regularity: given a TAGED, does it define a regular language?
- deterministic class

Thanks to Hubert for what he has given to Tree Automata Techniques:

Thanks to Hubert for what he has given to Tree Automata Techniques:

- his own contributions

Thanks to Hubert for what he has given to Tree Automata Techniques:

- his own contributions
- his major role in TATA book

Thanks to Hubert for what he has given to Tree Automata Techniques:

- his own contributions
- his major role in TATA book
- his major role in encouraging the "French Tree Automata group"!