Tree Automata and Constraints: a brief survey

Emmanuel Filiot       Florent Jacquemard       Sophie Tison

1 University of Paris 12, LACL
2 INRIA Paris-Rocquencourt, Ircam
3 University of Lille 1, LIFL

Dagstuhl Seminar 13192
Tree Transducers and Formal Methods
Non-linearity phenomena are non captured by tree automata, e.g.:

- The set of ground instances of a non-linear term is not recognizable.
- The class of recognizable tree languages is not closed under non-linear homomorphisms.

How to enrich transitions by equality and disequality constraints?
Two main ideas

- Add **local** tests, e.g. to capture the set of instances of $f(g(x), x)$
  Why: pattern matching, rewriting, representing finitely image by non-linear transformations of recognizable tree languages (type checking?), …
  When: from 80’s to now

- Add **global** tests, e.g. to capture "all the subterms rooted by $f$ are different".
  Why: integrity constraints (XML), non linear query languages, …
  When: more recently

*Leitmotiv: keeping as far as possible good closure and decision properties.*
Tree Automata with Local Constraints
An equality constraint (resp. a disequality constraint) is a predicate on $T(\Sigma)$ written $\pi = \pi'$ (resp. $\pi \neq \pi'$) where $\pi, \pi' \in \{1, \ldots, k\}^*$. 

Such a predicate is satisfied on a tree $t$, which we write $t \models \pi = \pi'$, if $\pi, \pi' \in Post$ and $\models t|_\pi = t|_{\pi'}$ (resp. $\pi \neq \pi'$ is satisfied on $t$ if $\pi = \pi'$ is not satisfied on $t$).
Equality and Disequality Constraints

An equality constraint (resp. a disequality constraint) is a predicate on $T(\Sigma)$ written $\pi = \pi'$ (resp. $\pi \neq \pi'$) where $\pi, \pi' \in \{1, \ldots, k\}^*$. Such a predicate is satisfied on a tree $t$, which we write $t \models \pi = \pi'$, if $\pi, \pi' \in \mathcal{Post}$ and $\models t|_\pi = t|_{\pi'}$ (resp. $\pi \neq \pi'$ is satisfied on $t$ if $\pi = \pi'$ is not satisfied on $t$).

These constraints are indeed unlabeled path constraints. More general disequality constraints have been defined by H. Seidl and A. Reuß (LPAR 2010, FOSSACS 2012).
Automata with Equality and Disequality Constraints

The general class

An automaton with equality and disequality constraints is a tuple \((Q, \Sigma, Q_f, \Delta)\) where

- \(\Sigma\) is a finite ranked alphabet
- \(Q\) is a finite set of states
- \(Q_f\) is a subset of \(Q\) of final states
- \(\Delta\) is a set of transition rules of the form:

\[
f(q_1, \ldots, q_n) \xrightarrow{c} q
\]

where \(f \in \Sigma, q_1, \ldots, q_n, q \in Q\), and \(c\) is a Boolean combination of equality and disequality constraints.
Automata with Equality and Disequality Constraints

The general class

- The move relation $\rightarrow_\mathcal{A}$ is defined by:
  
  $t \rightarrow_\mathcal{A} t'$ if and only if
  
  $t = C[f(q_1(u_1), \ldots, q_n(u_n))],
  
  t' = C[q(f(u_1, \ldots, u_n))]
  
  $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta,$
  
  $f(u_1, \ldots, u_n) \models c.$

- $\rightarrow^*_\mathcal{A}$ is the reflexive and transitive closure of $\rightarrow_\mathcal{A}$.

- $\mathcal{A}$ accepts a tree $t \in T(\Sigma)$ if $t \rightarrow^*_\mathcal{A} q$ for some final state $q$.

- The language accepted, or recognized, is the set $L(\mathcal{A})$ of trees $t \in T(\Sigma)$ accepted by $\mathcal{A}$. 
Automata with Equality and Disequality Constraints

The general class

- The move relation $\rightarrow_{A}$ is defined by:
  $t \rightarrow_{A} t'$ if and only if
  \[
  t = C[f(q_1(u_1), \ldots, q_n(u_n))],
  t' = C[q(f(u_1, \ldots, u_n))],
  f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta,
  f(u_1, \ldots, u_n) \models c.
  \]

- $\rightarrow^*_{A}$ is the reflexive and transitive closure of $\rightarrow_{A}$.

- $A$ accepts a tree $t \in T(\Sigma)$ if $t \rightarrow^*_{A} q$ for some final state $q$.

- The language accepted, or recognized, is the set $L(A)$ of trees $t \in T(\Sigma)$ accepted by $A$.

- Class with only equalities defined in 1981 (Dauchet & Mongy)
Automata with Equality and Disequality Constraints

Example 1

- $Q = \{q\}$,
- $\Sigma = \{f, a\}$
- final states: $\{q\}$
- $\Delta$ consists of the following rules:

$$r_1 : a \rightarrow q$$

$$r_2 : f(q, q) \xrightarrow{1=2} q$$
Example 1

- $Q = \{q\}$,
- $\Sigma = \{f, a\}$
- final states: $\{q\}$
- $\Delta$ consists of the following rules:

$$r_1 : \quad a \rightarrow q$$
$$r_2 : \quad f(q, q) \xrightarrow{1=2} q$$

The automaton recognizes the set of balanced trees.
Automata with Equality and Disequality Constraints

Example 2

- \( Q = \{q, q_g, q_{fin}\} \),
- \( \Sigma = \{f, g, a\} \)
- final states: \( \{q_{fin}\} \)
- \( \Delta \) consists of the following rules:

\[
\begin{align*}
  r_1 & : \quad a \rightarrow q \\
  r_2 & : \quad f(q, q) \rightarrow q \\
  r_3 & : \quad f(q_g, q) \xrightarrow{11=2} q_{fin} \\
  r_4 & : \quad g(q) \rightarrow q_g \\
  r_5 & : \quad g(q) \rightarrow q
\end{align*}
\]
Automata with Equality and Disequality Constraints

Example 2

- $Q = \{q, q_g, q_{fin}\}$,
- $\Sigma = \{f, g, a\}$
- final states: $\{q_{fin}\}$
- $\Delta$ consists of the following rules:

  $r_1:\quad a \rightarrow q$
  $r_2:\quad f(q, q) \rightarrow q$
  $r_3:\quad f(q_g, q) \xrightarrow{11=2} q_{fin}$
  $r_4:\quad g(q) \rightarrow q_g$
  $r_5:\quad g(q) \rightarrow q$

The automaton recognizes the ground instances of $f(g(x), x)$. 
Automata with Equality and Disequality Constraints

Example 3

- $Q = \{q\}$,
- $\Sigma = \{f, a\}$
- Final states: $\{q\}$
- $\Delta$ consists of the following rules:

  - $r_1 : a \rightarrow qa$
  - $r_2 : f(q_a, q_a) \rightarrow q$
  - $r_3 : f(q, q) \xrightarrow{12=21} q$
Automata with Equality and Disequality Constraints

Example 3

- $Q = \{q\}$,
- $\Sigma = \{f, a\}$
- Final states: $\{q\}$
- $\Delta$ consists of the following rules:

  - $r_1 : a \rightarrow qa$
  - $r_2 : f(qa, qa) \rightarrow q$
  - $r_3 : f(q, q) \xrightarrow{12=21} q$

The automaton recognizes "grids".
Automata with Equality and Disequality Constraints

Properties

- Can be determinized (and completed) (exponential blow up)
Automata with Equality and Disequality Constraints

Properties

- Can be determinized (and completed) (exponential blow up)

- Closure under boolean operations
Automata with Equality and Disequality Constraints

Properties

- Can be determinized (and completed) (exponential blow up)

- Closure under boolean operations

- Emptiness is undecidable for this class - e.g. use Post problem or previous encoding of grids -: undecidable even with only equalities between cousins.
Automata with Equality and Disequality Constraints

Properties

- Can be determinized (and completed) (exponential blow up)
- Closure under boolean operations
- Emptiness is undecidable for this class - e.g. use Post problem or previous encoding of grids -: undecidable even with only equalities between cousins.
- Capture homomorphic images of recognizable languages (only equality constraints are needed) and images by bottom-up transducers.

Undecidability comes in some sense from superposition of (equality) tests. To get a decidable class, we can try to limit this superposition...
Automata with Constraints

Restrictions

- restrict the form of the tests
- limit the number of equality tests
- limit the superposition of equality tests
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are $i = j$ or $i \neq j$.

$f(q_1, q_2, q_3) \rightarrow_{1=2, \ 1\neq3}$ allowed, $f(q_1, q_2) \rightarrow_{1=21}$ forbidden.
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are $i = j$ or $i \neq j$.

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

▷ can be determinized (exponential blow up)

▷ stable under boolean operations.

▷ Emptiness $\text{EXPTIME}$—complete but polynomial in the deterministic case.

Proof for ”normal” TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

▷ $f(q_1, q_2) \rightarrow q, 1 = 2$: 
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are \( i = j \) or \( i \neq j \).

\[
f(q_1, q_2, q_3) \rightarrow_{1=2, 1\neq3} \text{allowed, } f(q_1, q_2) \rightarrow_{1=2} \text{forbidden.}
\]

- can be determinized (exponential blow up)
- stable under boolean operations.
- Emptiness \( EXPTIME \)–complete but polynomial in the deterministic case.

Proof for "normal" TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

- \( f(q_1, q_2) \rightarrow q, 1 = 2 \): if the automaton is not deterministic, we have to check whether there is some tree leading both to \( q_1 \) and \( q_2 \): we can determinize
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are $i = j$ or $i \neq j$.

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

- can be determinized (exponential blow up)
- stable under boolean operations.
- Emptiness $EXPTIME$—complete but polynomial in the deterministic case.

Proof for "normal" TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

- $f(q_1, q_2) \rightarrow q, 1 = 2$: if the automaton is not deterministic, we have to check whether there is some tree leading both to $q_1$ and $q_2$: we can determinize
- $f(q, q) \rightarrow q, 1 \neq 2$: 
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are $i = j$ or $i \neq j$.

$f(q_1, q_2, q_3) \rightarrow_{1=2, \ 1\neq3}$ allowed, $f(q_1, q_2) \rightarrow_{1=21}$ forbidden.

- can be determinized (exponential blow up)
- stable under boolean operations.
- Emptiness $EXPTIME$—complete but polynomial in the deterministic case.

Proof for ”normal” TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

- $f(q_1, q_2) \rightarrow q, 1 = 2$: if the automaton is not deterministic, we have to check whether there is some tree leading both to $q_1$ and $q_2$: we can determinize
- $f(q, q) \rightarrow q, 1 \neq 2$: We need to count, but we just have to count up to maximal arity.
Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are $i = j$ or $i \neq j$.

$f(q_1, q_2, q_3) \rightarrow_{1=2, 1\neq3}$ allowed, $f(q_1, q_2) \rightarrow_{1=21}$ forbidden.

- can be determinized (exponential blow up)
- stable under boolean operations.
- Emptiness $EXPTIME$–complete but polynomial in the deterministic case.

Proof for ”normal” TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

- $f(q_1, q_2) \rightarrow q, 1 = 2$: if the automaton is not deterministic, we have to check whether there is some tree leading both to $q_1$ and $q_2$: we can determinize
- $f(q, q) \rightarrow q, 1 \neq 2$: We need to count, but we just have to count up to maximal arity.

Automata with Constraints between brothers

Constraints are reduced to constraints between brothers (siblings)

Allowed constraints are \( i = j \) or \( i \neq j \).

\[
f(q_1, q_2, q_3) \rightarrow_{1=2, 1\neq3} \text{ allowed, } f(q_1, q_2) \rightarrow_{1=21} \text{ forbidden.}
\]

- can be determinized (exponential blow up)
- stable under boolean operations.
- Emptiness \( EXPTIME \)–complete but polynomial in the deterministic case.

Proof for ”normal” TA: compute by fix-point the set of reachable states and check it contains a final state. Does it work here?

- \( f(q_1, q_2) \rightarrow q, 1 = 2 \): if the automaton is not deterministic, we have to check whether there is some tree leading both to \( q_1 \) and \( q_2 \): we can determinize
- \( f(q, q) \rightarrow q, 1 \neq 2 \): We need to count, but we just have to count up to maximal arity.


- Recognizability is decidable.
Reduction automata

Limit the number of equality tests along a path

A reduction automaton $\mathcal{A}$ is an automaton with equality and disequality constraints with an ordering on the states such that for each rule $f(q_1, \ldots, q_n) \xrightarrow{c} q$,

- $q$ is maximal in $q, q_1, \ldots, q_n$
- $q$ is a strict upper bound of $q_1, \ldots, q_n$ if $c$ contains an equality constraint.

Idea: Bound the number of equalities tested along a path. Defined/studied by Dauchet, Caron, Coquidé, Comon and Jacquemard (94 and following)
Reduction automata

An example

How to define the set of ground instances of $f(f(x, x), f(x, y))$?
Reduction automata

An example

How to define the set of ground instances of $f(f(x, x), f(x, y))$?

$$\Delta =$$

\[
\begin{align*}
    a & \rightarrow q \\
    f(q, q) & \rightarrow q_f \\
    f(q_f, q) & \rightarrow q_f \\
    f(q_f, q_f) & \xrightarrow{11 = 12 \land 11 = 21} q_{fin}
\end{align*}
\]
Reduction automata
An example

How to define the set of ground instances of $f(f(x, x), f(x, y))$?

$\Delta =$

\[ a \rightarrow q \]
\[ f(q, q) \rightarrow q_f \]
\[ f(q_f, q) \rightarrow q_f \]
\[ f(q_f, q_f) \xrightarrow{11=12 \land 11=21} q_{fin} \]

$q < q_f < q_{fin}$
Reduction automata
An example

How to define the set of ground instances of $f(f(x, x), f(x, y))$?

$\Delta =$

\[
\begin{align*}
  a & \rightarrow q \\
  f(q, q) & \rightarrow q_f \\
  f(q_f, q) & \rightarrow q_f \\
  f(q_f, q_f) & \xrightarrow{11=12\land 11=21} q_{fin}
\end{align*}
\]

$\triangleright\ q < q_f < q_{fin}$

$\triangleright\ q_{fin}$ final state
How to define the set of ground instances of \( f(f(x, x), f(x, y)) \) with \( \Sigma = \{a, f\} \).

Deterministic version:

\[
\Delta =
\begin{align*}
    a & \rightarrow q \\
    f(q, q_f) & \rightarrow q_f \\
    f(q_f, q_f) & \xrightarrow{11=12\land11=21} q_{fin} \\
    f(q, q) & \rightarrow q_f \\
    f(q_f, q) & \rightarrow q_f \\
    f(q_f, q_f) & \xrightarrow{11\neq12\lor11\neq21} q_f
\end{align*}
\]
Reduction automata

Properties

- closed under union and intersection
Reduction automata

Properties

- closed under union and intersection
- closed under complement for deterministic automata
Reduction automata

Properties

- closed under union and intersection
- closed under complement for deterministic automata
- emptiness decidable for deterministic automata (M. Dauchet and alt. 1993)
Reduction automata
Properties

- closed under union and intersection
- closed under complement for deterministic automata
- emptiness decidable for **deterministic** automata (M. Dauchet and alt. 1993)
- emptiness **undecidable** for non deterministic automata (F. Jacquemard and alt. 2008)
Deciding emptiness ”needs” determinism
Automata with only one kind of tests

- $TA\neq$: automata with only disequality tests
  - closed under union and intersection
  - emptiness
Automata with only one kind of tests

$TA\neq$: automata with only disequality tests

- closed under union and intersection
- emptiness decidable
Automata with only one kind of tests

- $TA_{\neq}$: automata with only disequality tests
  - closed under union and intersection
  - emptiness decidable in EXPTIME (Comon & Jacquemard, 97)

- $TA_{=}$: automata with only equality tests
  - closed under union and intersection
  - emptiness
Automata with only one kind of tests

- $TA_{\neq}$: automata with only disequality tests
  - closed under union and intersection
  - emptiness decidable in EXPTIME (Comon & Jacquemard, 97)

- $TA_{=}$: automata with only equality tests
  - closed under union and intersection
  - emptiness undecidable
Automata with only one kind of tests

- The complement of a $TA_=$ is a $TA_\neq$. 
Automata with only one kind of tests

- The complement of a $TA_=$ is a $TA_\neq$.
- The complement of a $TA_\neq$ is a $TA_=$.

Sketch of the (effective) proof: a state reached by $t$ in the automaton for the complement of $L$ corresponds to the set of all states that $t$ can’t reach in the automaton for $L$. (folklore construction, formalized by Creus & alt. STOC 2010).
Automata with only one kind of tests

- The complement of a $TA_-$ is a $TA_\neq$.
- The complement of a $TA_\neq$ is a $TA_-$. 

Sketch of the (effective) proof: a state reached by $t$ in the automaton for the complement of $L$ corresponds to the set of all states that $t$ can’t reach in the automaton for $L$. (folklore construction, formalized by Creus & alt. STOC 2010).

Universality is decidable for $TA_-$, undecidable for $TA_\neq$. 
The homomorphism Problem

Hom—problem

Instance A recognizable language \( R \), an homomorphism \( H \).

Answer “yes” if and only \( H(R) \) is recognizable
The homomorphism Problem

Instance A recognizable language $R$, an homomorphism $H$.

Answer “yes” if and only if $H(R)$ is recognizable.

Is it decidable?
The homomorphism Problem

*Hom*—problem

**Instance** A recognizable language $R$, an homomorphism $H$.

**Answer** “yes” if and only if $H(R)$ is recognizable.

Is it decidable?

Remark: If $H$ is shallow -i.e the images of symbols are shallow-,
The homomorphism Problem

Hom—problem

**Instance**  A recognizable language $R$, an homomorphism $H$.

**Answer**  “yes” if and only $H(R)$ is recognizable

Is it decidable?

Remark: If $H$ is shallow -i.e the images of symbols are shallow-, $H(R)$ is recognizable by an automaton with tests between brothers. as recognizability is decidable for language recognizable by an automaton with tests between brothers: the homomorphism problem is decidable for shallow homomorphisms.
The homomorphism Problem

Proposition

*The HOM problem is decidable* (Godoy & alt, STOC 2010) and *EXPTIME-Complete* (Creus & alt., LICS 2012)

- $TA_{Hom}$: equality test without superposition, no disequality tests: $t \xrightarrow{c} q$ where $t \in T(\Sigma, Q)$, and $c$ is a Boolean combination of equality constraints for positions of $t$ labeled by states.
  - capture homomorphic images of recognizable tree languages
  - emptiness, finiteness, recognizability are decidable for this class.

- $TA_{Hom,\neq}$: equality and disequality tests, no superposition of equality tests: emptiness is decidable
The image of a recognizable tree language by a bottom-up tree transducer can be represented finitely by a $TA_{Hom}$.

E.g., we can decide whether $T_1(L_1) \subset T_2(L_2)$, $L_1, L_2$ recognizable tree languages, $T_1, T_2$ bottom-up tree transducers.
What about global constraints?
Tree Automata with Global Constraints
Local Constraints
- non linear pattern matching
- non-linear transformation
- term rewriting, reachability analysis

Global Constraints
- tree structured data processing
- schema = type + integrity constraints
Type Definition for XML Data

**DTD**

```
entrylist
  entry*
    name
    email
      first
      last
      user
      dom
        char*  char*  char*  char*
```

**Tree Automaton**

```
entry(\(p_n, p_m\)) \rightarrow p_e
empty \rightarrow p_{el}
entrylist(\(p_e, p_{el}\)) \rightarrow p_{el}
name(\(p_f, p_l\)) \rightarrow p_n
first(\(p\)) \rightarrow p_f
last(\(p\)) \rightarrow p_l
email(\(p_u, p_d\)) \rightarrow p_m
user(\(p\)) \rightarrow p_u
dom(\(p\)) \rightarrow p_d
a(\(p\)) \rightarrow p
... 
\varepsilon \rightarrow p
```
Integrity Constraint

entrylist

entry

name

first

char*

last

char*

email

user

char*

dom

char*

email is a key (ID)

subtrees below email are pairwise distinct

Cannot be expressed with standard tree automata
Tree Automata with Local Constraints

- equality and disequality test in transitions
- checked during computation

Tree Automata with Global Constraints

- one standard tree automaton + one separate constraint
- checked on run, after computation
Tree

entrylist
  entry
    name
    email
  entry
    name
    email
  entry
    name
    email

first: Homer
last: Simpson
user: homer
dom: gmail.com

first: John
last: To
user: dito
dom: gmail.com

first: Beth
last: Ditto
user: dito
dom: blip.fm
Tree Automaton Run

entry_{list}^{pe} — entry^{pe}

entry^{pe} — name^{pn}

entry^{pe} — email^{pm}

entry^{pe} — user^{pu}

entry^{pe} — dom^{pd}

first^{pf} — H^{p} o^{p} m^{p} e^{p} r^{p}

last^{pl} — S^{p} i^{p} m^{p} p^{p} s^{p} o^{p} n^{p}

user^{pu} — h^{p} o^{p} m^{p} e^{p} r^{p}

dom^{pd} — g^{p} m^{p} a^{p} i^{p} l^{p} c^{p} o^{p} m^{p}

first^{pf} — J^{p} o^{p} h^{p} n^{p}

last^{pl} — T^{p} o^{p}

user^{pu} — d^{p} i^{p} t^{p} o^{p}

dom^{pd} — g^{p} m^{p} a^{p} i^{p} l^{p} c^{p} o^{p} m^{p}

first^{pf} — B^{p} e^{p} t^{p} h^{p}

last^{pl} — D^{p} i^{p} t^{p} o^{p}

user^{pu} — d^{p} i^{p} t^{p} o^{p}

dom^{pd} — b^{p} l^{p} i^{p} f^{p} m^{p}
Key Constraint

email is a key: \( \forall x, y \ p_m(x) \wedge p_m(y) \wedge x \neq y \Rightarrow t|x \neq t|y \)
Global Equality Constraint

all domain’s coincide: \( \forall x, y \, \text{pd}(x) \land \text{pd}(y) \Rightarrow t\mid_x = t\mid_y \)
Negation

user is a not a key:  \( \exists x, y \, p_u(x) \land p_u(y) \land x \neq y \land t|_x = t|_y \)
A TAGC $\mathcal{A}$ is given by

- a tree automaton $ta(\mathcal{A}) = \langle \Sigma, Q, F, \Delta \rangle$
- a constraint $\phi$: FO formula interpreted on a term $t \in \mathcal{T}(\Sigma)$ and a run $r$ of $ta(\mathcal{A})$ on $t$, build with predicates
  - $x = y$ (node equality)
  - $p(x)$ (the label of $x$ in $r$ is $p$)
  - $t\mid x = t\mid y$ (subterm equality)

A relabeling $r : dom(t) \rightarrow Q$ is a successful run of $\mathcal{A}$ on $t$ iff

- $r$ run of $ta(\mathcal{A})$ on $t$
- $r(root) \in F$
- $t, r \models \phi$.

Language $\mathcal{L}(\mathcal{A}) = \{ t \in \mathcal{T}(\Sigma) \mid \exists r \text{ successful run of } \mathcal{A} \text{ on } t \}$. 
### Constraints: Notations

<table>
<thead>
<tr>
<th>type</th>
<th>formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1 \approx p_2$</td>
<td>$\forall x, y \ p_1(x) \land p_2(y) \Rightarrow t</td>
</tr>
<tr>
<td>$p \approx_{ref} p$</td>
<td>subcase with same states</td>
</tr>
<tr>
<td>$p_1 \approx_{irr} p_2$</td>
<td>subcase with distinct states</td>
</tr>
<tr>
<td>$p_1 \not\approx p_2$</td>
<td>$\forall x, y \ p_1(x) \land p_2(y) \land x \neq y \Rightarrow t</td>
</tr>
<tr>
<td>$p \not\approx_{ref} p$</td>
<td>subcase with same states (key)</td>
</tr>
<tr>
<td>$p_1 \not\approx_{irr} p_2$</td>
<td>subcase with distinct states</td>
</tr>
</tbody>
</table>

Note that $\neg p_1 \approx p_2$ and $p_1 \not\approx p_2$ have different semantics.

Notation $\text{TAGC}[\tau_1, \ldots, \tau_n]$: the constraint is a Boolean combination of atomic constraints of types $\tau_1, \ldots, \tau_n$.

$\text{TAGC}^\land[\tau_1, \ldots, \tau_n]$: when constraint is positive conjunctive.
Rest of the talk

some subclasses of TAGC[≈, ̸≈] studied

\[
\begin{align*}
\text{TAGED} & \equiv \text{TAGC}^\wedge[≈, ̸≈_{\text{irr}}] \quad \text{[Fliot, Talbot, Tison 2007, 08]} \\
\text{RTA} & \equiv \text{TAGC}^\wedge[≈_{\text{ref}}] \quad \text{[J Klay Vacher 2009, 11]} \\
\text{Dag Automata} & \equiv \text{TAGC}^\wedge[̸≈_{\text{irr}}] \quad \text{[Charatonik 1999]} \\
\text{TAGC}[≈, ̸≈] & \quad \text{[Godoy et al 2010]}
\end{align*}
\]

decidable extensions

\[
\begin{align*}
\text{TAGC}[≈, ̸≈, \mathbb{N}] & \quad \text{arithmetic} \quad \text{[Godoy et al 2010,12]} \\
\text{TACB}[≈, ̸≈, \mathbb{N}] & \quad \text{with local equality tests} \\
& \quad + \text{modulo}
\end{align*}
\]

open classes

\[
\begin{align*}
\text{inclusion constraints}
\end{align*}
\]
TAGED [Fliiot, Talbot, Tison 2007, 08]

TAGED := TAGC^[\approx, \not\approx_{irr}] \text{ (positive conjunctive)}

decision tool for TQL spatial logic [Cardelli, Ghelli 2002].

Example: all domain's coincide: with $p_d \approx p_d$

```
entry^{pe}\leftarrow entry^{pe}\leftarrow entry^{pe}\leftarrow entry^{pe}
```

```
name^{pn}\leftarrow name^{pn}\leftarrow name^{pn}
```

```
email^{pm}\leftarrow email^{pm}\leftarrow email^{pm}
```

```
list^{pel}\leftarrow list^{pel}\leftarrow list^{pel}\leftarrow list^{pel}
```

```
first^{pf}\leftarrow first^{pf}\leftarrow first^{pf}
```

```
last^{pl}\leftarrow last^{pl}\leftarrow last^{pl}
```

```
user^{pu}\leftarrow user^{pu}\leftarrow user^{pu}
```

```
dom^{pd}\leftarrow dom^{pd}\leftarrow dom^{pd}
```

```
H\text{o}m\text{e}\text{r}\text{p}
```

```
S\text{i}m\text{p}s\text{o}n\text{p}
```

```
h\text{o}m\text{e}\text{r}\text{p}
```

```
h\text{o}m\text{e}\text{r}\text{p}
```

```
d\text{i}t\text{o}\text{p}
```

```
g\text{m}a\text{i}l\text{.c\text{o}m}
```

```
g\text{m}a\text{i}l\text{.c\text{o}m}
```

```
J\text{o}h\text{n}\text{p}
```

```
T\text{o}\text{p}
```

```
d\text{i}t\text{o}\text{p}
```

```
g\text{m}a\text{i}l\text{.c\text{o}m}
```

```
g\text{m}a\text{i}l\text{.c\text{o}m}
```
all domain’s coincide, there are at least two different email’s:
with \( p_d \approx p_d \land p'_m \not\approx p''_m \) and email(\( p_u \ p_d \)) \( \rightarrow p_m \ | \ p'_m \ | \ p''_m \), and
transitions such that \( p'_m, p''_m \) occur exactly once in a successful run.
TAGED: Closure Properties

The class of TAGED- (TAGC[^≈, irr]) languages is effectively closed under

- **∪**: linear construction
  disjoint union of automata and global constraints

- **∩**: quadratic construction (Cartesian product)

<table>
<thead>
<tr>
<th>$A_1 \cap A_2$</th>
<th>$A_1$</th>
<th>$A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1 \times Q_2$</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
</tr>
</tbody>
</table>

$\langle p_1, p_2 \rangle \approx \langle p'_1, p'_2 \rangle$ iff $p_1 \approx p'_1$ or $p_2 \approx p'_2$

$\langle p_1, p_2 \rangle \not\approx \langle p'_1, p'_2 \rangle$ iff $p_1 \not\approx p'_1$ or $p_2 \not\approx p'_2$

- not $\neg$

  \{trees with subtree $g(x, y) \mid x \neq y$\} is in TAGED, not its complement.

- not determinizable
TAGED: Decision 1

Membership is NP-complete

hardness: reduction of satisfiability of Boolean expressions.
Ex. \((x \lor y) \land (\neg x \lor y)\)

\[
\begin{array}{c}
\land \\
\lor \\
\land \\
\lor \\
\land \\
\end{array}
\begin{array}{ccc}
0 & 1 & 0 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
\end{array}
\begin{array}{c}
x \\
y \\
\neg \\
y \\
x \\
\end{array}
\]

This TAGED is in \(\text{TAGC}^\Lambda [\approx_{\text{ref}}]\)
Universality and inclusion are undecidable

Reduction of PCP into $\text{TAGC}^{\approx_{\text{ref}}}$ universality.

Emptiness

- EXPTIME-complete for $\text{TAGC}^{\approx}$
- NEXPTIME for $\text{TAGC}^{\not\approx_{\text{irr}}}$
- decidable for subclass of TAGED testing bounded number of $\approx, \not\approx_{\text{irr}}$ [Filiot et al 2007 CSL]
- decidable for subclass of TAGED testing bounded number of $\not\approx_{\text{irr}}$ [Filiot et al 2008 DLT]
Rigid Tree Automata

RTA = subclass $\text{TAGC}^\wedge[\approx_{\text{ref}}]$ of TAGED

for data flow analysis of security protocols

$\approx$-constraints are sufficient for non-linear pattern matching
(bounded number of local equality tests)

Ex. for the pattern $f(x,x)$,

\begin{align*}
a & \rightarrow q|x, \\
b & \rightarrow q|x, \\
f(q,q) & \rightarrow q|x, \\
f(q_x,q_x) & \rightarrow q_f \\
q_x & \approx q_x
\end{align*}

successful run of $\mathcal{A}$ on $f(f(a,b), f(a,b))$: $q_f(q_x(q,q), q_x(q,q))$. 
RTA: Expressiveness

\[ \text{RTA} \subsetneq \text{TAGED} \]

\[ \text{TAGC}^\approx \equiv \text{TAGC}^{\approx_{\text{ref}}} \quad \text{[Filiot et al 2008 DLT]} \]

construction \( \rightarrow \) with exponential blowup.

**Determinism**

\[ \text{TA} \subsetneq \text{DRTA} \subsetneq \text{RTA} \]

\( \{ f(x, x) \mid x \in T(\Sigma) \} \) not recognizable by DRTA.

Subclass of visibly RTA determinizable.

Rewrite closure
(under equational specification of cryptographic operators)
Pumping Lemma

If \( t \in L(A) \) and \( \text{depth}(t) > (|Q| + 1)|R| \), then there exist

- a context \( C \),
- two contexts \( C' \) and \( D \), with \( D \) non-trivial,
- a term \( u \)

such that \( t = C[C'[D[u]], \ldots, C'[D[u]]] \)

and for all \( n \geq 0 \),

\[
C[C'[D^n[u]], \ldots, C'[D^n[u]]] \in L(A).
\]

The set of balanced binary trees is not a RTA language.
RTA: Closure

like TAGED

- $\cup$ (polynomial)
- $\cap$ (exponential):

$$2 \text{ RTA} \to 1 \text{ TAGC}^\sim$$

for $\cap \to 1$ RTA for $\cap$

It is a lower bound (reduction of the emptiness of intersections for tree automata).

- not $\neg$

The set of balanced binary trees is not RTA.

Its complement is recognized by

$$
\begin{align*}
a & \rightarrow q|q_r \\
f(q_r, q) & \rightarrow q'_r \\
f(q, q'_r) & \rightarrow q'_r \\
f(q'_r, q) & \rightarrow q'_r \\
f(q, q_r) & \rightarrow q' \\
f(q'_r, q_r) & \rightarrow q'_r \\
f(q_f, q) & \rightarrow q_f, \\
f(q, q_f) & \rightarrow q_f \\
f(q'_r, q'_r) & \rightarrow q_r \approx q_r
\end{align*}
$$
RTA: Decision

Decision

- membership is NP-complete
- universality, inclusion undecidable
- finiteness:
  - PTIME for RTA (TAGC[^ref]),
  - EXPTIME for TAGC[^\approx].
- emptiness decidable in linear time

For emptiness: same state marking algorithm as for tree automata.

- regularity is undecidable for RTA, TAGED, TAGC[^\approx]
Global vs Local Constraints

TAGED (RTA) and automata with local constraints are orthogonal.

- balanced binary trees $\not\in$ TAGED
- ”all domain’s coincide” $\not\in$ local constraints automata.

Let’s call two nodes equivalent in a successful run if they have been “successfully tested” to be equal in the run.

<table>
<thead>
<tr>
<th></th>
<th># of eq. classes</th>
<th>cardinality of eq. classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>local constraints</td>
<td>unbounded</td>
<td>bounded</td>
</tr>
<tr>
<td>TAGEDs</td>
<td>bounded</td>
<td>unbounded</td>
</tr>
</tbody>
</table>
DAG Automata

**DA:** tree automata computing (relabeling of nodes by states) on DAGs representations of ranked trees (maximal sharing).
**DAG Automata**

**DA:** tree automata computing (relabeling of nodes by states) on DAGs representations of ranked trees (maximal sharing).

Membership is NP-complete for DA

[Charatonik 1999], [Anantharaman et al 2005]

Membership is PTIME for deterministic DA

Tree membership is PTIME complete

[Lohrey Maneth 2006]
DAG Automata and Global $\approx$ Constraints

- **DA**: a unique state is associated to equal subtrees (same node in the DAG representation)
- **RTA**: a unique subtree is associated to every occurrence of the same state $q$ if $q \approx q$

DA and RTA are orthogonal.

$\{f(x, x)\}$ is not recognizable by DA.
DAG Automata and Global ∉ Constraints

Emptiness is NP-complete for DA [Charatonik 1999]

[Vacher 2010 PhD]

- $DA \equiv TAGC^\Delta[ \not\approx_{irr} ]$
  
  $\Rightarrow q_1 \not\approx q_2$ iff $q_1 \neq q_2$.
  
  $\Leftarrow$ states $2^Q \setminus \{P \mid \exists q_1, q_2 \in P, q_1 \not\approx q_2\}$
DAG Automata and Global $\not\sim$ Constraints

Emptiness is NP-complete for DA  

[Vacher 2010 PhD]

- DA $\equiv$ TAGC$^\wedge[\not\sim_{irr}]$
  - $q_1 \not\sim q_2 \iff q_1 \neq q_2$.
  - States $2^Q \setminus \{P \mid \exists q_1, q_2 \in P, q_1 \not\sim q_2\}$

- TAGC$^\wedge[\sim, \not\sim_{irr}] \equiv DA[\sim]$
  - DA$[\sim]$: $q_1 \approx q_2 := \forall x, y \in \text{dom}(d) \ q_1(x) \land q_2(y) \Rightarrow x = y$

- Emptiness is still NP-complete for DA$[\sim]$
- Emptiness is decidable in NEXPTIME for TAGED

Open problem: generalization to TAGC$^\wedge[\sim, \not\sim]$  
(elementary upper bound for emptiness decision)
## Closure and Decision Results

<table>
<thead>
<tr>
<th></th>
<th>TA $\text{TAGC}[]$</th>
<th>RTA $\text{TAGC}^\approx_{\text{ref}}$</th>
<th>DA $\text{TAGC}^\not\approx_{\text{irr}}$</th>
<th>TAGED $\text{TAGC}^\approx,\not\approx_{\text{irr}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\cap$</td>
<td>PTIME</td>
<td>EXPTIME</td>
<td>not</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>$\neg$</td>
<td>EXPTIME</td>
<td>not</td>
<td>not</td>
<td>not</td>
</tr>
<tr>
<td>emptiness</td>
<td>linear time</td>
<td>linear time</td>
<td>NP-c.</td>
<td>NEXPTIME-c.</td>
</tr>
<tr>
<td>membership</td>
<td>PTIME</td>
<td>NP-c.</td>
<td>NP-c.</td>
<td>NP-c.</td>
</tr>
<tr>
<td>$\cap$-emptiness</td>
<td>EXPTIME-c.</td>
<td>EXPTIME-c.</td>
<td>EXPTIME-c.</td>
<td>EXPTIME-c.</td>
</tr>
<tr>
<td>universality</td>
<td>EXPTIME-c.</td>
<td>undec.</td>
<td>undec.</td>
<td>undec.</td>
</tr>
<tr>
<td>inclusion</td>
<td>EXPTIME-c.</td>
<td>undec.</td>
<td>undec.</td>
<td>undec.</td>
</tr>
<tr>
<td>finiteness</td>
<td>PTIME</td>
<td>EXPTIME</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
</tbody>
</table>
Emptiness is decidable for TAGC[≃, ‡]

we use TAGC[≃, ‡] ≡ TAGC^[≃, ‡]

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

extension to unranked trees immediate, using encoding into binary trees [Godoy et al 2012]

on ranked trees, emptiness is still decidable for
- TAGC[$\approx$, $\not\approx$] extended with local $=$ and $\neq$ constraints between siblings, à la [Bogaert Tison 1992]
- TAGC[$\approx$, $\not\approx$] where $\approx$ and $\not\approx$ are interpreted modulo flat equational theories
Arithmetic Constraints

linear inequality \( \sum_{q \in Q} a_q \cdot |q| \geq a \) or \( \sum_{q \in Q} a_q \cdot \|q\| \geq a \), \( a, a_q, a \in \mathbb{Z} \)

for a run \( r \) on a tree \( t \), 
\[
|q| = |r^{-1}(q)| \\
\|q\| = \{|t_x| \mid x \in \text{dom}(t), r(x) = q\}|
\]

natural inequality (type ‘\( \mathbb{N} \)’) when all \( a_q, a \) have the same sign

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

- emptiness decidable in NPTIME for TAGC[\( |.|\mathbb{Z} \)]
- emptiness undecidable for TAGC[\( \approx, |.|\mathbb{Z} \)] [Godoy et al 2010]
- TAGC[\( \approx, \nexists, |.|\mathbb{N}, \|.\|.\mathbb{N} \) \( \equiv \) TAGC^\[\approx, \nexists\] [id]
Monadic Second Order Logic

\[ \text{MSO}[+1, \approx, \not\approx, |.|_Z, \|\|_Z] \] monadic second-order logic

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

with predicates

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

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

EMSO: \exists X_1 \ldots \exists X_n \phi(X_1, \ldots, X_n) \land \psi(X_1, \ldots, X_n)\text{ where}

- \phi(X_1, \ldots, X_n) in MSO[+1]
- \psi(X_1, \ldots, X_n) in MSO[+1, \approx, \not\approx, \lor, \land, \exists, \forall, \mathbb{Z}, \parallel, \parallel ]\text{, free}

- EMSO[+1, \mathbb{Z}] decidable [Klaedtke Ruess 2002]
- fragment of EMSO[+1, \approx, \not\approx] decidable [Filiot et al 2008]
- EMSO[+1, \approx, \not\approx, \lor, \land, \exists, \forall, \mathbb{N}, \parallel, \parallel \mathbb{N}] decidable [Godoy et al 2010]
Perspectives

- Combining Local/Global Constraints for unranked ordered trees
  - combination of TAGC[\(\approx, \not\approx\)]
  - with UTASC: unranked tree automata with local sibling constraints [Löding Wong 2007, 09]

- Extension of TAGC for inclusion constraints

\[
\forall x \exists y \ p(x) \implies (q(y) \land t_x = t_y) \quad (x, y \in \text{positions}) \\
\forall u \exists v \ p(u) \implies (q(v) \land u = v) \quad (u, v \in \text{subtrees})
\]

- TAGC with constraints in monadic FO over \(q(y)\) and \(x = y\)
  interpretation in the domain of subtrees
  (related to automata on DAGs)

Thank you
Thank you