Algorithms for Tree Automata with Constraints

Efficiently tackling the Emptiness Problem for Tree Automata With Global Equality Constraints

Pierre-Cyrille Héam, Vincent Hugot, Olga Kouchnarenko

{pcheam,okouchnarenko}@lifc.univ-fcomte.fr,

vhugot@edu.univ-fcomte.fr

Université de Franche-Comté
LIFC-INRIA/CASSIS, project ACCESS

July 7, 2010

Plan of the talk

1. **Introduction of notions:**
   1. Vanilla Tree Automata
   2. Tree Automata with Constraints: TAGEDs
   3. The emptiness problem
2. **A general strategy:**
   1. Global algorithm
   2. Remarks on experimental protocol
3. **Proposed tactics:**
   1. Cleanup: hunting for spuriousness
   2. Signature quotienting
   3. Parenting relations
   4. A brutal algorithm
4. **Conclusion.**

Plan of the talk

1. **Introduction of notions:**
    1. Vanilla Tree Automata
    2. Tree Automata with Constraints: `TAGED`s
    3. The emptiness problem
2. **A general strategy:**
    1. Global algorithm
    2. Remarks on experimental protocol
3. **Proposed tactics:**
    1. Cleanup: hunting for spuriousness
    2. Signature quotienting
    3. Parenting relations
    4. A brutal algorithm
4. **Conclusion.**

Plan of the talk

1. **Introduction of notions:**
    1. Vanilla Tree Automata
    2. Tree Automata with Constraints: `TAGEDs`
    3. The emptiness problem
2. **A general strategy:**
    1. Global algorithm
    2. Remarks on experimental protocol
3. **Proposed tactics:**
    1. Cleanup: hunting for spuriousness
    2. Signature quotienting
    3. Parenting relations
    4. A brutal algorithm
4. **Conclusion.**

Plan of the talk

1. **Introduction of notions:**
   1. Vanilla Tree Automata
   2. Tree Automata with Constraints: `TAGED`s
   3. The emptiness problem
2. **A general strategy:**
   1. Global algorithm
   2. Remarks on experimental protocol
3. **Proposed tactics:**
   1. Cleanup: hunting for spuriousness
   2. Signature quotienting
   3. Parenting relations
   4. A brutal algorithm
4. **Conclusion.**

- **Tree automata**: powerful theoretical tools useful for
  - automated theorem proving
  - program verification
  - XML schema and query languages
  - ...
- **Extensions**: created to expand expressiveness.
- **Problem**: decidability and complexity of associated decision problems. Usable tools difficult to implement.
- Theme of my Master's project and internship: **efficient algorithms** for tree automata with constraints. For this internship: emptiness problem for positive TAGEDs (*EXPTIME*-complete)

- **Tree automata**: powerful theoretical tools useful for
  - automated theorem proving
  - program verification
  - XML schema and query languages
  - ...
- **Extensions**: created to expand expressiveness.
- **Problem**: decidability and complexity of associated decision problems. Usable tools difficult to implement.
- Theme of my Master's project and internship: **efficient algorithms** for tree automata with constraints. For this internship: emptiness problem for positive TAGEDs (*EXPTIME*-complete)

- **Tree automata**: powerful theoretical tools useful for
  - automated theorem proving
  - program verification
  - XML schema and query languages
  - ...
- **Extensions**: created to expand expressiveness.
- **Problem**: decidability and complexity of associated decision problems. Usable tools difficult to implement.
- Theme of my Master's project and internship: **efficient algorithms** for tree automata with constraints. For this internship: emptiness problem for positive TAGEDs (*EXPTIME*-complete)

- **Tree automata**: powerful theoretical tools useful for
  - automated theorem proving
  - program verification
  - XML schema and query languages
  - ...
- **Extensions**: created to expand expressiveness.
- **Problem**: decidability and complexity of associated decision problems. Usable tools difficult to implement.
- Theme of my Master's project and internship: **efficient algorithms** for tree automata with constraints. For this internship: emptiness problem for positive TAGEDs (*EXPTIME*-complete)

Tree automaton for True propositional formulæ

$$\mathcal{A} \stackrel{\text{def}}{=} \left(\Sigma = \{\, \wedge, \vee/_2, \neg/_1, 0, 1/_0 \,\},\ Q = \{\, q_0, q_1 \,\}, F = \{\, q_1 \,\}, \Delta\right)$$

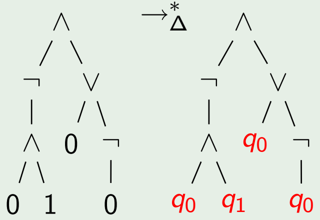
$$\begin{aligned} \Delta = \{ & b \rightarrow q_b, \\ & \wedge\,(q_b, q_{b'}) \rightarrow q_{b \wedge b'}, \\ & \vee\,(q_b, q_{b'}) \rightarrow q_{b \vee b'}, \\ & \neg(q_b) \rightarrow q_{\neg b} \\ & \mid\ b, b' \in 0, 1\} \end{aligned}$$

```
     ∧
   /   \
  ¬     ∨
  |    / \
  ∧   0   ¬
 / \      |
0   1     0
```

Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.

$0 \rightarrow q_0, 1 \rightarrow q_1 \in \Delta$

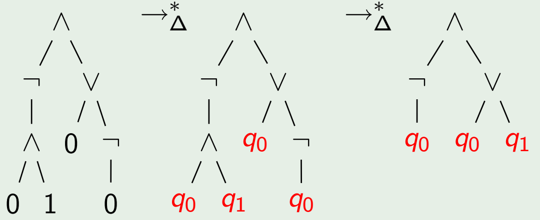


Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.

Tree automata
Definition through an example

$\wedge(q_0, q_1) \to q_0, \neg(q_0) \to q_1 \in \Delta$



Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.

$\neg(q_0) \to q_1,\ \vee(q_0, q_1) \to q_1 \in \Delta$

$$\wedge(\neg(\wedge(0,1)),\ \vee(0,\neg(0))) \xrightarrow{*}_{\Delta} \wedge(\neg(\wedge(q_0,q_1)),\ \vee(q_0,\neg(q_0))) \xrightarrow{*}_{\Delta} \wedge(\neg(q_0),\ \vee(q_0,q_1)) \xrightarrow{*}_{\Delta} \wedge(q_1,q_1)$$

Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

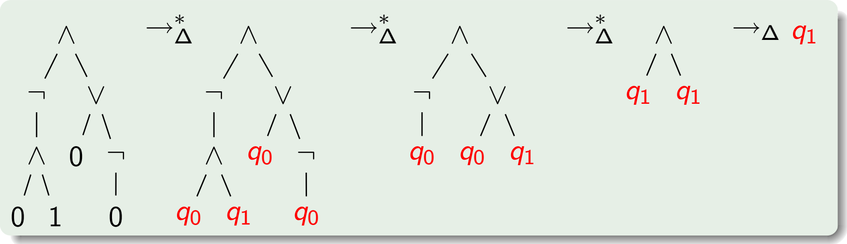
A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.

$\wedge(q_1, q_1) \to q_1 \in \Delta$

$$\wedge(\neg(\wedge(0,1)), \vee(0, \neg(0))) \xrightarrow{*}_{\Delta} \wedge(\neg(\wedge(q_0,q_1)), \vee(q_0, \neg(q_0))) \xrightarrow{*}_{\Delta} \wedge(\neg(q_0), \vee(q_0, q_1)) \xrightarrow{*}_{\Delta} \wedge(q_1, q_1) \to_{\Delta} q_1$$

Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.



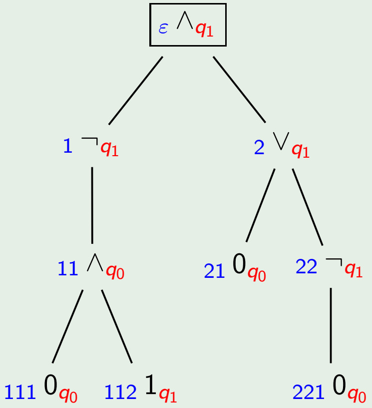
Definition: run of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$

A run ρ is a mapping from $\mathcal{P}os(t)$ to Q compatible with the transition rules.

Tree automata

Definition through an example

$\rho =$



Introduced in Emmanuel Filiot's PhD thesis on XML query languages. See [Filiot et al., 2008].

A TAGED is a tuple $\mathcal{A} = (\Sigma, Q, F, \Delta, =_{\mathcal{A}}, \neq_{\mathcal{A}})$, where

- (Σ, Q, F, Δ) is a tree automaton
- $=_{\mathcal{A}}$ is a reflexive symmetric binary relation on a subset of Q
- $\neq_{\mathcal{A}}$ is an irreflexive and symmetric binary relation on Q. Note that in our work, we have dealt with a slightly more general case, where $\neq_{\mathcal{A}}$ is not necessarily irreflexive.

A TAGED $\mathcal{A}$ is said to be *positive* if $\neq_{\mathcal{A}}$ is empty and *negative* if $=_{\mathcal{A}}$ is empty.

Runs must be compatible with equality and disequality constraints.

Introduced in Emmanuel Filiot's PhD thesis on XML query languages. See [Filiot et al., 2008].

A TAGED is a tuple $\mathcal{A} = (\Sigma, Q, F, \Delta, =_\mathcal{A}, \neq_\mathcal{A})$, where

- (Σ, Q, F, Δ) is a tree automaton
- $=_\mathcal{A}$ is a reflexive symmetric binary relation on a subset of Q
- $\neq_\mathcal{A}$ is an irreflexive and symmetric binary relation on Q. Note that in our work, we have dealt with a slightly more general case, where $\neq_\mathcal{A}$ is not necessarily irreflexive.

A TAGED $\mathcal{A}$ is said to be *positive* if $\neq_\mathcal{A}$ is empty and *negative* if $=_\mathcal{A}$ is empty.

Runs must be compatible with equality and disequality constraints.

Le ρ be a run of the TAGED $\mathcal{A}$ on a tree t:

Compatibility with the equality constraint $=_\mathcal{A}$

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \rho(\alpha) =_\mathcal{A} \rho(\beta) \implies t|_\alpha = t|_\beta \,.$$

Compatibility with the disequality constraint $\neq_\mathcal{A}$ (irreflexive)

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \rho(\alpha) \neq_\mathcal{A} \rho(\beta) \implies t|_\alpha \neq t|_\beta \,.$$

Compatibility with the disequality constraint $\neq_\mathcal{A}$ (non irreflexive)

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \alpha \neq \beta \wedge \rho(\alpha) \neq_\mathcal{A} \rho(\beta) \implies t|_\alpha \neq t|_\beta \,.$$

Le ρ be a run of the TAGED $\mathcal{A}$ on a tree t:

Compatibility with the equality constraint $=_\mathcal{A}$

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \rho(\alpha) =_\mathcal{A} \rho(\beta) \implies t|_\alpha = t|_\beta\,.$$

Compatibility with the disequality constraint $\neq_\mathcal{A}$ (irreflexive)

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \rho(\alpha) \neq_\mathcal{A} \rho(\beta) \implies t|_\alpha \neq t|_\beta\,.$$

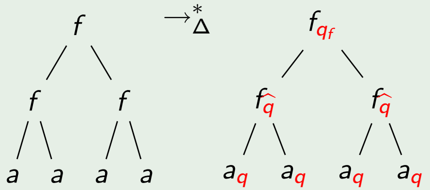
Compatibility with the disequality constraint $\neq_\mathcal{A}$ (non irreflexive)

$$\forall \alpha, \beta \in \mathcal{P}os(t) : \alpha \neq \beta \wedge \rho(\alpha) \neq_\mathcal{A} \rho(\beta) \implies t|_\alpha \neq t|_\beta\,.$$

TAGED for $\{\, f(t,t) \mid f \in \Sigma, t \in \mathcal{T}(\Sigma) \,\}$

$$\mathcal{A} \stackrel{\text{def}}{=} (\Sigma = \{\, a/_0, f/_2 \,\},\ Q = \{\, q, \widehat{q}, q_f \,\},\ F = \{\, q_f \,\},\ \Delta,\ \widehat{q} =_{\mathcal{A}} \widehat{q}),$$

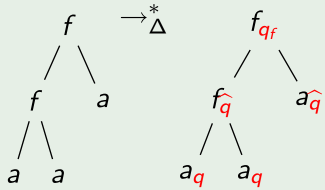
$$\text{where } \Delta \stackrel{\text{def}}{=} \{f(\widehat{q},\widehat{q}) \to q_f,\ f(q,q) \to q,\ f(q,q) \to \widehat{q},\ a \to q,\ a \to \widehat{q}, \}$$



TAGED for $\{\, f(t,t) \mid f \in \Sigma, t \in \mathcal{T}(\Sigma) \,\}$

$$\mathcal{A} \stackrel{\text{def}}{=} (\Sigma = \{\, a/_0, f/_2 \,\},\ Q = \{\, q, \widehat{q}, q_f \,\},\ F = \{\, q_f \,\},\ \Delta,\ \widehat{q} =_{\mathcal{A}} \widehat{q}),$$

where $\Delta \stackrel{\text{def}}{=} \{f(\widehat{q}, \widehat{q}) \to q_f,\ f(q,q) \to q,\ f(q,q) \to \widehat{q},\ a \to q,\ a \to \widehat{q}, \}$



TAGED emptiness

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}(\mathcal{A}) = \varnothing$?

Applications

- Introduced for XML query languages
- in model-checking...

Theorem [Filiot2008]

The Emptiness Problem for positive TAGEDs is *EXPTIME*-complete.

TAGED emptiness

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Applications

- Introduced for XML query languages
- in model-checking. . .

Theorem [Filiot2008]

The Emptiness Problem for positive TAGEDs is *EXPTIME*-complete.

TAGED emptiness

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Applications

- Introduced for XML query languages
- in model-checking. . .

Theorem [Filiot2008]

The Emptiness Problem for positive TAGEDs is *EXPTIME*-complete.

Global Strategy

A high-level view of how we tackled the problem

1. Part I: A strategy and several tactics
    1. Inexpensive reductions
    2. Splitting the TAGED
    3. Semi-expensive heuristics
    4. Brutal algorithm
2. Part II: experiments random TAGEDs
    1. Random generation of tree automata (4 generations)
    2. Random generation of constraints (3 generations)

Global Strategy

A high-level view of how we tackled the problem

1. Part I: A strategy and several tactics
   1. Inexpensive reductions
   2. Splitting the TAGED
   3. Semi-expensive heuristics
   4. Brutal algorithm
2. Part II: experiments random TAGEDs
   1. Random generation of tree automata (4 generations)
   2. Random generation of constraints (3 generations)

Global Strategy

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Reducing the problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{A}'$ a smaller positive TAGED.
$\twoheadrightarrow$ Standard reduction, *cleanup*, *signature-quotienting*

Quick negative decision

$\twoheadrightarrow$ $\mathcal{L}\mathrm{ng}\,(\mathfrak{ta}\,(\mathcal{A})) = \varnothing$?

Quick positive decision

$\twoheadrightarrow$ *parenting relations*

If all else fails

General exponential algorithm: *brutal algorithm*

Global Strategy

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Reducing the problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{A}'$ a smaller positive TAGED.
↠ Standard reduction, *cleanup*, *signature-quotienting*

Quick negative decision

↠ $\mathcal{L}\mathrm{ng}\,(\mathfrak{ta}\,(\mathcal{A})) = \varnothing$?

Quick positive decision

↠ *parenting relations*

If all else fails

General exponential algorithm: *brutal algorithm*

Global Strategy

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Reducing the problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{A}'$ a smaller positive TAGED.
$\twoheadrightarrow$ Standard reduction, *cleanup*, *signature-quotienting*

Quick negative decision

$\twoheadrightarrow$ $\mathcal{L}\mathrm{ng}\,(\mathfrak{ta}\,(\mathcal{A})) = \varnothing$?

Quick positive decision

$\twoheadrightarrow$ *parenting relations*

If all else fails

General exponential algorithm: *brutal algorithm*

Global Strategy

Emptiness Problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{L}\mathrm{ng}\,(\mathcal{A}) = \varnothing$?

Reducing the problem

INPUT: $\mathcal{A}$ a positive TAGED.
OUTPUT: $\mathcal{A}'$ a smaller positive TAGED.
$\twoheadrightarrow$ Standard reduction, *cleanup*, *signature-quotienting*

Quick negative decision

$\twoheadrightarrow$ $\mathcal{L}\mathrm{ng}\,(\mathfrak{ta}\,(\mathcal{A})) = \varnothing$?

Quick positive decision

$\twoheadrightarrow$ *parenting relations*

If all else fails

General exponential algorithm: *brutal algorithm*

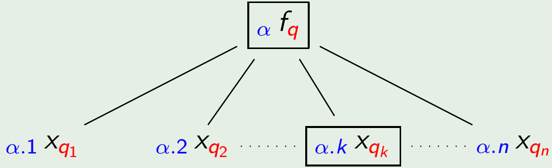
Cleanup

Improved version of standard reduction (reachability) algorithm for tree automata, which takes advantage of equality constraints to remove useless rules and states.

1. Spurious rules
2. Useless states
3. Σ-spurious states
4. Spurious states

Definition (Spurious rule)

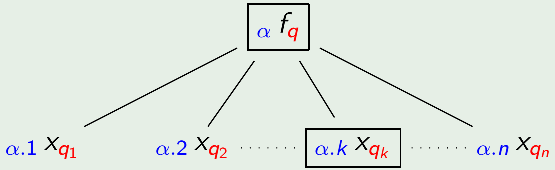
Let $\mathcal{A}$ be a TAGED. A rule $f(q_1, \ldots, q_n) \to q \in \Delta$ is *spurious* if there exists $k \in [\![1, n]\!]$ such that $q_k =_{\mathcal{A}} q$.



Cleanup: hunting for spuriousness
Spurious Rules

Definition (Spurious rule)

Let $\mathcal{A}$ be a $\mathtt{TAGED}$. A rule $f(q_1, \ldots, q_n) \to q \in \Delta$ is *spurious* if there exists $k \in [\![1, n]\!]$ such that $q_k =_{\mathcal{A}} q$.



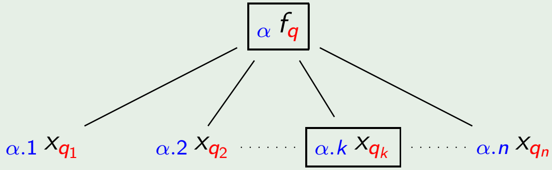
Lemma (Removal of spurious rules)

All spurious rules can be removed without altering the accepted language.

Cleanup: hunting for spuriousness
Spurious Rules

Definition (Spurious rule)

Let $\mathcal{A}$ be a TAGED. A rule $f(q_1, \ldots, q_n) \to q \in \Delta$ is *spurious* if there exists $k \in [\![1, n]\!]$ such that $q_k =_{\mathcal{A}} q$.



Proof idea

If a spurious rule was used, a term would have to be equal with one of its strict subterms. Which is absurd. □

Let $p_y^x, p, q \in Q$, $\sigma_1, \ldots, \sigma_m \in \Sigma$, and

$$\mathfrak{Rul}(q) = \left\{ \begin{array}{c} \sigma_1(p_1^1, \ldots, p_{n_1}^1, p, p_1'^1, \ldots, p_{n_1'}'^1) \to q \\ \vdots \\ \sigma_m(p_1^m, \ldots, p_{n_m}^m, p, p_1'^m, \ldots, p_{n_m'}'^m) \to q \end{array} \right\}$$

Sure requirements

$$p \in s\mathfrak{Req}(q)$$

Potential Requirements

$$p\mathfrak{Req}(q) = \{\, p \,\} \cup \left\{\, p_y^x, p_y'^x \;\middle|\; x, y \in \ldots \,\right\}$$

Let $p_y^x, p, q \in Q$, $\sigma_1, \ldots, \sigma_m \in \Sigma$, and

$$\mathfrak{Rul}(q) = \left\{ \begin{array}{c} \sigma_1(p_1^1, \ldots, p_{n_1}^1, p, p_1'^1, \ldots, p_{n_1'}'^1) \to q \\ \vdots \\ \sigma_m(p_1^m, \ldots, p_{n_m}^m, p, p_1'^m, \ldots, p_{n_m'}'^m) \to q \end{array} \right\}$$

Sure requirements

$$p \in \mathfrak{sReq}(q)$$

Potential Requirements

$$\mathfrak{pReq}(q) = \{\, p \,\} \cup \left\{ p_y^x, p_y'^x \;\middle|\; x, y \in \ldots \right\}$$

Let $p^x_y, p, q \in Q$, $\sigma_1, \dots, \sigma_m \in \Sigma$, and

$$\mathfrak{Rul}(q) = \left\{ \begin{matrix} \sigma_1(p^1_1, \dots, p^1_{n_1}, p, p'^1_1, \dots, p'^1_{n'_1}) \to q \\ \vdots \\ \sigma_m(p^m_1, \dots, p^m_{n_m}, p, p'^m_1, \dots, p'^m_{n'_m}) \to q \end{matrix} \right\}$$

Sure requirements

$$p \in \mathfrak{sReq}(q)$$

Potential Requirements

$$\mathfrak{pReq}(q) = \{\, p \,\} \cup \left\{ p^x_y, p'^x_y \;\middle|\; x, y \in \dots \right\}$$

Let $p_y^x, p, q \in Q$, $\sigma_1, \ldots, \sigma_m \in \Sigma$, and

$$\mathfrak{Rul}(q) = \left\{ \begin{array}{c} \sigma_1(p_1^1, \ldots, p_{n_1}^1, {\color{red}p}, p_1'^1, \ldots, p_{n_1'}'^1) \to q \\ \vdots \\ \sigma_m(p_1^m, \ldots, p_{n_m}^m, {\color{red}p}, p_1'^m, \ldots, p_{n_m'}'^m) \to q \end{array} \right\}$$

Sure requirements

$$\mathfrak{sReq}(q) \stackrel{\text{def}}{=} \bigcap_{\substack{r \in \mathfrak{Rul}(q) \\ q \notin \mathfrak{Ant}(r)}} \mathfrak{Ant}(r),$$

Potential Requirements

$$\mathfrak{pReq}(q) \stackrel{\text{def}}{=} \bigcup_{r \in \mathfrak{Rul}(q)} \mathfrak{Ant}(r).$$

$$\mathfrak{Frnd}(q) = \text{“transitive closure of } \mathfrak{pReq}(q)\text{”}.$$
$$\mathfrak{Need}(q) = \text{“transitive closure of } \mathfrak{sReq}(q)\text{”}.$$

Definition (Friend states)

$\mathfrak{Frnd}(q)$: the smallest subset of Q satisfying

1. $\mathfrak{pReq}(q) \subseteq \mathfrak{Frnd}(q)$
2. if $p \in \mathfrak{Frnd}(q)$ then $\mathfrak{pReq}(p) \subseteq \mathfrak{Frnd}(q)$

Definition (Needs)

$\mathfrak{Need}(q)$: smallest subset of Q satisfying

1. $\mathfrak{sReq}(q) \subseteq \mathfrak{Need}(q)$
2. if $p \in \mathfrak{Need}(q)$ then $\mathfrak{sReq}(p) \subseteq \mathfrak{Need}(q)$

"Only friends of q appear under q"

Lemma ("Rely on your Friends" principle)

Let ρ a run: $\forall \alpha, \beta \in \mathcal{P}os(t) : \beta \lhd \alpha \implies \rho(\beta) \in \mathfrak{Frnd}\left(\rho(\alpha)\right)$.

"Every need of q appears under q"

Lemma (Needs)

Let ρ a run such that $\rho(\beta) = q$. For any $p \in \mathfrak{Need}(q)$, there exists a position $\alpha_p \lhd \beta$ such that $\rho(\alpha_p) = p$.

"Only friends of q appear under q"

Lemma ("Rely on your Friends" principle)

Let ρ a run: $\forall \alpha, \beta \in \mathcal{P}os(t) : \beta \lhd \alpha \implies \rho(\beta) \in \mathfrak{Frnd}\left(\rho(\alpha)\right)$.

"Every need of q appears under q"

Lemma (Needs)

Let ρ a run such that $\rho(\beta) = q$. For any $p \in \mathfrak{Need}(q)$, there exists a position $\alpha_p \lhd \beta$ such that $\rho(\alpha_p) = p$.

"Only friends of a final state are useful"

Theorem (Removal of useless states)

Let $\mathcal{A} = (\Sigma, Q, F, \Delta)$ be a tree automaton. Then

$$\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}') \text{ with } \mathcal{A}' \stackrel{def}{=} \mathfrak{Rst}\left(\mathcal{A}, F \cup \bigcup_{q_f \in F} \mathfrak{Frnd}(q_f)\right).$$

Furthermore, the accepting runs are the same for $\mathcal{A}$ and $\mathcal{A}'$.

Proof idea

Every accepting run is rooted in a final state. Therefore they cannot use any state not in $F \cup \bigcup_{q_f \in F} \mathfrak{Frnd}(q_f)$. □

"Only friends of a final state are useful"

Theorem (Removal of useless states)

Let $\mathcal{A} = (\Sigma, Q, F, \Delta)$ *be a tree automaton. Then*

$$\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}') \text{ with } \mathcal{A}' \stackrel{def}{=} \mathfrak{Rst}\left(\mathcal{A}, F \cup \bigcup_{q_f \in F} \mathfrak{Frnd}(q_f)\right).$$

Furthermore, the accepting runs are the same for $\mathcal{A}$ *and* $\mathcal{A}'$.

Proof idea

Every accepting run is rooted in a final state. Therefore they cannot use any state not in $F \cup \bigcup_{q_f \in F} \mathfrak{Frnd}(q_f)$. □

Definition (Support of a state)

Support of q: the set of all symbols of Σ in which a term which evaluates to q may be rooted.

$$\mathfrak{Sup}(q) \stackrel{\text{def}}{=} \{\, f \in \Sigma \mid \exists f(\dots) \to q \in \Delta \,\}.$$

Definition (Σ-spurious state)

A state $q \in Q$ is a Σ-*spurious state* if there exists $p, p' \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} p'$ and $\mathfrak{Sup}(p) \cap \mathfrak{Sup}(p') = \varnothing$.

Lemma (Removal of Σ-spurious states)

Let $\mathcal{A}$ be a TAGED, *$S \subseteq Q$ the set of all its Σ-spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{Lng}(\mathcal{A}) = \mathcal{Lng}(\mathcal{A}')$.*

Definition (Support of a state)

Support of q: the set of all symbols of Σ in which a term which evaluates to q may be rooted.

$$\mathfrak{Sup}(q) \stackrel{\text{def}}{=} \{\, f \in \Sigma \mid \exists f(\dots) \to q \in \Delta \,\}\,.$$

Definition (Σ-spurious state)

A state $q \in Q$ is a *Σ-spurious state* if there exists $p, p' \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} p'$ and $\mathfrak{Sup}(p) \cap \mathfrak{Sup}(p') = \varnothing$.

Lemma (Removal of Σ-spurious states)

Let $\mathcal{A}$ be a TAGED, $S \subseteq Q$ the set of all its Σ-spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{Lng}(\mathcal{A}) = \mathcal{Lng}(\mathcal{A}')$.

Definition (Σ-spurious state)

A state $q \in Q$ is a Σ-*spurious state* if there exists $p, p' \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} p'$ and $\mathfrak{Sup}(p) \cap \mathfrak{Sup}(p') = \varnothing$.

Lemma (Removal of Σ-spurious states)

Let $\mathcal{A}$ be a TAGED*, $S \subseteq Q$ the set of all its Σ-spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}')$.*

Proof idea

If q appears in an accepting run, then so must p and p'. But they cannot satisfy the equality (rooted in different symbols). Absurd. So q cannot appear in any accepting run. □

Definition (Σ-spurious state)

A state $q \in Q$ is a Σ-*spurious state* if there exists $p, p' \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} p'$ and $\mathfrak{Sup}(p) \cap \mathfrak{Sup}(p') = \varnothing$.

Lemma (Removal of Σ-spurious states)

Let $\mathcal{A}$ be a TAGED, $S \subseteq Q$ the set of all its Σ-spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}')$.

Proof idea

If q appears in an accepting run, then so must p and p'. But they cannot satisfy the equality (rooted in different symbols). Absurd. So q cannot appear in any accepting run. $\square$

Definition (Spurious states)

Let $\mathcal{A}$ be a TAGED. A state $q \in Q$ is said to be a *spurious state* if there exists $p \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} q$.

Lemma (Removal of spurious states)

Let $\mathcal{A}$ be a TAGED, $S \subseteq Q$ the set of all its spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}')$.

Proof idea

Suppose q appears in an accepting run at position β, then $\exists \alpha_p \lhd \beta$ *st.* $\rho(\alpha_p) = p$. A strict subterm and its parent are equal. Absurd. So q does not appear. $\square$

Definition (Spurious states)

Let $\mathcal{A}$ be a TAGED. A state $q \in Q$ is said to be a *spurious state* if there exists $p \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} q$.

Lemma (Removal of spurious states)

Let $\mathcal{A}$ be a TAGED, $S \subseteq Q$ the set of all its spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}')$.

Proof idea

Suppose q appears in an accepting run at position β, then $\exists \alpha_p \lhd \beta$ *st.* $\rho(\alpha_p) = p$. A strict subterm and its parent are equal. Absurd. So q does not appear. $\square$

Definition (Spurious states)

Let $\mathcal{A}$ be a TAGED. A state $q \in Q$ is said to be a *spurious state* if there exists $p \in \mathfrak{Need}(q)$ such that $p =_{\mathcal{A}} q$.

Lemma (Removal of spurious states)

Let $\mathcal{A}$ be a TAGED, $S \subseteq Q$ the set of all its spurious states, and $\mathcal{A}' = \mathfrak{Rst}(\mathcal{A}, Q \setminus S)$. Then $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{A}')$.

Proof idea

Suppose q appears in an accepting run at position β, then $\exists \alpha_p \lhd \beta$ *st.* $\rho(\alpha_p) = p$. A strict subterm and its parent are equal. Absurd. So q does not appear. $\square$

Cleanup: hunting for spuriousness

An example

```
TAGED 'example 1' [64] = {
  states = #7{q0, q1, q2, q3, q4, q5, q6}
  final = #1{q6}
  rules = #16{
a2()->q0, a2()->q2, a2()->q4, a3()->q3, a5()->q0, a5()->q2,
a5()->q4, f1(q5)->q5, f3(q1)->q5, g1(q1, q5)->q5, g3(q0, q0)->q5,
g3(q1, q5)->q5, g5(q1, q1)->q5, h2(q2, q3, q4)->q1,
h3(q0, q0, q1)->q6, h3(q2, q3, q4)->q1
  }
  ==rel = #3{(q0,q0), (q3,q4), (q4,q3)}
}
```

State q_1 is Σ-spurious, because it depends on q_3 and q_4 ($q_3, q_4 \in \mathfrak{Need}(q_1)$ and $\mathfrak{Sup}(q_3) \cap \mathfrak{Sup}(q_4) = \{ a_3 \} \cap \{ a_2, a_5 \} = \varnothing$). Furthermore $q_1 \in \mathfrak{Need}(q_6)$, so q_6 is unreachable, and $\mathfrak{Lng}(\mathcal{A}) = \varnothing$.

```
TAGED 'example 1' [64] = {
  states = #7{q0, q1, q2, q3, q4, q5, q6}
  final = #1{q6}
  rules = #16{
a2()->q0, a2()->q2, a2()->q4, a3()->q3, a5()->q0, a5()->q2,
a5()->q4, f1(q5)->q5, f3(q1)->q5, g1(q1, q5)->q5, g3(q0, q0)->q5,
g3(q1, q5)->q5, g5(q1, q1)->q5, h2(q2, q3, q4)->q1,
h3(q0, q0, q1)->q6, h3(q2, q3, q4)->q1
  }
  ==rel = #3{(q0,q0), (q3,q4), (q4,q3)}
}
```

State q_1 is Σ-spurious, because it depends on q_3 and q_4 ($q_3, q_4 \in \mathfrak{Need}(q_1)$ and $\mathfrak{Sup}(q_3) \cap \mathfrak{Sup}(q_4) = \{ a_3 \} \cap \{ a_2, a_5 \} = \varnothing$).
Furthermore $q_1 \in \mathfrak{Need}(q_6)$, so q_6 is unreachable, and $\mathcal{L}\mathrm{ng}(\mathcal{A}) = \varnothing$.

Signature-Quotienting

Taking advantage of similarities between rules: postponing choice of symbols

$$\mathfrak{Rul}(q_{\mathrm{char}}) = \left\{ \begin{array}{c} a \to q_{\mathrm{char}}, \dots, z \to q_{\mathrm{char}} \\ 0 \to q_{\mathrm{char}}, \dots, 9 \to q_{\mathrm{char}} \\ A \to q_{\mathrm{char}}, \dots, Z \to q_{\mathrm{char}} \end{array} \right\}$$

$$\text{“}\{\, a, \dots, z, 0, \dots, 9, A, \dots, Z \,\} \to q_{\mathrm{char}} \in \Delta\text{”}$$

Signature-Quotienting

Taking advantage of similarities between rules: postponing choice of symbols

$$\mathfrak{Rul}(q_{\mathsf{char}}) = \left\{ \begin{array}{c} a \to q_{\mathsf{char}}, \dots, z \to q_{\mathsf{char}} \\ 0 \to q_{\mathsf{char}}, \dots, 9 \to q_{\mathsf{char}} \\ A \to q_{\mathsf{char}}, \dots, Z \to q_{\mathsf{char}} \end{array} \right\}$$

$$\text{“}\{\, a, \dots, z, 0, \dots, 9, A, \dots, Z \,\} \to q_{\mathsf{char}} \in \Delta\text{”}$$

Signature-Quotienting

Taking advantage of similarities between rules: postponing choice of symbols

$$\mathfrak{Rul}(q_{\mathrm{char}}) = \left\{ \begin{array}{c} a \to q_{\mathrm{char}}, \dots, z \to q_{\mathrm{char}} \\ 0 \to q_{\mathrm{char}}, \dots, 9 \to q_{\mathrm{char}} \\ A \to q_{\mathrm{char}}, \dots, Z \to q_{\mathrm{char}} \end{array} \right\}$$

$$\text{“}\{\, a, \dots, z, 0, \dots, 9, A, \dots, Z \,\} \to q_{\mathrm{char}} \in \Delta\text{”}$$

Definition (Conservation of arity)

Let $\approxeq^s$ an equivalence relation over Σ. It is *arity-preserving* if

$$\forall f, g \in \Sigma : \; f \approxeq^s g \implies \mathit{arity}(f) = \mathit{arity}(g).$$

Definition (Signature-quotiented TAGED)

Let $\mathcal{A} = (\Sigma, Q, F, \Delta, =_\mathcal{A}, \neq_\mathcal{A})$ be a TAGED. Then its *signature-quotiented TAGED*, or *signature-TAGED* for short, is the TAGED $\mathcal{A}^s = (\Sigma^s, Q, F, \Delta^s, =_\mathcal{A}, \neq_\mathcal{A})$, where

$$\Sigma^s \stackrel{\text{def}}{=} \Sigma/_{\approxeq^s}$$

$$\Delta^s \stackrel{\text{def}}{=} \{\, [\sigma]\,(p_1, \dots, p_n) \to q \mid \sigma(p_1, \dots, p_n) \to q \in \Delta \,\}\,.$$

Theorem (Signature-TAGED as over-approximation)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s$ its signature-TAGED. Then $\mathcal{L}ng(\mathcal{A}^s) = \varnothing \implies \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Definition (Signature-identity relation)

We define the *signature-identity* relation (denoted $\equiv^s$), such that:

$$f \equiv^s g \iff \mathtt{sigs}(f) = \mathtt{sigs}(g),$$

where $\mathtt{sigs}(\sigma) \stackrel{\text{def}}{=} \{ (p_1, \dots, p_n, q) \mid \sigma(p_1, \dots, p_n) \to q \in \Delta \}$.

Theorem (Friendly quotient)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s_{\equiv^s}$ its signature-TAGED, using $\equiv^s$ instead of $\approxeq^s$. Then $\mathcal{L}ng(\mathcal{A}^s_{\equiv^s}) = \varnothing \iff \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Theorem (Signature-TAGED as over-approximation)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s$ its signature-TAGED. Then $\mathcal{L}ng(\mathcal{A}^s) = \varnothing \implies \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Definition (Signature-identity relation)

We define the *signature-identity* relation (denoted $\equiv^s$), such that:

$$f \equiv^s g \iff \mathtt{sigs}(f) = \mathtt{sigs}(g),$$

where $\mathtt{sigs}(\sigma) \stackrel{\text{def}}{=} \{\, (p_1, \dots, p_n, q) \mid \sigma(p_1, \dots, p_n) \to q \in \Delta \,\}$.

Theorem (Friendly quotient)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s_{\equiv^s}$ its signature-TAGED, using $\equiv^s$ instead of $\approxeq^s$. Then $\mathcal{L}ng(\mathcal{A}^s_{\equiv^s}) = \varnothing \iff \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Theorem (Signature-TAGED as over-approximation)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s$ its signature-TAGED. Then $\mathcal{L}ng(\mathcal{A}^s) = \varnothing \implies \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Definition (Signature-identity relation)

We define the *signature-identity* relation (denoted $\equiv^s$), such that:

$$f \equiv^s g \iff \mathtt{sigs}(f) = \mathtt{sigs}(g),$$

where $\mathtt{sigs}(\sigma) \stackrel{\text{def}}{=} \{\, (p_1, \dots, p_n, q) \mid \sigma(p_1, \dots, p_n) \to q \in \Delta \,\}$.

Theorem (Friendly quotient)

Let $\mathcal{A}$ be a positive TAGED and $\mathcal{A}^s_{\equiv^s}$ its signature-TAGED, using $\equiv^s$ instead of $\approxeq^s$. Then $\mathcal{L}ng(\mathcal{A}^s_{\equiv^s}) = \varnothing \iff \mathcal{L}ng(\mathcal{A}) = \varnothing$.

Signature-Quotiented TAGED

Example (with an approximation relation)

```
TAGED 'restricted' [58] = {
  states = #6{q0, q1, q2, q3, q4, q5}
  final = #2{q1, q5}
  rules = #16{
    a1()->q0, a1()->q3, a3()->q2, a3()->q4, a4()->q0,
    a4()->q4, a5()->q2, a5()->q3, f1(q1)->q5, f5(q1)->q5,
    g1(q1, q5)->q5, g3(q0, q5)->q5, g5(q0, q5)->q5,
    g5(q1, q5)->q5, h3(q2, q3, q4)->q1, h4(q2, q3, q4)->q1
}}
```

```
Classes = #{<g5 g3 g1>; <h4 h3>; <a5 a4 a3 a1>; <f5 f1>}#
TAGED 'sig-quotient' [34] = {
  states = #6{q0, q1, q2, q3, q4, q5}
  final = #2{q1, q5}
  rules = #8{
    a4()->q0, a4()->q2, a4()->q3, a4()->q4,
    f5(q1)->q5, g5(q0, q5)->q5, g5(q1, q5)->q5,
    h4(q2, q3, q4)->q1
}}
```

Signature-Quotiented TAGED
Example (with an approximation relation)

```
TAGED 'restricted' [58] = {
  states = #6{q0, q1, q2, q3, q4, q5}
  final = #2{q1, q5}
  rules = #16{
    a1()->q0, a1()->q3, a3()->q2, a3()->q4, a4()->q0,
    a4()->q4, a5()->q2, a5()->q3, f1(q1)->q5, f5(q1)->q5,
    g1(q1, q5)->q5, g3(q0, q5)->q5, g5(q0, q5)->q5,
    g5(q1, q5)->q5, h3(q2, q3, q4)->q1, h4(q2, q3, q4)->q1
}}
```

```
Classes = #{<g5 g3 g1>; <h4 h3>; <a5 a4 a3 a1>; <f5 f1>}#
TAGED 'sig-quotient' [34] = {
  states = #6{q0, q1, q2, q3, q4, q5}
  final = #2{q1, q5}
  rules = #8{
    a4()->q0, a4()->q2, a4()->q3, a4()->q4,
    f5(q1)->q5, g5(q0, q5)->q5, g5(q1, q5)->q5,
    h4(q2, q3, q4)->q1
}}
```

Parenting Relations

Building successful and easy runs for cheap

1. Emptiness is easy for diagonal positive TAGEDs
2. Partial adaptation to non-diagonal cases
3. Previous tactics useful for (sometimes) proving emptiness.
4. This one useful for (sometimes) proving non-emptiness.

Definition (Diagonal positive TAGED)

A positive TAGED is *diagonal* if

$$(=_{\mathcal{A}}) \subseteq \{\, (q, q) \mid q \in Q \,\}\,.$$

Theorem (Diagonal testing)

Let $\mathcal{A}$ be a diagonal positive TAGED. Then

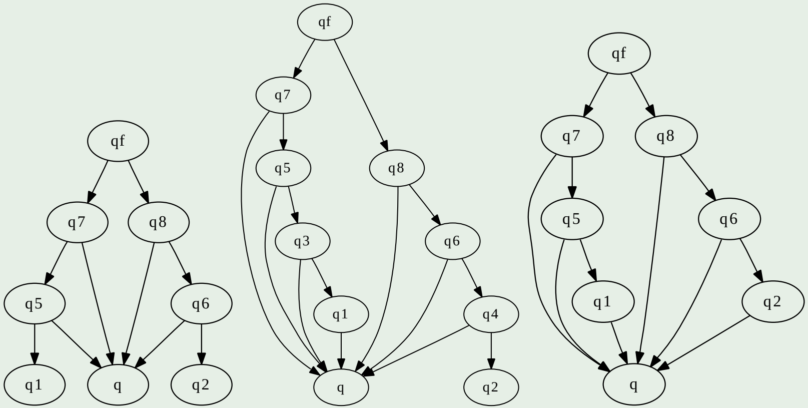
$$\mathcal{L}ng(\mathcal{A}) = \varnothing \iff \mathcal{L}ng(\mathfrak{ta}(\mathcal{A})) = \varnothing.$$

Proof idea

See beginning of proof of [Filiot et al., 2008, Theorem 1].

```
TAGED 'Heam' [146] = {
  alphab = #5{a/0, b/0, c/0, d/0, f/2}
  states = #10{q, q1, q2, q3, q4, q5, q6, q7, q8, qf}
  final = #1{qf}
  rules = #39{
    a()->q, a()->q1, a()->q2, b()->q, b()->q1, b()->q2,
    c()->q, c()->q1, c()->q2, d()->q, d()->q1, d()->q2,
    f(q, q)->q, f(q, q)->q1, f(q, q)->q2, f(q, q1)->q3,
    f(q, q1)->q5, f(q, q2)->q4, f(q, q2)->q6, f(q, q3)->q3,
    f(q, q3)->q5, f(q, q4)->q4, f(q, q4)->q6, f(q, q6)->q8,
    f(q, q7)->q7, f(q, qf)->qf, f(q1, q)->q3, f(q1, q)->q5,
    f(q2, q)->q4, f(q2, q)->q6, f(q3, q)->q3, f(q3, q)->q5,
    f(q4, q)->q4, f(q4, q)->q6, f(q5, q)->q7, f(q6, q)->q8,
    f(q7, q8)->qf, f(q8, q7)->qf, f(qf, q)->qf
  }
  ==rel = #2{(q1,q2), (q2,q1)}
}
```

Parenting relations
Introductory example



Parenting relations

Definition (Parenting relation)

Let $\mathcal{A}$ be a positive TAGED, and $q_f \in F$ one of its final states. Then a relation on Q $\prec$ is a *parenting relation of $\mathcal{A}$ (for q_f)* if it satisfies the four following properties:

- **(q_f-domination)**:
  The ordered set $(\operatorname{dom}(\prec), \prec^+)$ has a greatest element, which is q_f.
- **(Transitionality)**:
  $\forall q \in \operatorname{dom}(\prec) : \exists r \in \mathfrak{Rul}(q)$ *st.* $\mathfrak{Ant}(r) = \{ p \mid p \prec q \}$
- **(Strictness)**:
  $\prec^+$ is a strict partial order on its domain.
- **(Aspuriousness)**:
  There are no two states $p, q \in \operatorname{dom}(\prec)$ such that $p \prec^+ q$ and $p =_{\mathcal{A}} q$.

Definition (Parenting relation)

Let $\mathcal{A}$ be a positive TAGED, and $q_f \in F$ one of its final states. Then a relation on Q $\prec$ is a *parenting relation of* $\mathcal{A}$ *(for* q_f*)* if it satisfies the four following properties:

- **(q_f-domination)**:
  The ordered set $(\mathrm{dom}(\prec), \prec^+)$ has a greatest element, which is q_f.
- **(Transitionality)**:
  $\forall q \in \mathrm{dom}(\prec) : \exists r \in \mathfrak{Rul}(q)$ *st.* $\mathfrak{Ant}(r) = \{ p \mid p \prec q \}$
- **(Strictness)**:
  $\prec^+$ is a strict partial order on its domain.
- **(Aspuriousness)**:
  There are no two states $p, q \in \mathrm{dom}(\prec)$ such that $p \prec^+ q$ and $p =_{\mathcal{A}} q$.

Parenting relations

Definition (Parenting relation)

Let $\mathcal{A}$ be a positive TAGED, and $q_f \in F$ one of its final states. Then a relation on $Q \prec$ is a *parenting relation of $\mathcal{A}$ (for q_f)* if it satisfies the four following properties:

- **(q_f-domination)**:
  The ordered set $(\mathrm{dom}(\prec), \prec^+)$ has a greatest element, which is q_f.
- **(Transitionality)**:
  $\forall q \in \mathrm{dom}(\prec) : \exists r \in \mathfrak{Rul}(q)$ *st.* $\mathfrak{Ant}(r) = \{ p \mid p \prec q \}$
- **(Strictness)**:
  $\prec^+$ is a strict partial order on its domain.
- **(Aspuriousness)**:
  There are no two states $p, q \in \mathrm{dom}(\prec)$ such that $p \prec^+ q$ and $p \equiv_{\mathcal{A}} q$.

Definition (Parenting relation)

Let $\mathcal{A}$ be a positive TAGED, and $q_f \in F$ one of its final states. Then a relation on Q $\prec$ is a *parenting relation of $\mathcal{A}$ (for q_f)* if it satisfies the four following properties:

- **(q_f-domination)**:
  The ordered set $(\mathrm{dom}(\prec), \prec^+)$ has a greatest element, which is q_f.
- **(Transitionality)**:
  $\forall q \in \mathrm{dom}(\prec) : \exists r \in \mathfrak{Rul}(q)$ *st.* $\mathfrak{Ant}(r) = \{\, p \mid p \prec q \,\}$
- **(Strictness)**:
  $\prec^+$ is a strict partial order on its domain.
- **(Aspuriousness)**:
  There are no two states $p, q \in \mathrm{dom}(\prec)$ such that $p \prec^+ q$ and $p =_{\mathcal{A}} q$.

Definition (Restriction by states, projection)

Let $\mathcal{A} = (\Sigma, Q, F, \Delta, =_{\mathcal{A}}, \neq_{\mathcal{A}})$ be a TAGED, and let $S \subseteq Q$ be a set of states. We call *restriction of $\mathcal{A}$ to S* and denote $\mathfrak{Rst}(\mathcal{A}, S)$ the TAGED $(\Sigma, S, F \cap S, \Delta', =_{\mathcal{A}} \cap S^2, \neq_{\mathcal{A}} \cap S^2)$ where

$$\Delta' \stackrel{\text{def}}{=} \{ f(q_1, \dots, q_n) \to q \in \Delta \mid \{ q, q_1, \dots, q_n \} \subseteq S \} .$$

We also call *projection of $\mathcal{A}$ on S* the TAGED

$$\mathfrak{Prj}(\mathcal{A}, S) \stackrel{\text{def}}{=} (\Sigma, Q, S, \Delta, =_{\mathcal{A}}, \neq_{\mathcal{A}}).$$

Definition (Automaton under a state)

Let $\prec$ be a parenting relation of a TAGED $\mathcal{A}$, and $q \in \mathrm{dom}(\prec)$. We call *automaton under the state q* and denote $\mathfrak{Udr}(q, \prec)$, or simply $\mathfrak{Udr}(q)$, the automaton

$$\mathfrak{Udr}(q, \prec) = \mathfrak{Prj}\left(\mathfrak{Rst}\left(\mathcal{A}, \{ p \mid p \preccurlyeq^+ q \}\right), q\right).$$

Parenting relations

Definition (Restriction by states, projection)

Let $\mathcal{A} = (\Sigma, Q, F, \Delta, =_\mathcal{A}, \neq_\mathcal{A})$ be a TAGED, and let $S \subseteq Q$ be a set of states. We call *restriction of $\mathcal{A}$ to S* and denote $\mathfrak{Rst}\,(\mathcal{A}, S)$ the TAGED $(\Sigma, S, F \cap S, \Delta', =_\mathcal{A} \cap S^2, \neq_\mathcal{A} \cap S^2)$ where

$$\Delta' \stackrel{\text{def}}{=} \{\, f(q_1, \dots, q_n) \to q \in \Delta \mid \{\, q, q_1, \dots, q_n \,\} \subseteq S \,\}\,.$$

We also call *projection of $\mathcal{A}$ on S* the TAGED

$$\mathfrak{Prj}\,(\mathcal{A}, S) \stackrel{\text{def}}{=} (\Sigma, Q, S, \Delta, =_\mathcal{A}, \neq_\mathcal{A}).$$

Definition (Automaton under a state)

Let $\prec$ be a parenting relation of a TAGED $\mathcal{A}$, and $q \in \mathrm{dom}(\prec)$. We call *automaton under the state q* and denote $\mathfrak{Udr}(q, \prec)$, or simply $\mathfrak{Udr}(q)$, the automaton

$$\mathfrak{Udr}(q, \prec) = \mathfrak{Prj}\left(\mathfrak{Rst}\,(\mathcal{A}, \{\, p \mid p \preccurlyeq^+ q \,\})\,, q\right).$$

Definition (Fruitful parenting relation)

Let $\prec$ be a parenting relation of the positive TAGED $\mathcal{A}$, and $(\equiv_{\mathcal{A}}) \overset{\text{def}}{=} \left(=_{\mathcal{A}} \cap \operatorname{dom}(\prec)^2\right)^*$ We say that $\prec$ is *fruitful* if

$$\forall [q] \in \operatorname{dom}(\prec)/_{\equiv_{\mathcal{A}}}, \mathrm{Card}([q]) > 1 : \quad \bigcap_{q \in [q]} \mathcal{L}\mathrm{ng}(\mathfrak{Ud}\mathfrak{r}(q, \prec)) \neq \varnothing.$$

Theorem (Fruitful positive TAGEDs)

Let $\mathcal{A}$ be a positive TAGED. If there exists a fruitful parenting relation $\prec$ for one of its final states q_f, then it is non-empty.

Definition (Parenting core)

Let $\prec$ be a parenting relation of a TAGED $\mathcal{A}$. We call *core* of $\prec$ – and often denote $\lessdot^+$ – the relation

$$(\lessdot^+) \stackrel{\text{def}}{=} (\prec^+) \cap \operatorname{dom}(=_{\mathcal{A}})^2 .$$

Definition (Flat and pseudo-flat parenting relations)

Let $\prec$ be a parenting relation of a TAGED $\mathcal{A}$. It is called *flat* if its core $\lessdot^+$ is empty, and *pseudo-flat* if

$$\forall p, q, p' \in Q : \quad p \lessdot^+ q \implies (p =_{\mathcal{A}} p' \implies p = p') .$$

Theorem (flat and pseudo-flat tests)

Under the conditions and notations of theorem "fruitful `TAGED`s", let $[q]$ such that $\mathrm{Card}([q]) > 1$, and let

$$\mathcal{U} = \bigotimes_{q \in [q]} \mathfrak{Udr}(q, \prec).$$

Then the following statements hold:

1. *If $\prec$ is flat then $\mathcal{U}$ is a vanilla tree automaton or a diagonal positive `TAGED` with only one constraint, on its sole final state.*
2. *If $\prec$ is pseudo-flat then $\mathcal{U}$ is a diagonal `TAGED`.*

- Problem reduced to generation of parenting relations and emptiness of diagonal `TAGED`s
- Will not detect non-emptiness in *all* cases
- The more relations tested, the better

Parenting relations

Building terms is easy

Theorem (flat and pseudo-flat tests)

Under the conditions and notations of theorem "fruitful TAGEDs", let $[q]$ such that $\mathrm{Card}\,([q]) > 1$, and let

$$\mathcal{U} = \bigotimes_{q \in [q]} \mathfrak{Uor}(q, \prec).$$

Then the following statements hold:

1. *If $\prec$ is flat then $\mathcal{U}$ is a vanilla tree automaton or a diagonal positive TAGED with only one constraint, on its sole final state.*
2. *If $\prec$ is pseudo-flat then $\mathcal{U}$ is a diagonal TAGED.*

- Problem reduced to generation of parenting relations and emptiness of diagonal TAGEDs
- Will not detect non-emptiness in *all* cases
- The more relations tested, the better

Experiments: Generations 2 TA / 2 C

$\|Q\|$	Run	Something	Nothing	Failure
4.	26.8%	73.2%	0.0%	0.0%
7.	43.6%	55.6%	0.8%	0.0%
10.	48.8%	50.8%	0.4%	0.0%
13.	49.2%	50.8%	0.0%	0.0%
16.	50.0%	50.0%	0.0%	0.0%
19.	42.4%	57.6%	0.0%	0.0%
22.	41.2%	58.4%	0.4%	0.0%
25.	34.8%	65.2%	0.0%	0.0%
28.	30.4%	69.6%	0.0%	0.0%
31.	36.4%	63.6%	0.0%	0.0%
34.	38.8%	61.2%	0.0%	0.0%
37.	35.6%	64.4%	0.0%	0.0%
40.	28.0%	72.0%	0.0%	0.0%

Experiments: Generations 4 TA / 3 C

Height	Run	Something	Nothing	Failure	$\prec$ results
6	0.4%	69.6%	28.8%	1.2%	2.8%
9	0.4%	69.2%	25.6%	4.8%	6.4%
12	0.0%	55.6%	36.4%	8.0%	9.2%
15	0.0%	61.2%	26.4%	12.4%	7.6%
18	0.0%	53.2%	30.0%	16.8%	6.4%
21	0.0%	50.8%	30.0%	19.2%	8.8%
24	0.0%	46.8%	35.6%	17.6%	7.2%
27	0.0%	49.2%	28.8%	22.0%	8.8%
27	0.0%	45.6%	31.2%	23.2%	5.6%
30	0.0%	45.2%	31.2%	23.6%	6.8%
31	0.0%	50.8%	25.2%	24.0%	6.0%
34	0.0%	50.8%	26.8%	22.4%	6.4%
37	0.0%	43.6%	26.8%	29.6%	7.2%

Conclusion

Main points of the internship

1. Reduction and quick decisions:
    1. Cleanup
    2. Signature-quotienting
    3. Parenting relations
2. Brutal algorithm
3. Random generation of tree automata and `TAGED`s
4. `OCaml` implementation of the above *($\geqslant$ 2000 LOC)*.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (2007).
Tree Automata Techniques and Applications.
release October, 12th 2007.

Filiot, E., Talbot, J.-M., and Tison, S. (2008).
Tree Automata with Global Constraints.
In *12th International Conference on Developments in Language Theory (DLT)*, pages 314–326, Kyoto Japon.

Tabakov, D. and Vardi, M. (2005).
Experimental evaluation of classical automata constructions.
In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 396–411. Springer.