Automata on Infinite Trees with Equality and Disequality Constraints Between Siblings

Arnaud Carayol [1] Christof Löding [2] Olivier Serre [3]

[1] LIGM (Université Paris Est & CNRS)

[2] RWTH Aachen

[3] IRIF (Université Paris Diderot – Paris 7 & CNRS)

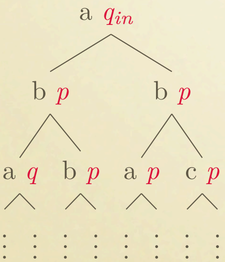
Supported by the European Community Research Training Network
"Games and Automata for Synthesis and Validation" (2002–2006)

Non-deterministic Parity Tree Automata

Non-deterministic parity tree automata :
$\mathcal{A} = \langle Q, A, \Delta, q_{in}, \mathrm{Col}\rangle$

- Q: control states
- A: labels alphabet
- $\Delta \subseteq Q \times A \times Q \times Q$: transition relation
- q_{in}: initial state
- $\mathrm{Col} : Q \to \mathbb{N}$: colouring function

Run on an A-labeled (infinite binary) tree t: Q-labelling of t consistent with Δ



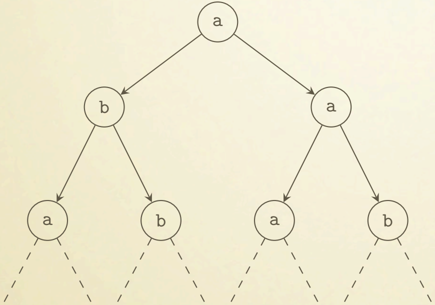
$\Delta = \{\cdots (q_{in}, a, p, p)$
$(p, b, q, p)(p, b, p, p) \cdots \}$

A branch is **accepting** iff the smallest colour infinitely often visited is even
A run is **accepting** iff all its branches are accepting
A tree is **accepted** iff there is an accepting run over it.

Tree Automata: Example

$A = \{a, b\}$

t:



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

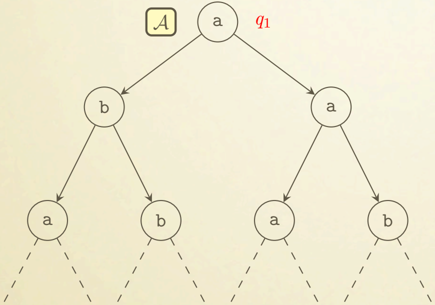
$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$

t:



$Q = \{q_1, q_2, q_3\}$ $\qquad \mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $\qquad q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $\qquad q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $\qquad q_3 \xrightarrow{b} (q_3, q_3)$

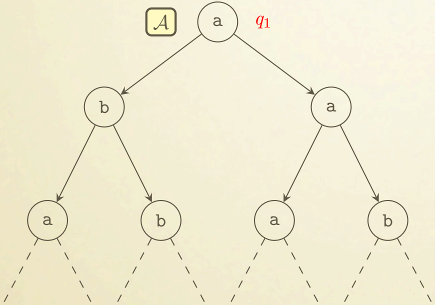
$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $\qquad F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$

t:



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ • $q_2 \xrightarrow{b} (q_1, q_3)$

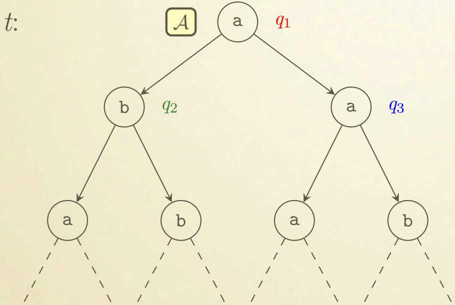
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

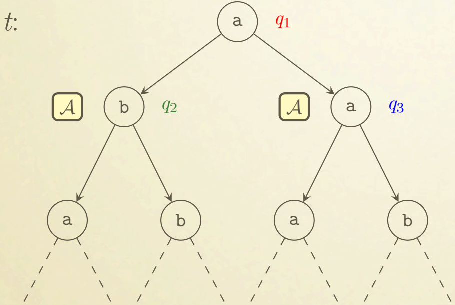
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

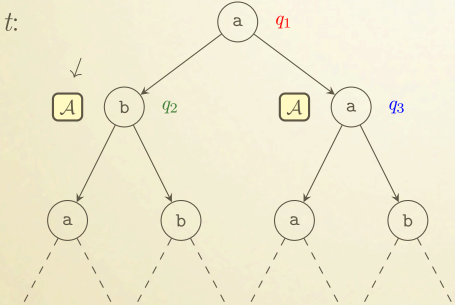
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

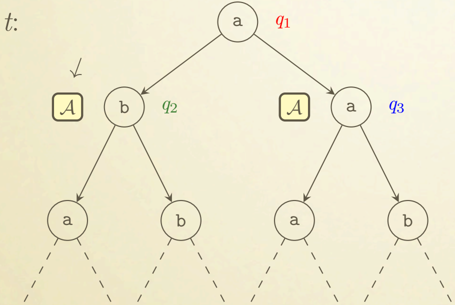
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$ •

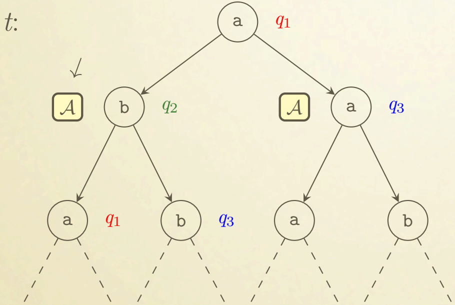
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

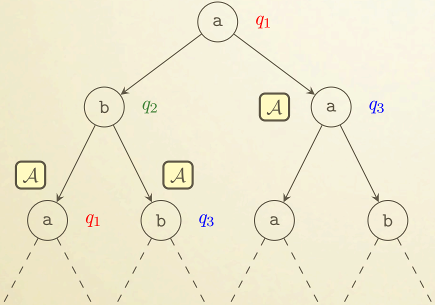
$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$

t:



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

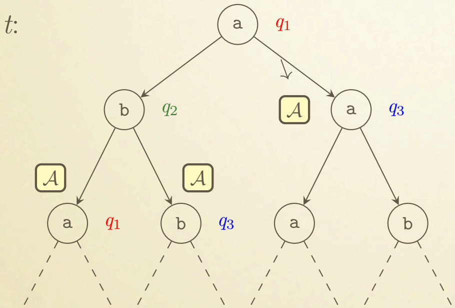
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

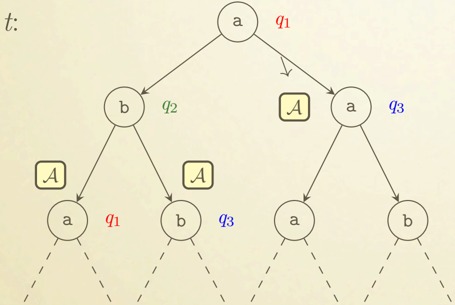
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

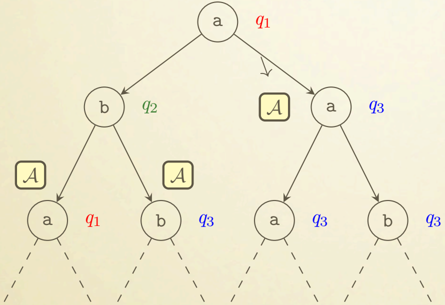
$q_3 \xrightarrow{a} (q_3, q_3)$ •

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$$A = \{a, b\}$$

t:



$\mathcal{A}$

$Q = \{q_1, q_2, q_3\}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $\qquad$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $\qquad$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $\qquad$ $q_3 \xrightarrow{b} (q_3, q_3)$

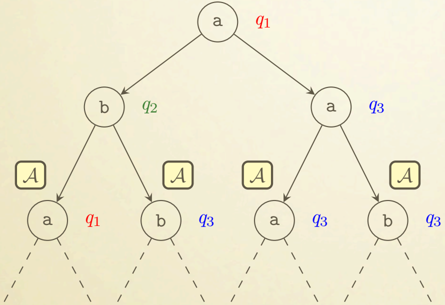
$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $\qquad$ $F = \{q_2, q_3\}$

Tree Automata: Example

$$A = \{a, b\}$$

t:



$Q = \{q_1, q_2, q_3\}$ $\qquad$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $\qquad$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $\qquad$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $\qquad$ $q_3 \xrightarrow{b} (q_3, q_3)$

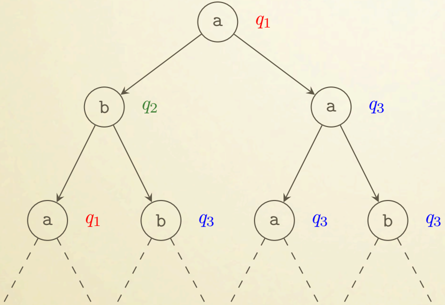
$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $\qquad$ $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$

t:



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

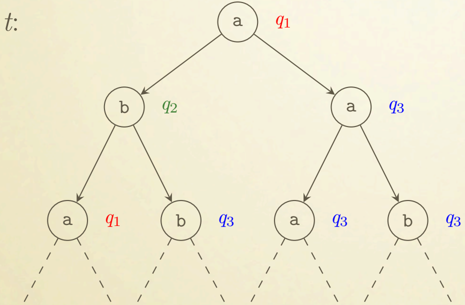
$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

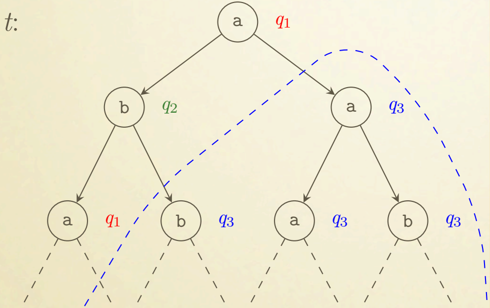
A branch is accepting if it has infinitely many occurrences of a state from F (Büchi).

A run is accepting if *all its branches* are accepting (∀).

A tree is accepted if *there exists* an accepting run (∃).

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

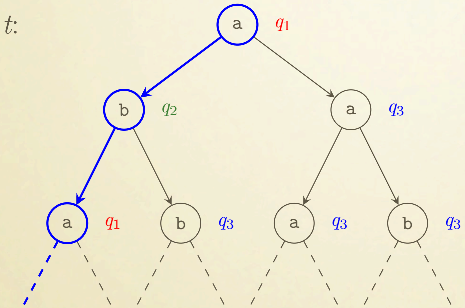
A branch is accepting if it has infinitely many occurrences of a state from F (Büchi).

A run is accepting if *all its branches* are accepting (∀).

A tree is accepted if *there exists* an accepting run (∃).

Tree Automata: Example

$A = \{a, b\}$



$Q = \{q_1, q_2, q_3\}$ $\mathcal{A}$

$q_1 \xrightarrow{\mathcal{A}} (q_1, q_3)$ $q_2 \xrightarrow{\mathcal{A}} (q_2, q_3)$

$q_1 \xrightarrow{\mathcal{A}} (q_2, q_3)$ $q_2 \xrightarrow{b} (q_1, q_3)$

$q_1 \xrightarrow{b} (q_1, q_3)$ $q_3 \xrightarrow{b} (q_3, q_3)$

$q_3 \xrightarrow{a} (q_3, q_3)$

initial state : q_1 $F = \{q_2, q_3\}$

A branch is accepting if it has infinitely many occurrences of a state from F (Büchi).

A run is accepting if *all its branches* are accepting ($\forall$).

A tree is accepted if *there exists* an accepting run ($\exists$).

Regular Tree Languages

A subset L of trees is **regular** if there exists some non-deterministic parity tree automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

Regular trees languages have many nice properties, among other:

- Coincide with MSO definable languages (hence, expressive).
- Form an effective Boolean algebra.
- Decidable emptiness and cardinality problem.

Whether the class can be extended while preserving (most of) its good properties is a challenging problem.

Regular Tree Languages

A subset L of trees is **regular** if there exists some non-deterministic parity tree automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

Regular trees languages have many nice properties, among other:

- Coincide with MSO definable languages (hence, expressive).
- Form an effective Boolean algebra.
- Decidable emptiness and cardinality problem.

Whether the class can be extended while preserving (most of) its good properties is a challenging problem.

We address this question by considering automata that can check equality between siblings

Tree Automata With Constraints

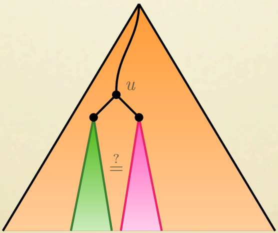
Main idea: works as usual tree automata except that transitions can be guarded by an equality/disequality requirement on siblings.

Tree Automata With Constraints

Main idea: works as usual tree automata except that transitions can be guarded by an equality/disequality requirement on siblings.

Formally: With any A-labelled tree t associate an $A \times \{=, \neq\}$ tree $t^{\stackrel{?}{=}}$ by annotating every node u in t by an extra information regarding on whether the left and the right subtrees rooted at u are equal or not. More formally, for every $u \in \{0,1\}^*$,

$$t^{\stackrel{?}{=}}(u) = \begin{cases} (t(u), =) & \text{if } t[u0] = t[u1] \\ (t(u), \neq) & \text{if } t[u0] \neq t[u1] \end{cases}$$



Tree Automata With Constraints

Main idea: works as usual tree automata except that transitions can be guarded by an equality/disequality requirement on siblings.

Formally: With any A-labelled tree t associate an $A \times \{=, \neq\}$ tree $t^{\stackrel{?}{=}}$ by annotating every node u in t by an extra information regarding on whether the left and the right subtrees rooted at u are equal or not. More formally, for every $u \in \{0,1\}^*$,

$$t^{\stackrel{?}{=}}(u) = \begin{cases} (t(u), =) & \text{if } t[u0] = t[u1] \\ (t(u), \neq) & \text{if } t[u0] \neq t[u1] \end{cases}$$

An automaton $\mathcal{A}$ with constraints over alphabet A is an automaton over alphabet $A \times \{=, \neq\}$ and one lets

$$L^{con}(\mathcal{A}) = \{t \mid t^{\stackrel{?}{=}} \in L(\mathcal{A})\}$$

Properties of Languages Accepted by Automata with Constraints

$\mathbf{REG}^{\stackrel{?}{=}}$: class of languages recognised by automata with constraints.

Theorem. The class $\mathbf{REG}^{\stackrel{?}{=}}$ is an effective Boolean algebra.

Conjecture. The class $\mathbf{REG}^{\stackrel{?}{=}}$ is not closed under projection.

Properties of Languages Accepted by Automata with Constraints

$\mathbf{REG}^{\stackrel{?}{=}}$: class of languages recognised by automata with constraints.

Theorem. The class $\mathbf{REG}^{\stackrel{?}{=}}$ is an effective Boolean algebra.

Conjecture. The class $\mathbf{REG}^{\stackrel{?}{=}}$ is not closed under projection.

It captures natural properties beyond MSO like "the tree satisfies φ and $\forall x$ if the subtree at x satisfies φ_1, then its two subtrees are different/equal, and satisfy φ_2".

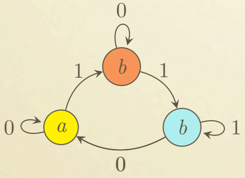
Properties of Languages Accepted by Automata with Constraints

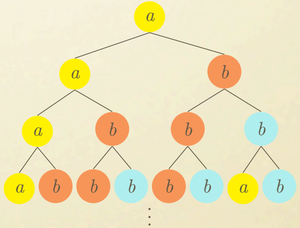
$\mathbf{REG}^{\overset{?}{=}}$: class of languages recognised by automata with constraints.

Theorem. The class $\mathbf{REG}^{\overset{?}{=}}$ is an effective Boolean algebra.

Conjecture. The class $\mathbf{REG}^{\overset{?}{=}}$ is not closed under projection.

It captures natural properties beyond MSO like "the tree satisfies φ and $\forall x$ if the subtree at x satisfies φ_1, then its two subtrees are different/equal, and satisfy φ_2".





Proposition. Let $\mathcal{A}$ be an automaton with constraints and let t be a regular tree. Then one can decide whether $t \in L^{con}(\mathcal{A})$.

The Cardinality Problem

The **cardinality profile** $\kappa_{\mathcal{A}}$ of $\mathcal{A}$, is a mapping that assigns to each state q of $\mathcal{A}$ the cardinality of $L^{con}(\mathcal{A}_q)$.

Proposition. Let $\aleph_0$ be the cardinality of the set of natural numbers, and $2^{\aleph_0}$ the cardinality of the set of the real numbers. Then

$$\kappa_{\mathcal{A}} : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$$

The Cardinality Problem

The **cardinality profile** $\kappa_{\mathcal{A}}$ of $\mathcal{A}$, is a mapping that assigns to each state q of $\mathcal{A}$ the cardinality of $L^{con}(\mathcal{A}_q)$.

Proposition. Let $\aleph_0$ be the cardinality of the set of natural numbers, and $2^{\aleph_0}$ the cardinality of the set of the real numbers. Then

$$\kappa_{\mathcal{A}} : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$$

For **regular tree languages** it is known from [Niwinski'91] that one can compute the cardinality profile.

The Cardinality Problem

The **cardinality profile** $\kappa_{\mathcal{A}}$ of $\mathcal{A}$, is a mapping that assigns to each state q of $\mathcal{A}$ the cardinality of $L^{con}(\mathcal{A}_q)$.

Proposition. Let $\aleph_0$ be the cardinality of the set of natural numbers, and $2^{\aleph_0}$ the cardinality of the set of the real numbers. Then

$$\kappa_{\mathcal{A}} : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$$

For **regular tree languages** it is known from [Niwinski'91] that one can compute the cardinality profile.

Our main result is the following:

Theorem. Let $\mathcal{A}$ be a parity tree automaton with constraints. Then, one can compute its cardinality profile.

Some Tools

First, get rids of equalities:

Theorem. Let $\mathcal{A}$ be an automaton with equality and disequality constraints. Then one can build an automaton $\mathcal{B}$ with **disequality everywhere** and s.t. $L^{con}(\mathcal{A})$ and $L^{con}(\mathcal{B})$ have the same cardinality.

Some Tools

First, get rids of equalities:

Theorem. Let $\mathcal{A}$ be an automaton with equality and disequality constraints. Then one can build an automaton $\mathcal{B}$ with **disequality everywhere** and s.t. $L^{con}(\mathcal{A})$ and $L^{con}(\mathcal{B})$ have the same cardinality.

Second, over-approximate the language $L^{con}(\mathcal{A}_q)$ by $L(\widehat{\mathcal{A}})$ the language accepted by forgetting the constraints and use the results of [Niwinski'91] on it.

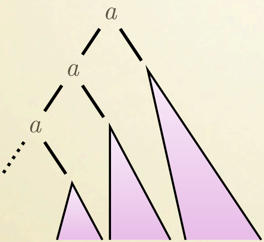
Example

Let t_a/t_b be defined by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

EXAMPLE

Let t_a/t_b be defined by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

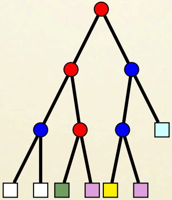
Let $\mathcal{A}$ be the safety automaton $(\{q_{in}, q_b\}, \{(a,\neq),(b,\neq)\}, q_{in}, \Delta_{\mathcal{A}})$ where $\Delta = \{(q_{in},(a,\neq),q_{in},t_b),(q_{in},(a,\neq),q_b,t_b),(q_b,(b,\neq),t_a,t_b)\}$.



Then, $|L(\widehat{\mathcal{A}})| = \aleph_0$. But, $L^{con}(\mathcal{A}) = \{t_a\}$.

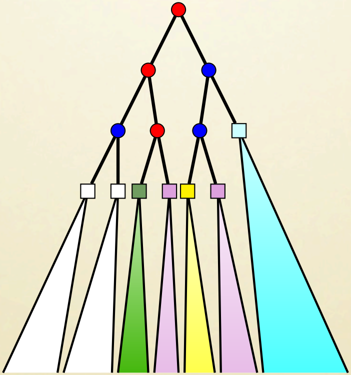
Countable Unconstrained Languages (1/2)

If $L(\widehat{\mathcal{A}})$ is countable, then it has a special shape. Namely there is a regular language of **finite** trees $L(\mathcal{B})$ such that the trees in $L(\widehat{\mathcal{A}})$ are exactly those obtained from a tree in $L(\mathcal{B})$ by replacing every leaf by a regular tree uniquely determined by the leaf label.



Countable Unconstrained Languages (1/2)

If $L(\widehat{\mathcal{A}})$ is countable, then it has a special shape. Namely there is a regular language of **finite** trees $L(\mathcal{B})$ such that the trees in $L(\widehat{\mathcal{A}})$ are exactly those obtained from a tree in $L(\mathcal{B})$ by replacing every leaf by a regular tree uniquely determined by the leaf label.

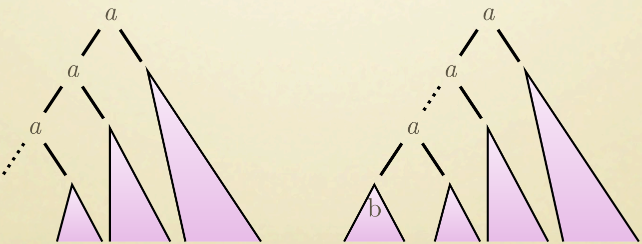


Countable Unconstrained Languages: Example

If $L(\widehat{\mathcal{A}})$ is countable, then it has a special shape. Namely there is a regular language of **finite** trees $L(\mathcal{B})$ such that the trees in $L(\widehat{\mathcal{A}})$ are exactly those obtained from a tree in $L(\mathcal{B})$ by replacing every leaf by a regular tree uniquely determined by the leaf label.

Let t_a/t_b be defined by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

Let $\mathcal{A}$ be the safety automaton $(\{q_{in}, q_b\}, \{(a, \neq), (b, \neq)\}, q_{in}, \Delta_{\mathcal{A}})$ where $\Delta = \{(q_{in}, (a, \neq), q_{in}, t_b), (q_{in}, (a, \neq), q_b, t_b), (q_b, (b, \neq), t_a, t_b)\}$.



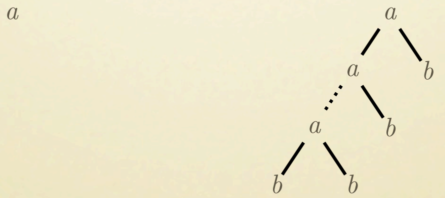
Countable Unconstrained Languages: Example

If $L(\widehat{\mathcal{A}})$ is countable, then it has a special shape. Namely there is a regular language of **finite** trees $L(\mathcal{B})$ such that the trees in $L(\widehat{\mathcal{A}})$ are exactly those obtained from a tree in $L(\mathcal{B})$ by replacing every leaf by a regular tree uniquely determined by the leaf label.

Let t_a/t_b be defined by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

Let $\mathcal{A}$ be the safety automaton $(\{q_{in}, q_b\}, \{(a, \neq), (b, \neq)\}, q_{in}, \Delta_{\mathcal{A}})$ where $\Delta = \{(q_{in}, (a, \neq), q_{in}, t_b), (q_{in}, (a, \neq), q_b, t_b), (q_b, (b, \neq), t_a, t_b)\}$.

Then $L(\mathcal{B})$ is (where $a \mapsto t_a$ and $b \mapsto t_b$):



Countable Unconstrained Languages (2/2)

If $L(\widehat{\mathcal{A}})$ is countable, then it has a special shape. Namely there is a regular language of **finite** trees $L(\mathcal{B})$ such that the trees in $L(\widehat{\mathcal{A}})$ are exactly those obtained from a tree in $L(\mathcal{B})$ by replacing every leaf by a regular tree uniquely determined by the leaf label.

Roadmap to compute the cardinality of $L(\mathcal{A})$ when $L(\widehat{\mathcal{A}})$ is countable:

- Safely assume that $\mathcal{A}$ has disequality everywhere.
- Built from $\mathcal{B}$ an automaton on finite trees with constraints $\mathcal{C}$ such that $L^{con}(\mathcal{A})$ and $L^{con}(\mathcal{C})$ have the same cardinal.
- Use the results from [Bogaert&Tison'02] to compute the cardinal of $L^{con}(\mathcal{C})$.

Algorithm to Compute the Cardinality Profile

Input: Tree automaton with disequality constraints everywhere $\mathcal{A}$
Data Structure:

Set $S \leftarrow Q$ the states of $\mathcal{A}$
Automaton $\mathcal{B} \leftarrow \mathcal{A}$
Function $\kappa : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$; $\kappa(q) \leftarrow 2^{\aleph_0}$ for all q

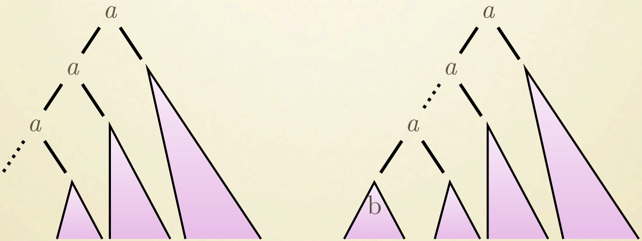
Code:

while $\exists q \in S$ s.t. $|L(\widehat{\mathcal{B}_q})| \leq \aleph_0$ **do**
  $\kappa(q) \leftarrow |L^{con}(\mathcal{B}_q)|$
  if $\kappa(q) = 0$ **then**
    $\mathcal{B} \leftarrow \mathcal{B}_{q \mapsto \emptyset}$
  else if $\kappa(q) < \aleph_0$ **then**
    Let $L^{con}(\mathcal{B}_q) = \{t_1, \ldots, t_n\}$
    $\mathcal{B} \leftarrow \mathcal{B}_{q \mapsto t_1, \ldots, t_n}$
  end if
  $S \leftarrow S \setminus \{q\}$
end while
return κ

Example of Execution

Recall that we defined t_a/t_b by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

And $\mathcal{A}$ as the safety automaton $(\{q_{in}, q_b\}, \{(a, \neq), (b, \neq)\}, q_{in}, \Delta_{\mathcal{A}})$ where $\Delta = \{(q_{in}, (a, \neq), q_{in}, t_b), (q_{in}, (a, \neq), q_b, t_b), (q_b, (b, \neq), t_a, t_b)\}$.



$|L(\widehat{\mathcal{A}})| = \aleph_0$ but $L^{con}(\mathcal{A}) = \{t_a\}$.

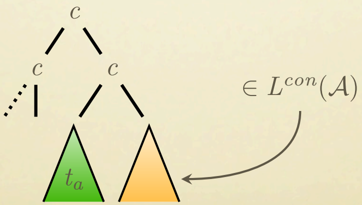
Example of Execution

Recall that we defined t_a/t_b by $t_a(\varepsilon) = a$, $t_b(\varepsilon) = b$, $t_a(u0) = t_b(u0) = a$ and $t_a(u1) = t_b(u1) = b$ for any $u \in \{0,1\}^*$.

And $\mathcal{A}$ as the safety automaton $(\{q_{in}, q_b\}, \{(a, \neq), (b, \neq)\}, q_{in}, \Delta_{\mathcal{A}})$ where $\Delta = \{(q_{in}, (a, \neq), q_{in}, t_b), (q_{in}, (a, \neq), q_b, t_b), (q_b, (b, \neq), t_a, t_b)\}$. $|L(\widehat{\mathcal{A}})| = \aleph_0$ but $L^{con}(\mathcal{A}) = \{t_a\}$.

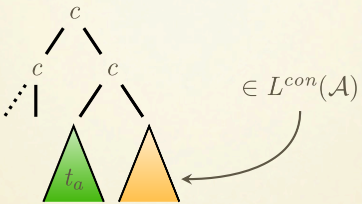
Consider $\mathcal{B}$ that (note that $|L(\widehat{\mathcal{B}})| = 2^{\aleph_0}$):

- Checks that the leftmost branch is labelled only by c's.
- Checks that any right subtree of a node on that branch is such that the root is labelled by c, the left subtree is t_a while the right subtree is accepted by the automaton $.\mathcal{A}$



Example of Execution

$|L(\widehat{\mathcal{A}})| = \aleph_0$ but $L^{con}(\mathcal{A}) = \{t_a\}$.



$\mathcal{B} = (Q_\mathcal{B}, \{(a, \neq), (b, \neq), (c, \neq)\}, q_c, \Delta_\mathcal{B}, \mathrm{Col})$ with $Q_\mathcal{B} = Q_\mathcal{A} \cup \{q_c, q'_c\}$ and $\Delta_\mathcal{B} = \Delta_\mathcal{A} \cup \{(q_c, (c, \neq), q_c, q'_c), (q'_c, (c, \neq), t_a, q_{in})\}$.

Previous Algorithm:

- First detects that $|L(\widehat{\mathcal{B}_{q_{in}}})| \leq \aleph_0$, computes $\kappa(q_{in}) = 1$ and change $\mathcal{B}$ to $\mathcal{B}_{q_{in} \mapsto t_a}$.
- Then detects that $|L(\widehat{\mathcal{B}_{q'_c}})| \leq \aleph_0$, computes $\kappa(q'_c) = 0$ and change $\mathcal{B}$ to $\mathcal{B}_{q'_c \mapsto \emptyset}$.
- Finally detects that $\kappa(q_c) = 0$.

Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

- At any stage the language (with contraints) is unchanged.
- ➔ Countable values are correct.

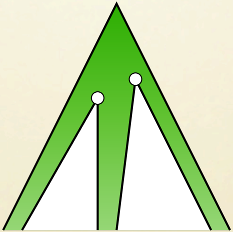
Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

➔ Countable values are correct.

- Define run-tree with holes as pieces of runs where:
  - Holes correspond to states where κ equals $2^{\aleph_0}$.
  - Parts without holes are accepting and satisfies the constraints.



- Prove that for every state q with $\kappa(q) = 2^{\aleph_0}$ and every $N \geq 0$ there are N q-run-tree with holes that are **pairwise different**.

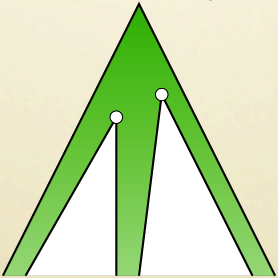
Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

➔ Countable values are correct.

- Define run-tree with holes as pieces of runs where:
    - Holes correspond to states where κ equals $2^{\aleph_0}$.
    - Parts without holes are accepting and satisfies the constraints.
- Prove that for every state q with $\kappa(q) = 2^{\aleph_0}$ and every $N \geq 0$ there are N q-run-tree with holes that are **pairwise different**.
- Combine them to obtain uncountably many accepted trees.



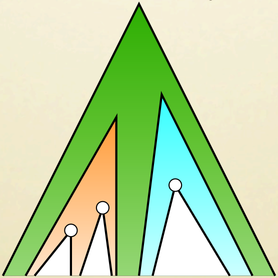
Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

➔ Countable values are correct.

- Define run-tree with holes as pieces of runs where:
  - Holes correspond to states where κ equals $2^{\aleph_0}$.
  - Parts without holes are accepting and satisfies the constraints.
- Prove that for every state q with $\kappa(q) = 2^{\aleph_0}$ and every $N \geq 0$ there are N q-run-tree with holes that are **pairwise different**.
- Combine them to obtain uncountably many accepted trees.



Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

- ➔ Countable values are correct.
- Define run-tree with holes as pieces of runs where:
  - Holes correspond to states where κ equals $2^{\aleph_0}$.
  - Parts without holes are accepting and satisfies the constraints.
- Prove that for every state q with $\kappa(q) = 2^{\aleph_0}$ and every $N \geq 0$ there are N q-run-tree with holes that are **pairwise different**.
- Combine them to obtain uncountably many accepted trees.

Büchi Case

Theorem. The algorithm returns the correct cardinality profile.

Proof ingredients.

➔ Countable values are correct.

- Define run-tree with holes as pieces of runs where:
  - Holes correspond to states where κ equals $2^{\aleph_0}$.
  - Parts without holes are accepting and satisfies the constraints.
- Prove that for every state q with $\kappa(q) = 2^{\aleph_0}$ and every $N \geq 0$ there are N q-run-tree with holes that are **pairwise different**.
- Combine them to obtain uncountably many accepted trees.
- For Büchi condition do the same but consider only run-tree with holes s.t. a final state occurs in any path from the root to a hole.

Does it Also Work for co-Büchi?

No :-(as there exists a co-Büchi automaton $\mathcal{A}$ s.t. $|L(\mathcal{A}_q)| = 2^{\aleph_0}$ for all q while $L^{con}(\mathcal{A}_q) = \emptyset$.

Does it Also Work for co-Büchi?

No :-(as there exists a co-Büchi automaton $\mathcal{A}$ s.t. $|L(\mathcal{A}_q)| = 2^{\aleph_0}$ for all q while $L^{con}(\mathcal{A}_q) = \emptyset$.

Define $\mathcal{A} = (\{q_a, q_b\}, \{a, b\}, q_a, \Delta, \mathrm{Col})$ where $\mathrm{Col}(q_a) = 2$ and $\mathrm{Col}(q_b) = 1$, and Δ consists of those transitions $(q_x, (x, \neq), q_0, q_1)$ where $x \in \{a, b\}$ and q_0, q_1 are any states.

- There is at most one possible run per tree: the one that assigns q_x to each node labelled by $(x, \neq)$.
- The unconstrained language from state q_x is the set of all trees such that the root is labelled by x and such that any branch contains finitely many b's ➔ Uncountable.

Does it Also Work for co-Büchi?

No :-(as there exists a co-Büchi automaton $\mathcal{A}$ s.t. $|L(\mathcal{A}_q)| = 2^{\aleph_0}$ for all q while $L^{con}(\mathcal{A}_q) = \emptyset$.

Define $\mathcal{A} = (\{q_a, q_b\}, \{a, b\}, q_a, \Delta, \mathrm{Col})$ where $\mathrm{Col}(q_a) = 2$ and $\mathrm{Col}(q_b) = 1$, and Δ consists of those transitions $(q_x, (x, \neq), q_0, q_1)$ where $x \in \{a, b\}$ and q_0, q_1 are any states.

- There is at most one possible run per tree: the one that assigns q_x to each node labelled by $(x, \neq)$.
- The unconstrained language from state q_x is the set of all trees such that the root is labelled by x and such that any branch contains finitely many b's ➔ Uncountable.
- But $L^{con}(\mathcal{A}_{q_x}) = \emptyset$ for $x \in \{a, b\}$. Indeed:
  - An accepted tree would contain at least one node u_1 labelled by b (to satisfy $\neq$ at the root).
  - Same for the subtree rooted at u_1, and so on...
  - Hence there is $u_1 \sqsubset u_2 \sqsubset u_3 \cdots$ all labelled by b, leading to violate co-Büchi condition.

How to Handle the co-Büchi Case? (1/2)

Trace: pair $\rho = (t_\rho, r_\rho)$ where t_ρ is an infinite **valid** tree and r_ρ is a run of $\mathcal{A}$ on $t_\rho^{\stackrel{?}{=}}$. starting from some arbitrary state. The trace is accepting if the run is.

How to Handle the co-Büchi Case? (1/2)

Trace: pair $\rho = (t_\rho, r_\rho)$ where t_ρ is an infinite **valid** tree and r_ρ is a run of $\mathcal{A}$ on $t_\rho^{\overset{?}{=}}$. starting from some arbitrary state. The trace is accepting if the run is.

We define two (monotone) operations on sets of traces,

$$\mathrm{Attr}(X) = \{(t_\rho, r_\rho) \mid \forall \text{ infinite branch } \pi,\ \exists u \sqsubset \pi \text{ s.t. } (t_\rho[u], r_\rho[u]) \in X\}$$

$$\begin{aligned}\mathrm{Safety}(X) = \{&(t_\rho, r_\rho) \mid \forall \text{ infinite branch } \pi, \text{either } \forall u \sqsubset \pi,\ \mathrm{Col}(r_\rho(u)) = 2,\\ &\text{or } \exists u \sqsubset \pi \text{ s.t. } (t_\rho[u], r_\rho[u]) \in X \text{ and } \mathrm{Col}(r_\rho(v)) = 2\ \forall v \sqsubset u\}\end{aligned}$$

and an increasing transfinite sequence $(X_\alpha)_\alpha$

$$\begin{cases} X_0 = \emptyset \\ X_{\alpha+1} = \mathrm{Attr}(\mathrm{Safety}(X_\alpha)) \\ X_\alpha = \bigcup_{\beta<\alpha} X_\beta & \text{for } \alpha \text{ limit ordinal} \end{cases}$$

How to Handle the co-Büchi Case? (1/2)

Trace: pair $\rho = (t_\rho, r_\rho)$ where t_ρ is an infinite **valid** tree and r_ρ is a run of $\mathcal{A}$ on $t_\rho^{\stackrel{?}{=}}$. starting from some arbitrary state. The trace is accepting if the run is.

We define two (monotone) operations on sets of traces,

$$\mathrm{Attr}(X) = \{(t_\rho, r_\rho) \mid \forall \text{ infinite branch } \pi, \exists u \sqsubset \pi \text{ s.t. } (t_\rho[u], r_\rho[u]) \in X\}$$

$$\begin{aligned}\mathrm{Safety}(X) = &\{(t_\rho, r_\rho) \mid \forall \text{ infinite branch } \pi, \text{either } \forall u \sqsubset \pi,\ \mathrm{Col}(r_\rho(u)) = 2, \\ &\text{or } \exists u \sqsubset \pi \text{ s.t. } (t_\rho[u], r_\rho[u]) \in X \text{ and } \mathrm{Col}(r_\rho(v)) = 2\ \forall v \sqsubset u\}\end{aligned}$$

and an increasing transfinite sequence $(X_\alpha)_\alpha$

$$\begin{cases} X_0 = \emptyset \\ X_{\alpha+1} = \mathrm{Attr}(\mathrm{Safety}(X_\alpha)) \\ X_\alpha = \bigcup_{\beta<\alpha} X_\beta & \text{for } \alpha \text{ limit ordinal} \end{cases}$$

Lemma. The limit of $(X_\alpha)_\alpha$ is the set of accepting traces.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_{\mathcal{A}}$ of $\mathcal{A}$

$$\mathbf{p}_{\mathcal{A}}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_{\mathcal{A}}$.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_\mathcal{A}$ of $\mathcal{A}$

$$\mathbf{p}_\mathcal{A}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_\mathcal{A}$.

A **profile** is some $\mathbf{p} : Q \to 2^{RegTrees} \cup \{\infty\}$ that is smaller than $\mathbf{p}_\mathcal{A}$.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_{\mathcal{A}}$ of $\mathcal{A}$

$$\mathbf{p}_{\mathcal{A}}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_{\mathcal{A}}$.

A **profile** is some $\mathbf{p} : Q \to 2^{RegTrees} \cup \{\infty\}$ that is smaller than $\mathbf{p}_{\mathcal{A}}$.

Define a profile counterpart of operators Attr and Safety and show that one can compute Attr($\mathbf{p}$) and Safety($\mathbf{p}$) from $\mathbf{p}$.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_{\mathcal{A}}$ of $\mathcal{A}$

$$\mathbf{p}_{\mathcal{A}}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_{\mathcal{A}}$.

A **profile** is some $\mathbf{p} : Q \to 2^{RegTrees} \cup \{\infty\}$ that is smaller than $\mathbf{p}_{\mathcal{A}}$.

Define a profile counterpart of operators Attr and Safety and show that one can compute $\text{Attr}(\mathbf{p})$ and $\text{Safety}(\mathbf{p})$ from $\mathbf{p}$.

Lemma. Let $\mathbf{p}$ be a profile. If $\mathbf{p} = \text{Attr}(\text{Safety}(\mathbf{p}))$ then $\mathbf{p} = \mathbf{p}_{\mathcal{A}}$.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_{\mathcal{A}}$ of $\mathcal{A}$

$$\mathbf{p}_{\mathcal{A}}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_{\mathcal{A}}$.

A **profile** is some $\mathbf{p} : Q \to 2^{RegTrees} \cup \{\infty\}$ that is smaller than $\mathbf{p}_{\mathcal{A}}$.

Define a profile counterpart of operators Attr and Safety and show that one can compute $\mathrm{Attr}(\mathbf{p})$ and $\mathrm{Safety}(\mathbf{p})$ from $\mathbf{p}$.

Lemma. Let $\mathbf{p}$ be a profile. If $\mathbf{p} = \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$ then $\mathbf{p} = \mathbf{p}_{\mathcal{A}}$.

To converge, add a speed-up operator on profiles: $\mathbf{p} \mapsto \mathrm{SpeedUp}(\mathbf{p})$.

How to Handle the co-Büchi Case? (2/2)

Work with the **infinity profile** $\mathbf{p}_{\mathcal{A}}$ of $\mathcal{A}$

$$\mathbf{p}_{\mathcal{A}}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & \textit{otherwise} \end{cases}$$

Lemma. One can compute the cardinality profile from $\mathbf{p}_{\mathcal{A}}$.

A **profile** is some $\mathbf{p} : Q \to 2^{RegTrees} \cup \{\infty\}$ that is smaller than $\mathbf{p}_{\mathcal{A}}$.

Define a profile counterpart of operators Attr and Safety and show that one can compute $\mathrm{Attr}(\mathbf{p})$ and $\mathrm{Safety}(\mathbf{p})$ from $\mathbf{p}$.

Lemma. Let $\mathbf{p}$ be a profile. If $\mathbf{p} = \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$ then $\mathbf{p} = \mathbf{p}_{\mathcal{A}}$.

To converge, add a speed-up operator on profiles: $\mathbf{p} \mapsto \mathrm{SpeedUp}(\mathbf{p})$.

Lemma. Let $\mathbf{p}_0$ be the profile that maps $\emptyset$ to every state and let, for any $i \geq 0$, $\mathbf{p}_{i+1} = \mathrm{SpeedUp}(\mathrm{Attr}(\mathrm{Safety}(\mathbf{p}_i)))$.
Then $(\mathbf{p}_i)_{i \geq 0}$ converges in a **finite** number of steps to $\mathbf{p}_{\mathcal{A}}$.

Conclusion

Main Contribution: a class of languages of infinite trees that:

- Encompass regular languages.
- Form a Boolean algebra.
- Have interesting expressive power.
- Enjoy a decidable cardinality problem.

Conclusion

Main Contribution: a class of languages of infinite trees that:

- Encompass regular languages.
- Form a Boolean algebra.
- Have interesting expressive power.
- Enjoy a decidable cardinality problem.

Further Work:

- Simplify the proof for the parity case.
- Investigate other decision problems, *eg.* the regularity problem.
- Find automata models with decidable emptiness that capture extension of MSO with isomorphism tests.
- Look for applications.