Rigid Tree Automata With Isolation

Nathaniel Wesley Filardo and Jason Eisner

May 9, 2017

Introduction

We want to analyse Prolog-style programs.

- We're designing a programming language in that school.

Use cases we want to consider:

- Efficient storage of recursive structures with equalities inside. (e.g., [(A,A),(B,B),...] stored as [A,B,...].)
- Improved analysis through recursive structures with equality. (e.g., track aliases into and out of lists)

Review of Rigid Tree Automata

RTA (Jacquemard et al., 2011) are like regular automata

- Set of states Q, $Q_\mathsf{F} \subseteq Q$ "final" states,
- Transition rules of the form $\mathtt{f}\langle q_1, \ldots, q_n \rangle \to q_0$

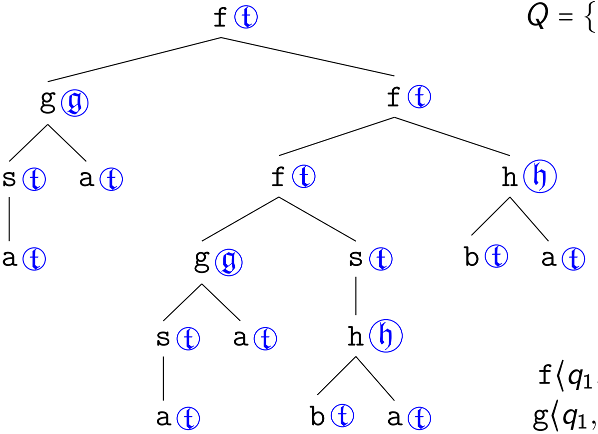
but impose *global* equality constraints:

- Add "rigid states" $Q_\mathsf{R} \subseteq Q$.
- A run is accepted iff
  - All transitions are permitted (as with regular TAs)
  - The root is annotated with a final state (ditto)
  - For each rigid state $q \in Q_\mathsf{R}$, all nodes annotated with q dominate equal trees.

Review of Rigid Tree Automata

Example of RTA (but non-TAC+) language:

- trees over ranked alphabet $\{\mathtt{f}/2, \mathtt{g}/2, \mathtt{h}/2, \mathtt{s}/1, \mathtt{a}/0, \mathtt{b}/0\}$,
- where all g-dominated trees are equal (so, too, h).



$Q = \{\mathfrak{t}, \mathfrak{g}, \mathfrak{h}\}$, $Q_\mathsf{F} = \{\mathfrak{t}\}$

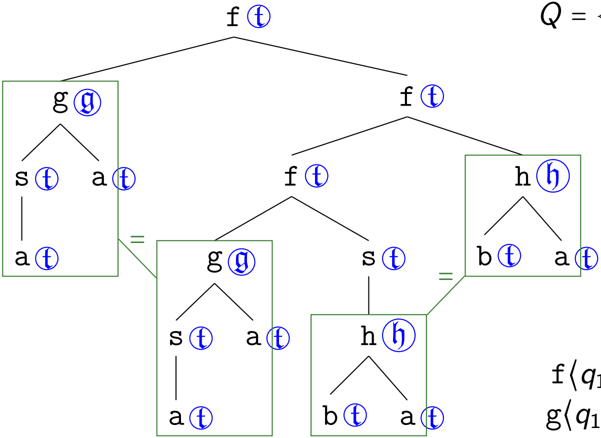
$Q_\mathsf{R} = \{\mathfrak{g}, \mathfrak{h}\}$

$$\mathtt{a}\langle\rangle \to \mathfrak{t}$$
$$\mathtt{b}\langle\rangle \to \mathfrak{t}$$
$$\mathtt{s}\langle q\rangle \to \mathfrak{t} \mid q \in Q$$
$$\mathtt{f}\langle q_1, q_2\rangle \to \mathfrak{t} \mid q_i \in Q$$
$$\mathtt{g}\langle q_1, q_2\rangle \to \mathfrak{g} \mid q_i \in Q$$
$$\mathtt{h}\langle q_1, q_2\rangle \to \mathfrak{h} \mid q_i \in Q$$

Review of Rigid Tree Automata

Example of RTA (but non-TAC+) language:

- trees over ranked alphabet $\{\mathtt{f}/2, \mathtt{g}/2, \mathtt{h}/2, \mathtt{s}/1, \mathtt{a}/0, \mathtt{b}/0\}$,
- where all $\mathtt{g}$-dominated trees are equal (so, too, $\mathtt{h}$).



$Q = \{\mathfrak{t}, \mathfrak{g}, \mathfrak{h}\}$, $Q_{\mathsf{F}} = \{\mathfrak{t}\}$

$Q_{\mathsf{R}} = \{\mathfrak{g}, \mathfrak{h}\}$

$$\mathtt{a}\langle\rangle \to \mathfrak{t}$$
$$\mathtt{b}\langle\rangle \to \mathfrak{t}$$
$$\mathtt{s}\langle q\rangle \to \mathfrak{t} \mid q \in Q$$
$$\mathtt{f}\langle q_1, q_2\rangle \to \mathfrak{t} \mid q_i \in Q$$
$$\mathtt{g}\langle q_1, q_2\rangle \to \mathfrak{g} \mid q_i \in Q$$
$$\mathtt{h}\langle q_1, q_2\rangle \to \mathfrak{h} \mid q_i \in Q$$

Review of Rigid Tree Automata

Finitely many rigid states means no ability to capture languages like...

- $\{[], [\mathtt{p}\langle n_1, n_1\rangle], [\mathtt{p}\langle n_1, n_1\rangle, \mathtt{p}\langle n_2, n_2\rangle], \cdots \mid n_i \in L_\mathsf{n}\}$
- $\{[], [n_1, n_1], [n_1, n_1, n_2, n_2], \cdots \mid n_i \in L_\mathsf{n}\}$

with L_n regular and $|L_\mathsf{n}| = \infty$.

- For *finite* L_n, can absorb equalities into the state space.

Isolation

Isolating RTA adds controlled reuse of rigid states:

- New rule form: $\mathtt{f}\langle q_1, \ldots, q_n\rangle \xrightarrow{!I} q_0$ with $I \subseteq Q_{\mathrm{R}}$.
- Each $q \in I$ is "forgotten" when traversing this rule.
  - *Two trees annotated with the same rigid state must be equal, unless the path between them has an isolation of that state.*
- $I = \varnothing$ everywhere: RTA.

Isolation

Positive Examples

Lists of equal pairs:

$$\{[], [\mathrm{p}\langle n_1, n_1\rangle], [\mathrm{p}\langle n_1, n_1\rangle, \mathrm{p}\langle n_2, n_2\rangle], \cdots \mid n_i \in \mathbb{N}\}$$

$$Q = \{\mathfrak{n}, \mathfrak{n}', \mathfrak{p}, \mathfrak{t}\}$$

$$Q_\mathsf{F} = \{\mathfrak{t}\} \quad Q_\mathsf{R} = \{\mathfrak{n}'\}$$

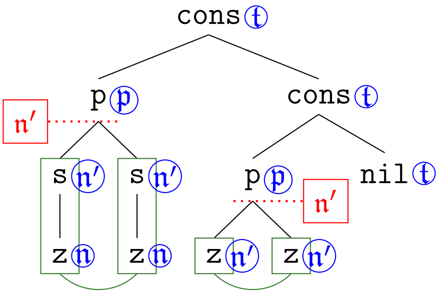
$$\mathrm{z}\langle\rangle \to \{\mathfrak{n}, \mathfrak{n}'\}$$

$$\mathrm{s}\langle\mathfrak{n}\rangle \to \{\mathfrak{n}, \mathfrak{n}'\}$$

$$\mathrm{nil}\langle\rangle \to \mathfrak{t}$$

$$\mathrm{cons}\langle\mathfrak{p}, \mathfrak{t}\rangle \to \mathfrak{t}$$

$$\mathrm{p}\langle\mathfrak{n}', \mathfrak{n}'\rangle \xrightarrow{!\{\mathfrak{n}'\}} \mathfrak{p}$$



Isolation

Positive Examples

Not limited to “arms length”:

$L = \{\#, \mathtt{t}\langle n, l, n\rangle \mid l \in L, n \in \mathbb{N}\}$

$$Q = \{\mathfrak{n}, \mathfrak{n}', \mathfrak{t}\}$$

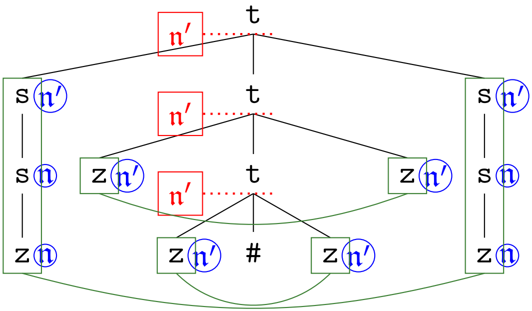
$$Q_F = \{\mathfrak{t}\} \quad Q_R = \{\mathfrak{n}'\}$$

$$\mathtt{z}\langle\rangle \to \{\mathfrak{n}, \mathfrak{n}'\}$$

$$\mathtt{s}\langle\mathfrak{n}\rangle \to \{\mathfrak{n}, \mathfrak{n}'\}$$

$$\#\langle\rangle \to \mathfrak{t}$$

$$\mathtt{t}\langle\mathfrak{n}', \mathfrak{t}, \mathfrak{n}'\rangle \xrightarrow{!\{\mathfrak{n}'\}} \mathfrak{t}$$



Isolation

Positive Examples

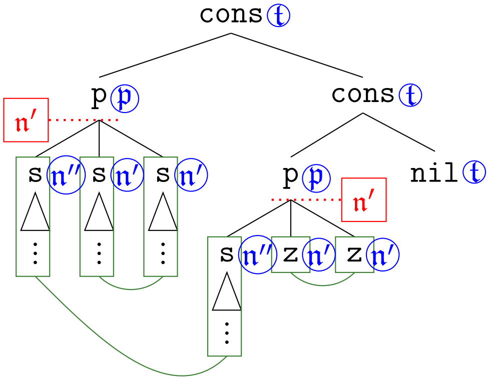
Can mix isolated and non-isolated states:

$\{[], [\mathtt{p}\langle n_0, n_1, n_1\rangle], [\mathtt{p}\langle n_0, n_1, n_1\rangle, \mathtt{p}\langle n_0, n_2, n_2\rangle], \cdots \mid n_i \in \mathbb{N}\}$:

$Q = \{\mathfrak{n}, \mathfrak{n}', \mathfrak{n}'', \mathfrak{p}, \mathfrak{t}\}$

$Q_F = \{\mathfrak{t}\} \quad Q_R = \{\mathfrak{n}', \mathfrak{n}''\}$

$$\mathtt{z}\langle\rangle \to \{\mathfrak{n}, \mathfrak{n}', \mathfrak{n}''\}$$
$$\mathtt{s}\langle\mathfrak{n}\rangle \to \{\mathfrak{n}, \mathfrak{n}', \mathfrak{n}''\}$$
$$\mathtt{nil}\langle\rangle \to \mathfrak{t}$$
$$\mathtt{cons}\langle\mathfrak{p}, \mathfrak{t}\rangle \to \mathfrak{t}$$
$$\mathtt{p}\langle\mathfrak{n}'', \mathfrak{n}', \mathfrak{n}'\rangle \xrightarrow{!\{\mathfrak{n}'\}} \mathfrak{p}$$

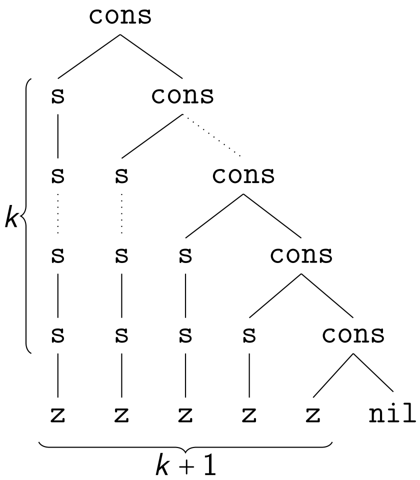


Isolation

Negative Examples

No IRTA for TAC+ language $L = \{[n, n-1, \ldots, 0] \mid n \in \mathbb{N}\}$.

- Claimed IRTA with k states?
  - Take $n = k$.

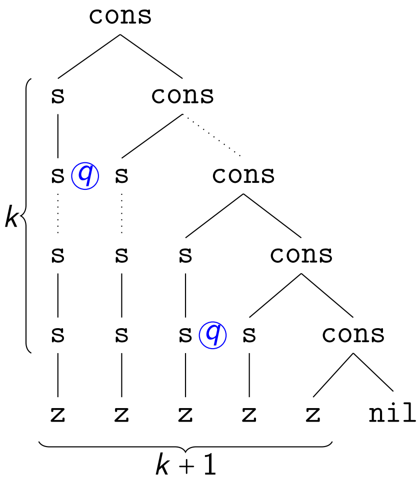


Isolation

Negative Examples

No IRTA for TAC+ language $L = \{[n, n-1, \ldots, 0] \mid n \in \mathbb{N}\}$.

- Claimed IRTA with k states?
  - Take $n = k$.
- Pigeonhole: at least one Peano node's state q is reused for a different tree.

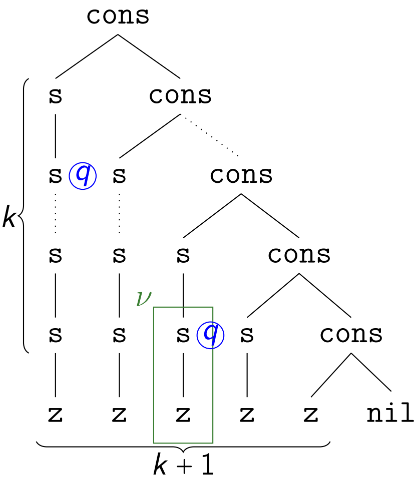


Isolation

Negative Examples

No IRTA for TAC+ language $L = \{[n, n-1, \ldots, 0] \mid n \in \mathbb{N}\}$.

- Claimed IRTA with k states?
  - Take $n = k$.
- Pigeonhole: at least one Peano node's state q is reused for a different tree.
- *Smallest* such node ν dominates states used for only one tree throughout the run!
  - Obey any rigidity constraints.

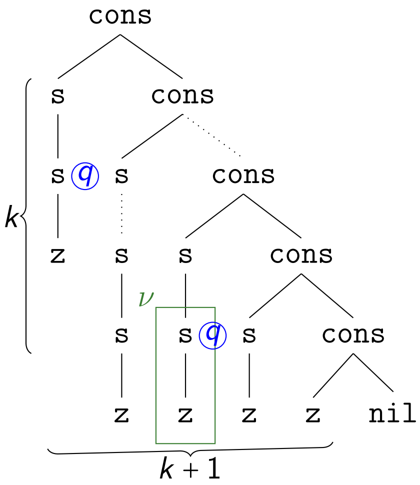


Isolation

Negative Examples

No IRTA for TAC+ language $L = \{[n, n-1, \ldots, 0] \mid n \in \mathbb{N}\}$.

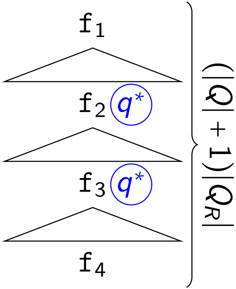
- Claimed IRTA with k states?
  - Take $n = k$.
- Pigeonhole: at least one Peano node's state q is reused for a different tree.
- *Smallest* such node ν dominates states used for only one tree throughout the run!
  - Obey any rigidity constraints.
- Substitute ν in for all q: accepted, $\notin L$.



Isolation

Pumping Lemma

RTA pumping lemma:



- Each $q \in Q_R$ at most once on root-leaf path.
- Root-leaf path of length $(|Q|+1)|Q_R|$ has sub-path with
  - no rigid states within, and
  - equal (non-rigid) terminal state q^*
- Can pump there and rewrite nodes above.
  - There may be rigid states above and as cousins of the pumping site.

- Isolation allows $q \in Q_R$ many times on a root-leaf path!
  - Need new construction!

Isolation

Pumping Lemma

First: rewriting $[t\ q/M]$

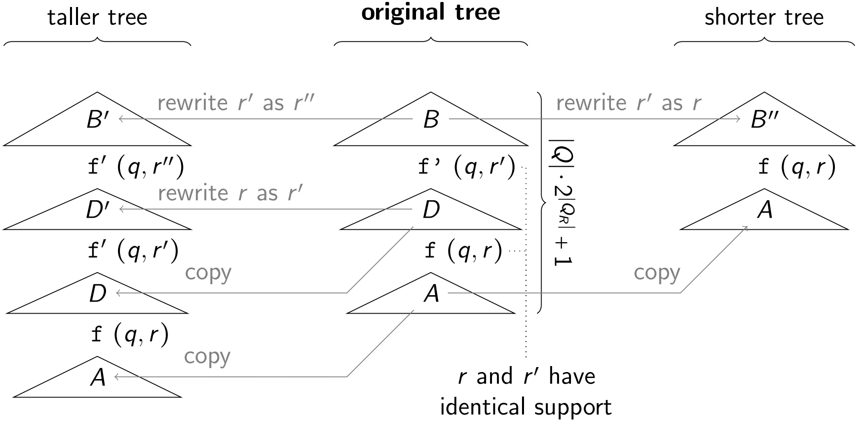
- Input: run on tree t in state q, runs on rigid states M.
  - Runs within M must be compatible
  - M need not contain all rigid states
- Output: new tree t' and run on t' in state q.
- Simple top-down replacement for RTA:

$$\begin{aligned} q \text{ in } M :& [t\ q/\{q \mapsto t', \ldots\}] \mapsto t'\ q \\ \text{Otherwise} :& [\mathtt{f}\langle t_1\ q_1, \ldots\rangle\ q_0/M] \mapsto \mathtt{f}\langle [t_1\ q_1/M], \ldots\rangle\ q_0 \end{aligned}$$

Isolation

Pumping Lemma

Root-leaf path of length $|Q| \cdot 2^{|Q_R|} + 1$: somewhere here-on, a state q will be reused *with the same set of rigid states in scope*.



Isolation

Emptiness Testing

Emptiness of an ((I)R)TA is P-time by witness search:

- Loop while $\exists\ q^*$ un-witnessed s.t. $\exists$ a rule $\mathtt{f}\langle q_1, \ldots, q_k\rangle \to q^*$ s.t. $\forall_i\ q_i$ witnessed.
  - Use q_i witnesses and rule to build q^* witness.
- Iff, when the loop terminates, any $q \in Q_{\mathrm{F}}$ is witnessed, the automaton accepts at least one tree (q's witness).

This algorithm ...

- generates *state-acyclic* witnesses: work for *any* $Q_{\mathrm{R}} \subseteq Q$, any I.
- does not really *use* the witnesses; could just use bits.

For every IRTA A, RTA A' sets $I = \varnothing$ everywhere:

- $\mathcal{L}(A') \subseteq \mathcal{L}(A)$
- $\mathcal{L}(A') = \varnothing \Rightarrow \mathcal{L}(A) = \varnothing$.

Isolation

Boolean Closure

- IRTA trivially closed under union, by nondeterminism.
- IRTA not closed under intersection.
  - Construct a family of machines whose intersection is traces of 2-counter machines' halting runs. (As per TATA, thm 4.4.7)
  - Deciding emptiness of intersection thus Turing-complete.
  - IRTA have trivial emptiness test; not able to represent intersection.
- Conjectured not to be closed under complementation.
  - Lacking a proof at the moment
  - The RTA proof uses balanced binary trees, which *are* IRTA-recognizable.

Future Work: Isolated Rigid Tree Set Automata

We want to analyse Prolog-style programs.

- *Type* analysis:
  - tracks domains of variables (upper-bound answer sets).
  - uses sets of trees, e.g., tree automata.
  - "`f(X) :- g(X,Y),h(Y)`":
    - *intersect* domains of uses of `Y`,
    - Domain of X is *narrowed* by above intersection.
    - Domain of X is a subset of upper bound of `f`'s first argument's domain.

Future Work: Isolated Rigid Tree Set Automata

We want to analyse Prolog-style programs.

- Type-aware *mode* analysis
  - tracks *instantiatedness* of partial answers (shape and domains).
    - Extremes: ground term, free variable (over some domain).
    - Usually: *bound* structure (shape) over free variables.
  - needs *sets of sets* of trees.
  - "`f(X) :- g(X,Y),h(Y)`":
    - "Can rule run if `X` is (not) bound in the call to `f/1`?"
    - "Given variable instantiations, what subgoals are callable (and what is their effect on instantiations)?"
    - Answer questions using *abstract unification*:

$$T_1 \otimes T_2 \stackrel{\text{def}}{=} \{\tau_1 \cap \tau_2 \mid \tau_i \in T_i\}$$

Future Work: Isolated Rigid Tree Set Automata

- Automata framework great for sets of trees; generalize?
- Existing TSA unsuitable
  - Notably, cannot recognize sets of singleton sets.
- New (yes?) framework time!

Future Work: Isolated Rigid Tree Set Automata

New mechanism for describing n-nested sets of trees.

- Consider first a *regular* framework, no constraints.
- Partiton states Q by nesting level: $Q = \bigcup_{i=1}^{n} Q_i$.
- Base case constructors for moving from level k to $k+1$:

$$\begin{aligned} \text{FREE}\; q \to q' \quad &\Rightarrow \quad \mathcal{L}(q) \in \mathcal{L}(q') \\ \text{GROUND}\; q \to q' \quad &\Rightarrow \quad \forall_{\alpha \in \mathcal{L}(q)} \{\alpha\} \in \mathcal{L}(q') \\ \text{SUB}\; q \to q' \quad &\Rightarrow \quad \forall_{\varnothing \subsetneq \alpha \subseteq \mathcal{L}(q)} \alpha \in \mathcal{L}(q') \end{aligned}$$

- Recursive constructor is product former of equal-level states: $\text{BOUND}\; \mathtt{f}\, \langle q_1, \ldots, q_k \rangle \to q_0$. Defined on...
  - trees: $\text{BOUND}\; \mathtt{f}\, \langle t_1, \ldots, t_k \rangle \stackrel{\text{def}}{=} \mathtt{f}\langle t_1, \ldots, t_k \rangle$
  - sets: $\text{BOUND}\; \mathtt{f}\, \langle \tau_1, \ldots, \tau_k \rangle \stackrel{\text{def}}{=} \{\text{BOUND}\; \mathtt{f}\, \langle t_1, \ldots, t_k \rangle \mid t_i \in \tau_i\}$
  - states: $\text{BOUND}\; \mathtt{f}\, \langle q_1, \ldots, q_k \rangle \to q_0$
    $\Rightarrow \quad \text{BOUND}\; \mathtt{f}\, \langle \mathcal{L}(q_1), \ldots, \mathcal{L}(q_k) \rangle \subseteq \mathcal{L}(q_0)$

Future Work: Isolated Rigid Tree Set Automata

Generalise to rigidity:

- A state of any level may be rigid.
  - expands in only one way in a run
- Level-1: equalities within terms inside sets ($\sim$ data variables).
  - $\{\{\mathtt{f}\langle t, t\rangle \mid t \in \tau\}\} = \mathcal{L}(q_{\mathsf{F}})$ if $q_\tau \in Q_{\mathsf{R}}$, $\mathcal{L}(q_\tau) = \tau$ and $\textsc{bound}\ \mathtt{f}\,\langle q_\tau, q_\tau\rangle \to q_{\mathtt{f}}, \textsc{free}\ q_{\mathtt{f}} \to q_{\mathsf{F}}$.
- Level-2: equalities of sets, maybe not terms ($\sim$ type var).
  - $\{\{\mathtt{f}\langle t_1, t_2\rangle \mid t_i \in \tau\} \mid \tau \in T\} = \mathcal{L}(q_{\mathsf{F}})$ if $q_T \in Q_{\mathsf{R}}$, $\mathcal{L}(q_T) = T$ and $\textsc{bound}\ \mathtt{f}\,\langle q_T, q_T\rangle \to q_{\mathsf{F}}$.
- Level-1 isolated during move to level-2:
  - $\{\{\mathtt{f}\langle t, t\rangle\} \mid t \in \tau\} = \mathcal{L}(q_{\mathsf{F}})$ if $q_\tau \in Q_{\mathsf{R}}$, $\mathcal{L}(q_\tau) = \tau$, and $\textsc{bound}\ \mathtt{f}\,\langle q_\tau, q_\tau\rangle \to q_{\mathtt{f}}, \textsc{ground}\ q_{\mathtt{f}} \xrightarrow{!\{q_\tau\}} q_{\mathsf{F}}$.

End result (?): a unified framework for abstract unification.

Questions?