Rigid Tree Automata With Isolation

Nathaniel Wesley Filardo and Jason Eisner

July 17, 2016
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)
RTA (Jacquemard et al., 2011) are like regular automata

- Set of states $Q$, $Q_F \subseteq Q$ “final” states,
- Transition rules of the form $f(q_1, \ldots, q_n) \rightarrow q_0$

but impose *global* equality constraints:

- Add “rigid states” $Q_R \subseteq Q$.
- 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 $q \in Q_R$, all nodes annotated with $q$
    dominate equal trees.
Example of RTA (but non-TAC+) language:

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

\[
Q = \{ t, g, h \}, \quad Q_F = \{ t \} \\
Q_R = \{ g, h \}
\]
Review of Rigid Tree Automata

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

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

\[
Q = \{t, g, h\}, \quad Q_F = \{t\}, \quad Q_R = \{g, h\}
\]

\[
\begin{align*}
a\langle\rangle & \rightarrow t \\
b\langle\rangle & \rightarrow t \\
s\langle q\rangle & \rightarrow t \mid q \in Q \\
f\langle q_1, q_2\rangle & \rightarrow t \mid q_i \in Q \\
g\langle q_1, q_2\rangle & \rightarrow g \mid q_i \in Q \\
h\langle q_1, q_2\rangle & \rightarrow h \mid q_i \in Q
\end{align*}
\]
Finitely many rigid states means no ability to capture languages like...

- $\{[\cdot], [p(n_1, n_1)], [p(n_1, n_1), p(n_2, n_2)], \ldots | n_i \in L_n\}$
- $\{[\cdot], [n_1, n_1], [n_1, n_1, n_2, n_2], \ldots | n_i \in L_n\}$

with $L_n$ regular and $|L_n| = \infty$.

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

**Isolating** RTA adds controlled reuse of rigid states:

- New rule form: $f(q_1, \ldots, q_n) \xrightarrow{I} q_0$ with $I \subseteq Q_R$.
- 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.
- $I = \emptyset$ everywhere: RTA.
Isolation

Positive Examples

Lists of equal pairs:
\[
\{[], [p\langle n_1, n_1 \rangle], [p\langle n_1, n_1 \rangle, p\langle n_2, n_2 \rangle], \ldots \mid n_i \in \mathbb{N}\}
\]

\[
Q = \{n, n', p, t\} \\
Q_F = \{t\} \quad Q_R = \{n'\}
\]

\[
z\langle \rangle \rightarrow \{n, n'\} \\
s\langle n \rangle \rightarrow \{n, n'\} \\
nil\langle \rangle \rightarrow t \\
\text{cons}\langle p, t \rangle \rightarrow t \\
p\langle n', n' \rangle \xrightarrow{\text{cons}} p
\]
Isolation

Positive Examples

Not limited to “arms length”:

\[ L = \{ \#, t(n, l, n) \mid l \in L, n \in \mathbb{N} \} \]

\[ Q = \{ n, n', t \} \]

\[ Q_F = \{ t \} \quad Q_R = \{ n' \} \]

\[ z\langle \rangle \rightarrow \{ n, n' \} \]

\[ s\langle n \rangle \rightarrow \{ n, n' \} \]

\[ \#\langle \rangle \rightarrow t \]

\[ t\langle n', t, n' \rangle \xrightarrow{\{ n' \}} t \]
Isolation

Positive Examples

Can mix isolated and non-isolated states:
\[
\{[], [p(n_0, n_1, n_1)], [p(n_0, n_1, n_1), p(n_0, n_2, n_2)], \ldots | n_i \in \mathbb{N}\}:
\]

\[
Q = \{n, n', n'', p, t\},
\]

\[
Q_F = \{t\},
\]

\[
Q_R = \{n', n''\},
\]

\[
z() \rightarrow \{n, n', n''\},
\]

\[
s(n) \rightarrow \{n, n', n''\},
\]

\[
nil() \rightarrow t,
\]

\[
cons(p, t) \rightarrow t,
\]

\[
p(n'', n', n') \xrightarrow{!\{n'\}} p
\]
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 $n = k$.

\[
\begin{array}{c}
\text{cons} \\
\text{s} \quad \text{cons} \\
\text{s} \quad \text{s} \quad \text{cons} \\
\text{s} \quad \text{s} \quad \text{s} \quad \text{s} \quad \text{cons} \\
\text{s} \quad \text{s} \quad \text{s} \quad \text{s} \quad \text{s} \quad \text{s} \quad \text{cons} \\
\text{z} \quad \text{z} \quad \text{z} \quad \text{z} \quad \text{z} \quad \text{z} \quad \text{z} \quad \text{nil} \\
\end{array}
\]
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 \( L = \{[n, n-1, \ldots, 0] | n \in \mathbb{N}\} \).

- 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 \( \nu \) 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 \( \nu \) dominates states used for only one tree throughout the run!
    - Obey any rigidity constraints.
- Substitute \( \nu \) 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:

\[
q \text{ in } M : [ t \ q / \{ q \mapsto t', \ldots \} ] \mapsto t' \ q
\]

Otherwise:

\[
[ f\langle t_1 \ q_1, \ldots \rangle \ q_0 / M ] \mapsto f\langle [ t_1 \ q_1 / M ], \ldots \rangle \ q_0
\]
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.

original tree

$r$ and $r'$ have identical support

f' $(q, r')$

f $(q, r)$

A

D

D'

f' $(q, r')$

f' $(q, r''')$

B'

B

f $(q, r)$

A

taller tree

shorter tree

B''

r and r' have identical support

rewrite $r'$ as $r''$

rewrite $r$ as $r'$

rewrite $r'$ as $r$

copy

copy

copy

copy
Isolation
Emptiness Testing

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

- Loop while $\exists q^* \text{ un-witnessed s.t. } \exists \text{ a rule } f(q_1, \ldots, q_k) \rightarrow q^*$ s.t. $\forall i \ q_i \text{ witnessed.}$
  - Use $q_i$ witnesses and rule to build $q^*$ witness.
- Iff, when the loop terminates, any $q \in Q_F$ is witnessed, the automaton accepts at least one tree ($q$’s witness).

This algorithm ...

- generates state-acyclic witnesses: work for any $Q_R \subseteq Q$, any $I$.
- does not really use the witnesses; could just use bits.

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

- $L(A') \subseteq L(A)$
- $L(A') = \emptyset \Rightarrow L(A) = \emptyset$. 


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.
  - “\(f(X) \leftarrow g(X, Y), h(Y)\)”: 
    - “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*:

\[
T_1 \bowtie T_2 \overset{\text{def}}{=} \{ \tau_1 \cap \tau_2 \mid \tau_i \in T_i \}
\]
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.
- Partition states $Q$ by nesting level: $Q = \bigcup_{i=1}^{n} Q_i$.
- Base case constructors for moving from level $k$ to $k + 1$:
  - **FREE** $q \rightarrow q'$ \implies \{ $L(q)$ \} \subseteq $L(q')$
  - **GROUND** $q \rightarrow q'$ \implies $\forall \alpha \in L(q) \{ \alpha \} \subseteq L(q')$
  - **SUB** $q \rightarrow q'$ \implies $\forall \varnothing \neq \alpha \subseteq L(q) \{ \alpha \} \subseteq L(q')$
- Recursive constructor is product former of equal-level states:
  - **BOUND** $f \langle q_1, \ldots, q_k \rangle \rightarrow q_0$. Defined on...
    - trees: **BOUND** $f \langle t_1, \ldots, t_k \rangle \overset{\text{def}}{=} f\langle t_1, \ldots, t_k \rangle$
    - sets: **BOUND** $f \langle \tau_1, \ldots, \tau_k \rangle \overset{\text{def}}{=} \{ \text{BOUND} f \langle t_1, \ldots, t_k \rangle | t_i \in \tau_i \}$
    - states: **BOUND** $f \langle q_1, \ldots, q_k \rangle \rightarrow q_0$
      \implies \text{BOUND} $f \langle L(q_1), \ldots, L(q_k) \rangle \subseteq L(q_0)$
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).
  - $\{\{f(t, t) \mid t \in \tau\}\} = \mathcal{L}(q_F)$ if $q_{\tau} \in Q_R$, $\mathcal{L}(q_{\tau}) = \tau$ and $\text{BOUND } f(q_{\tau}, q_{\tau}) \to q_f$, $\text{FREE } q_f \to q_F$.
- Level-2: equalities of sets, maybe not terms ($\sim$ type var).
  - $\{\{f(t_1, t_2) \mid t_i \in \tau\} \mid \tau \in T\} = \mathcal{L}(q_F)$ if $q_T \in Q_R$, $\mathcal{L}(q_T) = T$ and $\text{BOUND } f(q_T, q_T) \to q_F$.
- Level-1 isolated during move to level-2:
  - $\{\{f(t, t)\} \mid t \in \tau\} = \mathcal{L}(q_F)$ if $q_{\tau} \in Q_R$, $\mathcal{L}(q_{\tau}) = \tau$, and $\text{BOUND } f(q_{\tau}, q_{\tau}) \to q_f$, $\text{GROUND } q_f \xrightarrow{!\{q_{\tau}\}} q_F$.

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