Rigid Tree Automata With Isolation

Nathaniel Wesley Filardo and Jason Eisner
Johns Hopkins University

Abstract. Rigid Tree Automata (RTAs) are a strict super-class of Regular Tree Automata (TAs), additionally capable of recognizing certain nonlinear patterns such as \( \{ f(x, x) | x \in X \} \). RTAs were developed for use in tree-automata-based model checking; we hope to use them as part of a static analysis system for a logic programming language. In developing that system, we noted that RTAs are not closed under Kleene-star or pre-concatenation with a regular language. We now introduce a strict super-class of RTA, called Isolating Rigid Tree Automata, which can accept rigid structures with arbitrarily many isolated rigid substructures, such as “lists of equal pairs,” by allowing rigidity to be confined within subtrees. This class is Kleene-star and concatenation closed and retains many features of RTAs, including linear-time emptiness testing and NP-complete membership testing. However, it gives up closure under intersection.

1 Rigid Tree Automata

Rigid Tree Automata (RTAs) \([2]\) extend regular bottom-up nondeterministic Tree Automata by imposing \textit{global} constraints on accepting runs. They are well-suited to describe regular structures containing finitely many typed variables, such as \( \{ f(g(x), h(x, y)) | x \in L, y \in L' \} \) where \( L, L' \) are regular tree languages representing types. They can also describe families of “all-equal lists” \( \{ [], [x], [x, x], [x, x, x], \ldots | x \in L \} \).\(^1\) As these examples show, variables may be reused, each occurrence \textit{co-varying} with the others. RTAs may also express \textit{unions} of such nonlinear structures, including infinite unions via recursion, as in the case of all-equal lists.

An RTA is very much like a TA. Each has an underlying language signature \( \mathcal{F} \); a set of states \( Q \); a set of accepting states \( Q_F \subseteq Q \); and a transition map \( \Delta \), which is a set of rules of the form \( f(q_1, \ldots, q_m) \rightarrow q_0 \) where \( q_i \in Q \) and \( f/n \in \mathcal{F} \). A run of an RTA \( A \) on a tree \( t \) is exactly like that of a TA: a map that annotates each node \( \nu \) of \( t \) with a state from \( Q \) in a way that respects \( \Delta \). That is, if node \( \nu \) has label \( g/m \in \mathcal{F} \) and its \( m \) children are annotated with \( q_1, \ldots, q_m \in Q \), then \( \nu \) may be annotated with \( q_0 \) if \( (g(q_1, \ldots, q_m) \rightarrow q_0) \in \Delta \).

The novelty of the RTA class is that an RTA designates a set of \textbf{rigid} states, \( Q_R \subseteq Q \), and runs are accepted more selectively. A tree is \textbf{accepted} by the RTA \( A = (\mathcal{F}, Q, Q_F, Q_R, \Delta) \) iff there exists a run in which the root position is annotated by \( q \in Q_F \) (this is the TA acceptance criterion) and, for each \( q \in Q_R \), all

\(^1\) We adopt some standard shorthand: \([ \] = \text{n1}() \) and \([a, b, \ldots] = \text{cons}(a, \text{cons}(b, \ldots)) \).
subtrees whose roots are annotated by $q$ are equal.\(^2\) For example, \{h(x, x) \mid x \in L\} is recognized by an RTA $\langle \mathcal{F} \cup \{h/2\}, Q \cup \{q^*, \{q_f\}, \Delta \cup \{h(q_f, q_p) \rightarrow q^*\}\} \rangle$ if $q^* \notin Q$ and $L$ is recognized by a regular TA $A = \langle \mathcal{F}, Q, \{q_r\}, \Delta \rangle$ whose sole accepting state $q_f$ is non-reentrant (i.e., only occurs on the right of rules in $\Delta$).\(^3\) The set of languages described by RTAs are a strict superset of those described by regular TAs [2, Theorem 5]: the RTA language above is not regular, but any regular TA is an RTA with $Q_R = \emptyset$.

2 Kleene Non-Closure of Rigid Tree Automata

RTA cannot, however, describe (finite) structures with arbitrary numbers of variables, as each variable corresponds to a state in $Q_R$. Let us look at two examples. We use the notations $\cdot_\sqcap$, $L^*_{\sqcap}$, and $L^n_{\sqcap}$ as defined in [1, §2.2.1].

First, consider $P = \{[], [p(x_1, x_1)], [p(x_1, x_1), p(x_2, x_2)], \ldots \mid x_i \in L_x\}$, with $L_x$ regular and $|L_x| = \infty$.\(^4\) The RTA pumping lemma [2, Lemma 1] says that no RTA can recognize $P$. (The essential obstacle is that $P$ needs to enforce separate equalities on unboundedly many pairs, which cannot be done with only finitely many rigid states.) This implies that the RTA family is not closed under pre-concatenation with a regular language, since $P = L \sqcap M$ where $L = \{\text{nil}() \cup \text{cons}(\varnothing, l) \mid l \in L\}$ is regular (note the recursive definition, allowing trees with arbitrarily many $\varnothing$ leaves) and $M = \{p(x, x) \mid x \in L_x\}$ is rigid. RTAs are trivially closed under post-concatenation with a regular language: $L \sqcup M$ is an RTA language over $\mathcal{F}$ if $L$ is rigid over $\mathcal{F} \cup \{\varnothing\}$ and $M$ is regular over $\mathcal{F}$, as the rigidity in $L$ will not be able to test the structure induced by concatenation with $M$, making concatenation behave locally as if $L$ were regular.\(^5\)

Second, consider the set of lists $D = \{[], [x_1, x_1], [x_1, x_1, x_2, x_2], \ldots \mid x_i \in L_x\}$ for some regular $L_x$ with $|L_x| = \infty$. Again, the RTA pumping lemma implies that $D$ cannot be recognized by an RTA. This shows that RTAs are not closed under Kleene-star, since $D = E^{*_{\sqcap}}$ for the RTA language $E = \{\text{nil} \cup \{\text{cons}(x, \text{cons}(x, \varnothing)) : x \in L_x\}\}$. Note that $E^{k_{\sqcap}}$ is an RTA language for any finite $k$ and any regular (or even rigid) language $L_x$.

3 Isolation

We augment RTA transition rules with the ability to discard rigidity constraints across subtrees, introducing Isolating Rigid Tree Automata (IRTA), a proper

\(^2\) The states $Q_R$ are thus “rigid” as each expands in one way throughout the tree.

\(^3\) These requirements on accepting states of $A$ are needed for our RTA construction, in which $q_F$ becomes a rigid state. However, they involve no loss of generality, since if $L$ is recognized by any regular TA $A' = \langle \mathcal{F}, Q, Q_F, \Delta \rangle$, it is also recognized by an equivalent one that uses a single, non-reentrant accepting state, as required: $A = \langle \mathcal{F}, Q \cup \{q_f\}, \{q_F\}, \Delta \cup \{\text{cons}(x_1, \ldots, x_k) \rightarrow q_f \mid (\text{cons}(x_1, \ldots, x_k) \rightarrow q) \in \Delta, q \in Q_F\}\rangle$.

\(^4\) For concreteness and to avoid any ability of the lemma to find pumping opportunities in $L_x$, restrict to runs over “short” trees from $L_x$ for this and the next example.

\(^5\) One could define a notion of concatenation that was more specialized to RTAs, where $\varnothing$ itself was interpreted rigidly. On this definition, RTAs would be closed under both pre- and post-concatenation with regular languages.
super-class of RTA. Each transition rule is decorated with a set of rigid states to isolate, making it of the form $f(q_1, \ldots, q_n) \xrightarrow{I} q_0$ with $f/n \in F$, $\forall i, q_i \in Q$, and $I \subseteq Q$. Intuitively, when such a rule is used in a run to reach a node $\nu$, the equality constraint for a rigid state $q \in I$ is no longer enforced between $\nu$-annotated nodes strictly dominated by $\nu$ and $q$-annotated nodes elsewhere. Every RTA is an IRTA with $I = \emptyset$ everywhere.

The non-RTA examples from before are easily captured (see Figure 1 in the appendix for illustrations). As before, suppose that $L_\xi$ is recognized by the TA $A = (F, Q, \{q_F\}, \Delta)$ without non-reentrant accepting state $q_F$. Then taking $F' = F \cup \{p/2, \text{cons}/2, \text{nil}/0\}$,

- The language $P$ is recognized by the IRTA $(F', Q \cup \{q^*\}, \{q_F\}, \Delta')$ with $\Delta' = \Delta \cup \{p(q_F, q_F) \overset{\{q_F\}}{\longrightarrow} q_p, \text{cons}(q_p, q^*) \rightarrow q^*, \text{nil}\} \rightarrow q^*\}.

- The language $D$ is recognized by the IRTA $(F', Q \cup \{q_1^*, q_2^*\}, \{q_F\}, \Delta')$ with $\Delta' = \Delta \cup \{\text{cons}(q_F, q_1^*) \rightarrow q_2^*, \text{cons}(q_F, q_2^*) \overset{\{q_F\}}{\longrightarrow} q_1^*, \text{nil}\} \rightarrow q_1^*\}.

The use of $\emptyset \not\subseteq I \not\subseteq Q$ allows for hybrid structures with both global and local equalities, such as $D' = \{12, \{x_0, x_1, x_1\}, \{x_0, x_0, x_0, x_2, x_2\}, \ldots \mid x_i \in L_\xi\}. Here the equality of every third entry $(x_0)$ would be enforced throughout the entire list using a rigid state that is not isolated (à la RTA), while the other entries are only equal in adjacent pairs, using a rigid state that is periodically isolated as in $D$.

To describe the semantics of IRTA rules more formally, we first restate the acceptance condition for TAs and RTAs as a bottom-up algorithm for generating accepting runs, if any, on an input tree. A simple change then will suffice to make this algorithm construct IRTA runs.

Membership testing for a deterministic TA can be accomplished by bottom-up annotation of the given tree $t$. A step of this algorithm visits any unannotated node of $t$ whose children have already been annotated, and annotates it with the only state that respects $\Delta$ (given the child annotations), or rejects $t$ if there is no such state. $t$ is accepted if the root is annotated by a final state. In the nondeterministic case, each node of $t$ is simultaneously annotated with all states that can respect $\Delta$ (given some choice of the child annotations), and $t$ is accepted if its root node is annotated with at least one final state.

We can extend this approach to RTAs by augmenting the annotations. Let $t_\nu$ denote the subtree of $t$ rooted at node $\nu$. Each annotation of $\nu$, rather than being a state in $Q$, is now a pair $(q, r) \in Q \times \mathcal{P}(Q_R \times T(F))$. Intuitively, this pair records the existence of some run on $t_\nu$ that annotates $\nu$ with $q$, where $r : Q_R \rightarrow \{\text{subtrees of } t_\nu\}$ is a partial function (represented as a set of ordered

6 In this work, we consider the family of nondeterministic (I)RTAs. Of course there is also a class of deterministic IRTAs that generalize deterministic RTAs.

7 We choose the isolating set $I$ as part of the transition rule. In the case of deterministic IRTAs, however, it might increase power to change the form of the rules to defer the choice of $I$ until the next rule is selected. The next rule would then have the form $g(\ldots, q_{0i}, \ldots) \rightarrow q_{1i}$, allowing the choice of $I$ at the $q_{0i}$-annotated node $\nu$ depend on the annotations at $\nu$’s siblings, and on the functor $g$ and annotation $q_{i-1}$ at $\nu$’s parent.
pairs) that maps each rigid state \( q' \) used in the run to the tree \( t' \) such that \( q' \) was used in the run only to annotate the roots of copies of \( t' \). When visiting a node \( \nu \) with label \( g/m \), if \( (g(q_1, \ldots, q_m) \rightarrow q) \in \Delta \) and the \( m \) children are annotated with \( (q_1, r_1), \ldots, (q_m, r_m) \), the algorithm annotates this node with \( (q, r) \), provided that \( r = \bigcup_{i=0}^{m} r_i \) is a partial function, where \( r_0 = \{ (q, t_\nu) \} \) if \( q \in Q_R \) and otherwise \( r_0 = \emptyset \). The full tree \( t \) is accepted if its root has a label \( (q, r) \) for some \( q \in Q_F \).

The generalization to IRTAs is now straightforward: the algorithm simply “forgets” subtrees when directed to do so by the transition rules. When visiting a node \( \nu \) with label \( g/m \), if \( (g(q_1, \ldots, q_m) \rightarrow^I q) \in \Delta \) and the \( m \) children are annotated with \( (q_1, r_1), \ldots, (q_m, r_m) \), the algorithm computes \( r' = r_0 \cup \{ (q', r') \in r \mid q' \notin I \} \), where \( r = \bigcup_{i=0}^{m} r_i \) and \( r_0 \) is as before, and annotates this node with \( (q, r') \), provided that \( r' \) is a partial function.

4 Pumping Lemma

The pumping lemma construction for RTAs given in [2, §2.4] relies heavily on the fact that any path from a the root of an accepted run to a leaf thereof will contain each rigid state at most once. Thus if there is an accepting run with a path of length \(|Q_R|(1 + |Q|)\), there must exist a nontrivial sub-path with all nodes there-on labeled with states from \( Q \setminus Q_R \) (i.e., not rigidly) and with both endpoints equally labeled. This is no longer true in IRTA: a root-leaf path in an accepted run can contain a rigid state at most once between isolations of that state, but isolations may occur arbitrarily often.

Nevertheless, a pumping-style construction is still possible (see Figure 2 for an illustration). Given an accepted tree \( t \) of height \(|Q| \cdot 2^{|Q|\cdot n} + 1 \), a root-leaf path of that length is guaranteed to have two distinct nodes analyzed with the same (possibly rigid) state and with the same set of rigid states having not been isolated. Let two such colliding nodes be \( \delta \) and \( \alpha \), respectively labeled as \( (q, r) \) and \( (r, r') \) with \( r \) and \( r' \) having equal domains. We can then partition the tree into three regions by writing it as \( B[D[A]] \), where \( B \) (“before”) and \( D \) (“during”) are 1-contexts, with \( D \) rooted at \( \delta \), and \( A = t_\alpha \) (“after”) is a tree rooted at \( \alpha \). We can construct a new 1-context \( D' \) from \( D \) by “rewriting”: use the values from \( r' \), rather than \( r \), to satisfy rigid states in \( D \), traversing bottom up and manipulating \( r' \) as directed by the automaton’s rules. The result will be a revised label of \( (q, r'') \) for the root of \( D' \); use the same rewrite procedure to turn \( B \), which used rigid trees from \( r' \), into \( B' \) using \( r'' \). Now \( B''[D'[D[A]]] \) is another accepted tree satisfying the pumping preconditions. One could, alternatively, rewrite \( B \) to \( B'' \) using \( r \) to obtain \( B''[A] \), another accepted tree.

This pumping construction merely builds other trees; it does not repeat parts of the tree structure exactly. Still, it shows that if an IRTA accepts a sufficiently tall tree, it accepts infinitely many trees. It also shows an argument (different from that of §5.1 below) that emptiness of an IRTA’s language is decidable: one could exhaustively enumerate and test trees of height up to \(|Q| \cdot 2^{|Q|\cdot n} \) only, since the shortest accepted tree cannot be taller than that—an such tree could be pumped down using the \( B''[A] \) construction.
5 Decision Problems

5.1 Emptiness

RTAs may be tested for non-emptiness using a state-marking algorithm [2, §6.1]. The RTA algorithm constructs acyclic runs, demonstrating occupancy of the RTA’s states by visiting them in a “depth-first” order. If a state is non-empty, then this algorithm will construct a witness tree for it of height at most $n$, where $n$ is the number of states in the RTA. The RTA is non-empty iff at least one of its final states is non-empty.

To find a witness of an IRTA’s non-emptiness, it suffices to find a witness for the corresponding RTA (which drops the ‘I’ decoration, and thus enforces even more equality than the IRTA requires). This works because if the IRTA has any witness $t$, then it has a witness $t'$ that would be accepted by the RTA, which can be found by rewriting subtrees to be equal much as in section 4.

5.2 Membership Testing

As with RTAs [2, §6.2], membership testing of a tree $t$ (with $n$ nodes) against an IRTA $(\mathcal{F}, Q, Q_F, Q_R, \Delta)$ is NP-complete. The proof for RTA reduces 3-SAT to membership testing. We need only show that an annotation of $t$’s nodes can be checked in polynomial time to determine whether it constitutes a valid run (section 3). This involves checking each node of $t$ separately to ensure that its annotation $(q, r)$ can be derived from the annotations of its children by one of the rules in $\Delta$. Given such a rule, checking the $r$ annotation (which dominates the runtime) involves comparing at most $a|Q_R|$ pairs of subtrees of $t$, each having at most $n$ nodes, where $a$ is an upper bound on the number of children (the largest arity of any symbol in $\mathcal{F}$). Thus, the total runtime is $O(an^2|Q_R||\Delta|)$.

5.3 Universality

As all RTAs are IRTAs, tests for universality ($\mathcal{L}(A) = \mathcal{T}\mathcal{F}$?), equality ($\mathcal{L}(A) = \mathcal{L}(A')$?), and inclusion ($\mathcal{L}(A) \subseteq \mathcal{L}(A')$?) all remain non-computable for our new class: the proof from [2, §6.4] continues to hold. For practical purposes, we envision the possibility of a 3-way inclusion test that spends limited computational power to prove or disprove inclusion, but sometimes fails to do either.

6 Closure Properties

Pre-concatenation with a Regular Language IRTAs are, by design, trivially closed under this operation. When constructing an IRTA for $L \odot M$ from an IRTA for $M$, where $L$ is regular over $\mathcal{F} \cup \{\square\}$, isolate all rigid states in $M$ on any transition to the sole $L$ state that labels $\square$.

Kleene Closure Similarly, when constructing an IRTA for $L^* \odot$ from an IRTA for $L$ over $\mathcal{F} \cup \{\square\}$, isolate all rigid states of $L$ on transitions to the $\square$ state.

Projection Closure If $L_x$ is an IRTA language, then the set of trees that appear at a given address $\alpha$ (e.g., 1st child of 2nd child of root) within trees of $L_x$ is also an IRTA language. After eliminating unreachable rules (rules that contain empty IRTA states as determined by §5.1) to obtain a “trimmed” IRTA

---

8 Hash consing can eliminate a factor of $n$ by allowing $O(1)$-time subtree comparison.


\[ \langle \mathcal{F}, Q, Q_{F}, Q_{R}, \Delta \rangle, \text{ a simple recursive algorithm can nondeterministically follow transitions of } \Delta \text{ backwards from } Q_{F} \text{ to find the collection } Q_{q} \text{ of states that can appear at address } \alpha. \text{ The desired IRTA is then } \langle \mathcal{F}, Q, \cup_{q \in Q_{q}} Q_{q}, Q_{R}, \Delta \rangle. \]

**Complementation Non-closure** We conjecture that IRTAs are, like RTAs, not closed under complementation. The existing demonstration from [2, Example 7 and §4.2] is, however, no longer sufficient: the set \( B \) of balanced binary trees over \( \mathcal{F} = \{ a/0, f/2 \} \) is an IRTA language. Let \( Q = \{ q_{0}, q_{1} \} \); then \( B \) is recognized by \( \langle \mathcal{F}, Q, Q, \{ a() \rightarrow q_{0}, f(q_{0}, q_{0}) \rightarrow_{(q_{0})} q_{1}, f(q_{1}, q_{1}) \rightarrow_{(q_{1})} q_{0} \} \rangle \). Unfortunately, finding a replacement has proven tricky!

**Intersection Non-closure** It is possible to construct a series of IRTA machines whose intersection would give the language of accepting runs of a two-counter machine, as in [1, Thm. 4.4.7]. Therefore, as IRTA has a decidable emptiness test, it must not be intersection-closed. Despite that, we conjecture that some special cases of intersection may still be possible; in particular, we speculate that intersecting an IRTA language with either a regular language or an RTA language will tractably yield an IRTA language.

**Union Closure** IRTAs are trivially closed under union, by nondeterminism.

7 **Comparison to TAC+/ TA_\_**

The IRTA class is neither more general nor more specific than tree automata with local equality constraints (TAC+ or TA_\_, [3]). The non-inclusion of IRTA in TAC+ follows from the non-inclusion of RTA. RTA’s ability to enforce constraints globally rather than solely at fixed relative positions allow it to recognize, e.g., the class of trees \( L = \{[0],[1,0],\ldots,[n,n-1,\ldots,1,\ldots]\} \) (with integers represented as their Peano encodings). \( L \) is recognized by the TAC+ \( \{ \{ \text{z/0, s/1, nil/0, cons/2} \}, \{ q_{z}, q_{s}, q_{n}, q_{c} \}, \{ q_{c} \}, \Delta \} \), where \( \Delta = \{ \text{cons}(q_{c}, q_{c}) \rightarrow_{11+21} q_{c}, \text{z()} \rightarrow q_{c}, \text{s}(q_{c}) \rightarrow q_{c}, \text{s}(q_{c}) \rightarrow q_{n}, \text{nil()} \rightarrow q_{n}, \text{cons}(q_{c}, q_{n}) \rightarrow q_{c} \} \). The first rule in \( \Delta \) is the centerpiece. \( L \) is not an IRTA language: suppose that \( L \) is recognized by an IRTA \( A \) with \( k \) states, and consider an accepting run of \( A \) on \( t = [k,\ldots,1,0] \). Let \( \nu \) be a minimum-height Peano node of \( t \) such that its state annotation \( q_{\nu} \) is reused for some \( \nu' \) in \( t \) with \( t_{\nu} \neq t_{\nu'} \). \( \nu \) exists by pigeonhole. By minimality, each proper descendant of \( \nu \) uses a state that annotates equal trees throughout the run on \( t \). Substituting \( t_{\nu} \) in for all \( q_{\nu} \)-annotated nodes yields another accepting run on a new tree \( t' \). However, \( t' \notin L \): either \( t' \) is not a list, or \( t' \) has the same length as \( t \) but different elements.

8 **Conclusion**

We have introduced a new class of automata, Isolating Rigid Tree Automata, which are a Kleene-closed super-class of Rigid Tree Automata. We hope, despite the loss of intersection closure, that IRTA will be useful for modeling inductive (i.e., recursive) data types for programming languages where a data constructor may make non-linear use of its (finitely many) arguments (e.g., Prolog).
References
1. Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Loding, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications.

A Additional Figures

(a) An example tree from $P$.
(b) An example tree from $D$.

Fig. 1: Runs of IRTAs, as given in § 3, for languages defined in § 2. Horizontal dotted lines indicate isolation: any two nodes labeled by the same rigid state must dominate equal trees, unless separated by a line labeled by that state.

Fig. 2: Graphic depiction of the IRTA pumping construction of § 4, showing how to derive both a shorter and taller tree from a tree of height $|Q| \cdot 2^{|Q|} + 1$. 