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 $\{\mathtt{f}\langle x,x\rangle \mid x \in X\}$. 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 *global* constraints on accepting runs. They are well-suited to describe regular structures containing finitely many typed variables, such as $\{\mathtt{f}\langle \mathtt{g}\langle x\rangle, \mathtt{h}\langle x,y\rangle\rangle \mid x \in L, y \in L'\}$ where L, L' are *regular* tree languages representing types. They can also describe families of "all-equal lists" $\{[], [x], [x,x], [x,x,x], \ldots \mid x \in L\}$.[1] As these examples show, variables may be reused, each occurrence *co-varying* with the others. RTAs may also express *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 Δ, which is a set of rules of the form $\mathtt{f}\langle q_1,\ldots,q_n\rangle \to q_0$ where $\forall_i q_i \in Q$ and $\mathtt{f}/n \in \mathcal{F}$. A **run** of an RTA A on a tree t is exactly like that of a TA: a map that annotates each node ν of t with a state from Q in a way that **respects** Δ. That is, if node ν has label $\mathtt{g}/m \in \mathcal{F}$ and its m children are annotated with $q_1,\ldots,q_m \in Q$, then ν may be annotated with q_0 if $(\mathtt{g}\langle q_1,\ldots,q_m\rangle \to q_0) \in \Delta$.

The novelty of the RTA class is that an RTA designates a set of **rigid** states, $Q_R \subseteq Q$, and runs are accepted more selectively. A tree is **accepted** by the RTA $A = \langle \mathcal{F}, Q, Q_F, Q_R, \Delta\rangle$ 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: $[] = \mathtt{nil}\langle\rangle$ and $[a,b,\ldots] = \mathtt{cons}\langle a, \mathtt{cons}\langle b,\ldots\rangle\rangle$.

subtrees whose roots are annotated by q are *equal*.[2] For example, $\{\mathtt{h}\langle x, x\rangle \mid x \in L\}$ is recognized by an RTA $\langle \mathcal{F} \cup \{\mathtt{h}/2\}, Q \cup \{q^*\}, \{q^*\}, \{q_F\}, \Delta \cup \{\mathtt{h}\langle q_F, q_F\rangle \to q^*\}\rangle$ if $q^* \notin Q$ and L is recognized by a regular TA $A = \langle \mathcal{F}, Q, \{q_F\}, \Delta\rangle$ whose sole accepting state q_F is **non-reentrant** (i.e., only occurs on the right of rules in Δ).[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 $Q_R = \varnothing$.

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 $\cdot_\square$, $L^{*,\square}$, and $L^{n,\square}$ as defined in [1, §2.2.1].

First, consider $P = \{[], [\mathtt{p}\langle x_1, x_1\rangle], [\mathtt{p}\langle x_1, x_1\rangle, \mathtt{p}\langle x_2, x_2\rangle], \cdots \mid x_i \in L_\mathrm{x}\}$, with L_x regular and $|L_\mathrm{x}| = \infty$.[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 $P = L \cdot_\square M$ where $L = \{\mathtt{nil}\langle\rangle, \mathtt{cons}\langle \square, l\rangle \mid l \in L\}$ is regular (note the recursive definition, allowing trees with arbitrarily many $\square$ leaves) and $M = \{\mathtt{p}\langle x, x\rangle \mid x \in L_\mathrm{x}\}$ is rigid. RTAs *are* trivially closed under *post*-concatenation with a regular language: $L \cdot_\square M$ is an RTA language over $\mathcal{F}$ if L is rigid over $\mathcal{F} \cup \{\square\}$ 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 $D = \{[], [x_1, x_1], [x_1, x_1, x_2, x_2], \cdots \mid x_i \in L_\mathrm{x}\}$ for some regular L_x with $|L_\mathrm{x}| = \infty$. 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 $D = E^{*,\square}$ for the RTA language $E = \{\mathtt{nil}\} \cup \{\mathtt{cons}\langle x, \mathtt{cons}\langle x, \square\rangle\rangle : x \in L_\mathrm{x}\}$, Note that $E^{k,\square}$ 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: $A = \langle \mathcal{F}, Q \cup \{q_F\}, \{q_F\}, \Delta \cup \{\mathtt{f}\langle q_1, \ldots, q_k\rangle \to q_F \mid (\mathtt{f}\langle q_1, \ldots, q_k\rangle \to q) \in \Delta, q \in Q_F\}\rangle$.

[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 $\square$ itself was interpreted rigidly. On this definition, RTAs would be closed under both pre- and post-concatenation with regular languages.

super-class of RTA.[6] Each transition rule is decorated with a set of rigid states to isolate, making it of the form $\mathtt{f}\langle q_1,\dots,q_n\rangle \xrightarrow{!I} q_0$ with $\mathtt{f}/n \in \mathcal{F}$, $\forall_i . q_i \in Q$, and $I \subseteq Q_R$.[7] Intuitively, when such a rule is used in a run to reach a node ν, the equality constraint for a rigid state $q \in I$ is no longer enforced between q-annotated nodes strictly dominated by ν and q-annotated nodes elsewhere. Every RTA is an IRTA with $I = \varnothing$ everywhere.

The non-RTA examples from before are easily captured (see Figure 1 in the appendix for illustrations). As before, suppose that L_x is recognized by the TA $A = \langle \mathcal{F}, Q, \{q_F\}, \Delta\rangle$ with non-reentrant accepting state q_F. Then taking $\mathcal{F}' = \mathcal{F} \cup \{\mathtt{p}/2, \mathtt{cons}/2, \mathtt{nil}/0\}$,

- The language P is recognized by the IRTA $\langle \mathcal{F}', Q \cup \{q^*\}, \{q^*\}, \{q_F\}, \Delta'\rangle$ with $\Delta' = \Delta \cup \{\mathtt{p}\langle q_F, q_F\rangle \xrightarrow{!\{q_F\}} q_p, \mathtt{cons}\langle q_p, q^*\rangle \to q^*, \mathtt{nil}\langle\rangle \to q^*\}$.
- The language D is recognized by the IRTA $\langle \mathcal{F}', Q \cup \{q_1^*, q_2^*\}, \{q_1^*\}, \{q_F\}, \Delta'\rangle$ with $\Delta' = \Delta \cup \{\mathtt{cons}\langle q_F, q_1^*\rangle \to q_2^*, \mathtt{cons}\langle q_F, q_2^*\rangle \xrightarrow{!\{q_F\}} q_1^*, \mathtt{nil}\langle\rangle \to q_1^*\}$.

The use of $\varnothing \subsetneq I \subsetneq Q_R$ allows for hybrid structures with both global and local equalities, such as $D' = \{[], [x_0, x_1, x_1], [x_0, x_1, x_1, x_0, x_2, x_2], \cdots \mid x_i \in L_\mathrm{x}\}$. 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 Δ (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 Δ (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_ν denote the subtree of t rooted at node ν. Each annotation of ν, rather than being a state in Q, is now a pair $(q, r) \in Q \times \wp(Q_R \times \mathcal{T}(\mathcal{F}))$. Intuitively, this pair records the existence of some run on t_ν that annotates ν with q, where $r : Q_R \rightharpoonup \{\text{subtrees of } t_\nu\}$ 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 $\mathtt{g}\langle \dots, q_0!I, \dots\rangle \to q_{-1}$, allowing the choice of I at the q_0-annotated node ν depend on the annotations at ν's siblings, and on the functor $\mathtt{g}$ and annotation q_{-1} at ν'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 ν with label g/m, if $(\mathsf{g}\langle q_1, \dots, q_m\rangle \to q) \in \Delta$ and the m children are annotated with $(q_1, r_1), \dots, (q_m, r_m)$, 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 $r_0 = \varnothing$. 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 ν with label g/m, if $(\mathsf{g}\langle q_1, \dots, q_m\rangle \xrightarrow{!I} q) \in \Delta$ and the m children are annotated with $(q_1, r_1), \dots, (q_m, r_m)$, the algorithm computes $r' = r_0 \cup \{(q', t') \in r \mid q' \notin I\}$, where $r = \bigcup_{i=1}^{m} r_i$ 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 $|Q| \cdot 2^{|Q_R|} + 1$, 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 δ and α, 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 δ, and $A = t_\alpha$ ("after") is a tree rooted at α. 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 $|Q| \cdot 2^{|Q_R|}$ only, since the shortest accepted tree cannot be taller than that—any 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 $\langle \mathcal{F}, Q, Q_F, Q_R, \Delta \rangle$ 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 Δ. 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 $O(an^2|Q_R||\Delta|)$.[8]

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 $L \cdot_{\square} M$ 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 $L^{*,\square}$ 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 α (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.

$\langle \mathcal{F}, Q, Q_F, Q_R, \Delta \rangle$, a simple recursive algorithm can nondeterministically follow transitions of Δ backwards from Q_F to find the collection Q_q of states that can appear at address α. The desired IRTA is then $\langle \mathcal{F}, Q, \bigcup_{q \in Q_F} Q_q, Q_R, \Delta \rangle$.

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 $\mathcal{F} = \{\mathtt{a}/0, \mathtt{f}/2\}$ *is* an IRTA language. Let $Q = \{q_0, q_1\}$; then B is recognized by $\langle \mathcal{F}, Q, Q, Q, \{\mathtt{a}\langle\rangle \to q_0, \mathtt{f}\langle q_0, q_0\rangle \xrightarrow{!\{q_0\}} q_1, \mathtt{f}\langle q_1, q_1\rangle \xrightarrow{!\{q_1\}} q_0\}\rangle$. 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+ / $\mathrm{TA}_=$

The IRTA class is neither more general nor more specific than tree automata with local equality constraints (TAC+ or $\mathrm{TA}_=$, [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 t in which every two subterms $\mathtt{g}\langle t_1\rangle$ and $\mathtt{g}\langle t_2\rangle$ satisfy $t_1 = t_2$, even if they are arbitrarily far apart in the tree [2, Example 3]. To show conversely that TAC+ is not included in IRTA, consider the language $L = \{[0], [1, 0], \ldots, [n, n-1, \ldots, 1, 0], \ldots\}$ (with integers represented as their Peano encodings). L is recognized by the TAC+ $\langle \{\mathtt{z}/0, \mathtt{s}/1, \mathtt{nil}/0, \mathtt{cons}/2\}, \{q_z, q_s, q_n, q_c\}, \{q_c\}, \Delta\rangle$, where $\Delta = \{\mathtt{cons}\langle q_s, q_c\rangle \xrightarrow{11=21} q_c, \mathtt{z}\langle\rangle \to q_z, \mathtt{s}\langle q_z\rangle \to q_s, \mathtt{s}\langle q_s\rangle \to q_s, \mathtt{nil}\langle\rangle \to q_n, \mathtt{cons}\langle q_z, q_n\rangle \to q_l\}$. The first rule in Δ 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 ν be a minimum-height Peano node of t such that its state annotation q_ν is reused for some ν' in t with $t_\nu \neq t_{\nu'}$. ν exists by pigeonhole. By minimality, each proper descendant of ν uses a state that annotates equal trees throughout the run on t. Substituting t_ν in for all q_ν-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.

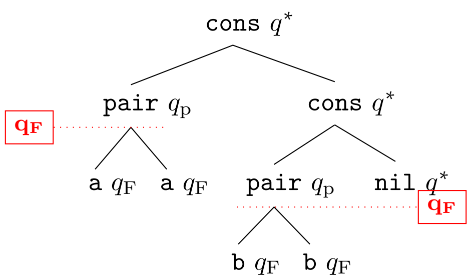
8 Conclusion

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).

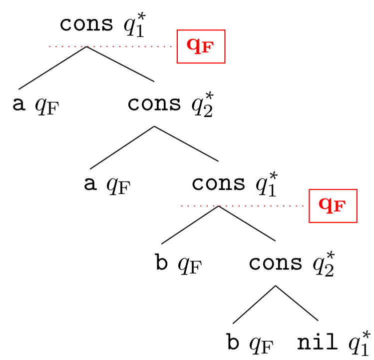
References

1. Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Loding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.*
2. Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Information and Computation*, 209(3):486–512.
3. Jocelyne Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, 1981.

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.

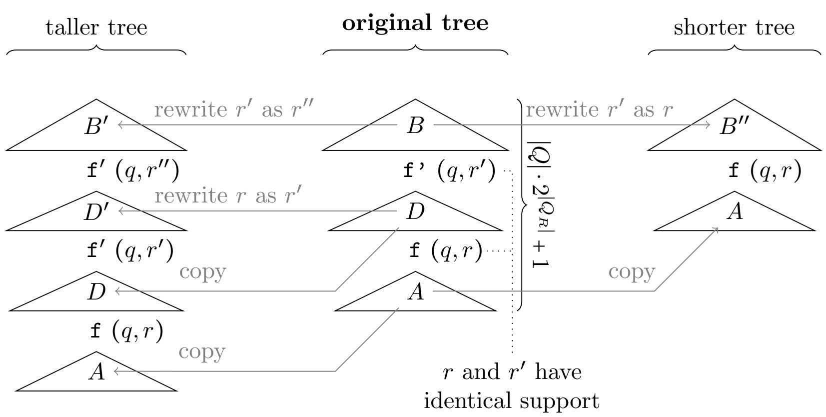


Fig. 2: Graphic depiction of the IRTA pumping construction of § 4, showing how to derive both a shorter and taller tree from a tree of height $|Q| \cdot 2^{|Q_R|} + 1$.