The Two-Counter Machine Strikes Again

Nathaniel Wesley Filardo
Johns Hopkins University
nwf@cs.jhu.edu

September 4, 2018

Abstract

In *Tree Automata Techniques and Applications* [3, Thm. 4.4.7], the emptiness test of non-deterministic Reduction Automata (RA) [1] is shown to be undecidable. The proof is a remarkable construction, encoding the evolution of a two-counter machine (2CM) [4] into a form analyzable by reduction automata. We have, in studying the myriad families of tree automata with equality constraints,[1] found this 2CM reduction to be unexpectedly useful: we exhibit a family of simple tree languages whose intersection is exactly the reduction at hand. These simpler languages are often easily found to be recognized (or not) by particular classes; when they are, the 2CM reduction implies that the class at hand has at most one of a decidable emptiness test or intersection closure. We first review the relevant machines (RA, 2CM) and then present an alternate exposition of the proof from [3], before giving our decomposed machines and concluding.

1 Review of TAC+ and Reduction Automata

Tree Automata with (Local) Equality Constraints (TAC+, also TAEC or RATEG [5]) are generalizations of regular tree automata that allow for local testing of equality between subtrees. Given a signature of symbols-and-arities $\mathcal{F}$, a TAC+ is specified by a set of states, Q, a set of final states $Q_F \subseteq Q$, and a set of transition rules, each of the form $\mathtt{f}\langle q_1, \ldots, q_n\rangle \xrightarrow{c} q_0$ with $\forall_i q_i \in Q$ and $\mathtt{f}/n \in \mathcal{F}$ (as typical of TAs in general) and c a set of local equality constraints. Such a constraint is a pair of paths. For example, the rule $\mathtt{f}\langle q_7, q_8\rangle \xrightarrow{1.1=2.3,2.1=2.2} q_5$ indicates that the state q_5 may label trees whose root node is f, whose two children are labeled by q_7 and q_8 respectively, whose first child's first child is the same tree as its second child's third child, and whose second child has equal first and second children.

Reduction Automata are a subclass of TAC+ with a particular, *lattice-order* restriction on transition rules. In particular, there must exist a lattice ordering on Q so that $(\mathtt{f}\langle q_1, \ldots, q_n\rangle \xrightarrow{c} r) \in \Delta$ implies that $\forall_i r \geq q_i$ with inequality *strict* if c contains an equality constraint. Thus, on any root-to-leaf path in an accepting run, the state annotations form a non-increasing sequence. For deterministic automata, this has the effect of tightly regulating the reach of equalities within the tree, enabling a proof of decidable emptiness testing of this class [1, §4.1]. However, the proof fails to carry over to the non-deterministic class, where the ability to annotate identical trees with different states wreaks havoc.

2 Review of Two Counter Machines

A two-counter machine can be thought of as a non-deterministic finite-state string automaton (NFA) paired with a pair of counters, just as a push-down automaton is a NFA paired with a stack. Counters are stores of arbitrary natural numbers, initialized to 0 when the automaton is started. Edges in the automaton are annotated with control directives and tests; exactly what set of control operations are available on counters varies from construction to construction, but here we use a rephrasing of Minsky's "Program machine" construction [4], which supports, for each counter, unconditional increment and decrement and a conditional test for equality with zero. An edge of the FSA annotated with such a conditional test (or its negation) can only be traversed when the indicated counter currently stores 0 (or a value other than 0, respectively).

For present purposes, we can even deprive our 2CMs of input sequences to be recognized, and focus only on the question of reachability of the (without loss of generality) sole, non-reentrant accepting state from the sole initial state with counters initialized to zero. Note that it is possible to initialize the counters to any finite state by a linear prefix of states and transitions.

[1] Some notes from our surveying efforts can be found in our nascent "Automata Zoo," available at https://github.com/nwf/autzoo, which could certainly benefit from additional eyes and effort! Some of the material here originally appeared there.

An ingenious construction shows that even deciding the recognition of the empty string, even for deterministic 2CMs, is equivalent to solving a Turing machine's halting problem [4, p. 255-258].[2] In brief, the construction relies on a series of observations:

- Counters can be zeroed by repeatedly decrementing.
- It is possible, while zeroing a counter initially holding v, to add $k*v$ to a fixed set of counters (with k fixed per target counter), by repeatedly incrementing the target counters. For example, given counters holding values $\langle 5,3,2,1,0\rangle$, we can compute $\langle 0, 3+3*5, 2+2*5, 1+0*5, 0+1*5\rangle = \langle 0, 18, 12, 1, 5\rangle$. This gives us a destructive multiply-and-add operation; with a scratch counter, the broadcast value v can be preserved.
- Similaly, while zeroing a counter holding v, one can add $\lfloor v/k \rfloor$ to a fixed set of counters and add $\mathrm{mod}(v,k)$ to another set. At the start state for this operation, if the source counter is not zero, the machine enters a chain of states which interleave decrementing the souce and testing it for zero. If this happens before k decrements have succeeded, the remainder is the number of decrements since the start state that have succeeded, which is easily tracked within the state label, and can be easily transferred to a set of counters by increment operations. Otherwise, after k decrements and a nonzero counter, increment each of the first set of target counters and re-enter the start state.
- A counter can be viewed as a stack of bits, with multiplication (by 2) and addition (of 0 or 1) forming a "push" operation and division and modulus (by 2) forming a "pop" operation. These operations are destructive, requiring separate source and target counters.
- The next ingredient is well-known: a Turing machine, with its tape, can be emulated using two stacks, one holding the tape to the left of the head and the other holding the tape to the right. Thus, we can see that *four* counters suffice to encode a Turing machine, and a moment's thought can bring that down to *three* since the two stacks can share a scratch counter. The final twists, then, are...
- Two counters can emulate any finite number of counters: encode $\langle a,b,c,\ldots\rangle$ as $2^a3^b5^c\ldots$: a product in which the i^{th} value is the exponent of the i^{th} prime (this technique is sometimes called "Gödel numbering" [2]). To increment b, multiply the counter by 3; to test it for equality to zero, divide by 3 and observe the remainder (and multiply by 3 to restore the value).
- These constructions can, of course, be combined. To push 1 to the b stack, using a as a scratch counter, decrement until zero $2^03^b5^c\ldots$ to make the other counter, initially 0, hold $2^b3^b5^c\ldots$. Then repeatedly decrement the a counter to multiply by 3 to construct $2^03^{2b}5^c\ldots$. Multiply by 3 to obtain $2^03^{2b+1}5^c\ldots$.

Putting it all together, a 2CM can (with a great many operations per emulated clock cycle) emulate a Turing machine.

3 Building Up The Tree Language

Suppose we have a 2CM over the state space P. We can encode a 2CM configuration (state $p \in P$ and counters $\in \mathbb{N}^2$) as a tree $\mathtt{m}_p\langle c_1, c_2\rangle$ where $\{\mathtt{m}_p/2 \mid p \in P\} \subset \mathcal{F}$, and c_i are Peano-encoded natural numbers built up from $\{\mathtt{z}/0, \mathtt{s}/2\} \subset \mathcal{F}$. In our automata, all Peano nodes will be annotated with one of $\{\mathrm{s},\mathrm{z}\} \subset Q$ according to the rules $\{\mathtt{z}\langle\rangle \to \mathrm{z}, \mathtt{s}\langle\mathrm{z}\rangle \to \mathrm{s}, \mathtt{s}\langle\mathrm{s}\rangle \to \mathrm{s}\}$. Within a 2CM, a transition is permitted if it is between an appropriate pair of 2CM NFA states and the counter(s) tested are (not) zero, all of which is finite; therefore, we annotate the $\mathtt{m}_p$ apex nodes with exactly this information: a $\mathtt{m}$-labeled triple of a 2CM state and two flags from $\{\mathrm{s},\mathrm{z}\}$ to indicate the counter state, so $\{\mathtt{m}\langle p,x,y\rangle \mid p \in P, x, y \in \{\mathrm{s},\mathrm{z}\}\} \subset Q$. The annotation is performed by rules of the form $\mathtt{m}_p\langle \mathrm{s},\mathrm{z}\rangle \to \mathtt{m}\langle p,\mathrm{s},\mathrm{z}\rangle$.

At this point, we can build up a TAC+ to recognize computational histories, encoded as right-branching lists of $\{\mathtt{cons}/2, \mathtt{nil}/0\} \subset \mathcal{F}$ whose elements are configurations as just described.

- We will expand the state space to include states which we will use $\mathtt{cons}/2$ nodes: $\{\mathtt{c}\langle p,x,y\rangle \mid p \in P, x, y \in \{\mathrm{s},\mathrm{z}\}\} \subset Q$.
- If $f \in P$ is the (unique, non-reentrant) final state of the 2CM in question, then all $\{\mathtt{c}\langle f,x,y\rangle \mid x, y \in \{\mathrm{s},\mathrm{z}\}\}$ are final states of the TAC+: that is, the TAC+ requires that the 2CM be in the accepting state at the top of the tree.
- We need some administrative rules to get the run started, $\{\mathtt{nil}\langle\rangle \to \mathrm{n}, \mathtt{cons}\langle\mathtt{m}\langle 0,\mathrm{z},\mathrm{z}\rangle,\mathrm{n}\rangle \to \mathtt{c}\langle 0,\mathrm{z},\mathrm{z}\rangle\}$: the rightmost configuration of the 2CM must be the initialization condition, $0 \in P$ being the initial state.

[2] The extreme "Turing tarpit" [6] nature of a 2CM means that any input or output languages we wish to talk about must be properly encoded. For example, even though reachability of the final state of a 2CM is equivalent to TM's HALT, a 2CM cannot reconfigure its counters from an input of $(n,0)$ to an output of $(2^n,0)$ [7], even though exponentiation is *certainly* a (Turing) computable function! And indeed, given the input $(2^n,0)$, there is a 2CM halting with its counters holding $(2^{2^n},0)$.

- To wire it all together, we simply need to give the remaining rules for $\texttt{cons}/2$ nodes; here, we will impose the transition semantics of the underlying 2CM. Note that on every $\texttt{cons}/2$ node has a machine configuration encoding as its left child (i.e., at path 1) and so all $\texttt{cons}/2$ other than the rightmost (whose special handling has just been given) has the previous state of the computation at path 2.1. These rules will *copy* the arguments of the $\texttt{m}\langle p, x, y\rangle$ label of the machine configuration at 1 to the $\texttt{c}\langle p, x, y\rangle$ label for the $\texttt{cons}/2$ node being labeled; will *match against* the arguments to the $\texttt{c}\langle p, x, y\rangle$ label for the for the $\texttt{cons}/2$ node at 2; will *enforce arithmetic consequences* on the two configurations (e.g., ensuring that an incremented counter is labeled $\texttt{s}$ in the later configuration); and will *enforce equalities* representing increments, decrements, or no-actions on the two counters.

An example will likely suffice, rather than a full construction. Consider a 2CM transition from state k to state m which requires that the first counter be non-zero and increments the second counter without restricting its value. The TAC+ would have two rules to encode this transition:

$$\{\texttt{cons}\langle \texttt{m}\langle m, \mathrm{s}, \mathrm{s}\rangle, \texttt{c}\langle k, \mathrm{s}, x\rangle\rangle \xrightarrow{1.1=2.1.1,1.2.1=2.1.2} \texttt{c}\langle m, \mathrm{s}, \mathrm{s}\rangle \mid x \in \{\mathrm{s}, \mathrm{z}\}\}$$

The constraint $1.1 = 2.1.1$ ensures that the first counter is the same in both configurations, and $1.2.1 = 2.1.2$ ensures that the second counter has been incremented (by ensuring that the later configuration's second counter's first child is equal to the earlier configuration's second counter). The $\{\mathrm{s}, \mathrm{z}\}$ components of the $\texttt{m}$ and $\texttt{c}$ annotations are assigned based on preconditions of the transition and consequences of arithmetic. All state transitions can be described this way, if tediously. Notably, the TAC+ so described is deterministic if the 2CM is. Figure 1 shows an schematic example of a computational history accepted by the TAC+ automaton we have described.

4 Making the Tree Friendly for Reduction Automata

It is easy to see that there is no chance of a lattice ordering on Q, in general, for the construction just described: the 2CM can cycle through states (with different counter configurations!) to cause the TAC+ to have to reuse a state annotation on the $\texttt{cons}/2$ spine. One may initially believe that there is no way out of this situation, as each component of the transition history must be related to its predecessor by local equality constraints, requiring this kind of cycles-through-constraining-rules behavior. Enter non-determinism, which will allow equal trees to be given different annotations throughout the run.

In the TAC+-based construction above, we used a simple list of $\texttt{cons}/2$ to hold the history together. Instead, we could have a *duplicating tree* of history; where a list would encode $[a, b, c, \ldots]$, we will now build up the "split-cons" structure over $\{\texttt{co}/2, \texttt{ns}/2, \texttt{nil}/0\} \subset \mathcal{F}$ shown in Figure 2. In a split-cons tree, every $\texttt{co}/2$'s right child is either a $\texttt{co}/2$ or $\texttt{nil}/0$, every $\texttt{co}/2$'s left child is a $\texttt{ns}/2$ node whose right child is equal to its parent's right child; data elements are stored at $\texttt{ns}/2$'s left child ($\texttt{co}/2$'s left child's left child).

The repetition within a "split-cons" structure *could* be enforced, by a TAC+, using equality constraints at the right spine of $\texttt{co}/2$ nodes, where $1.2 = 2$. However, the real master-stroke of the TATA proof is that this structure can also be enforced by *a (uniformly) bounded number of equalities* on any root-to-leaf path through the tree. Specifically, their construction uses a constraint $1.2 = 2$ at the root and $2.2 = 2.1.2$ at each $\texttt{ns}/2$ node whose path from the root is $2.2 \ldots 2.1$ (except the last such node). To recognize and specially label these positions within a tree, we could construct a TA thus:

- Let there be a state $q_\forall \in Q$ which accepts all trees over $\mathcal{F}$, and let $q_{\text{nil}} \in Q$ be targeted by exactly $\texttt{nil}\langle\rangle \to q_{\text{nil}}$.
- Let $q_{\text{F}} \in Q$ denote the sole accepting state of the machine and let $\texttt{co}\langle q_{\text{ra}}, q_{\text{rs}}\rangle \to q_{\text{F}}$ be the sole rule targeting it.
- Use the transition rules $\{\texttt{ns}\langle q_\forall, q_{\text{nil}}\rangle \to q_0, \texttt{co}\langle q_0, q_{\text{nil}}\rangle \to q_{\text{rs}}, \texttt{co}\langle q_{\text{ra}}, q_{\text{rs}}\rangle \to q_{\text{rs}}\}$ to label the "right spine" ($q_{\text{rs}} \in Q$) of $\texttt{co}/2$ nodes and force the left children of these nodes to be "right-adjacent" ($q_{\text{ra}} \in Q$) or the special terminating $\texttt{ns}$ node with $\texttt{nil}\langle\rangle$ right child ($q_0 \in Q$).
- Use the new states $\{q_1, q_2\} \subset Q$ to recognize suitable right-adjacent "split-cons" structure with the rules $\{\texttt{ns}\langle q_\forall, q_\forall\rangle \to q_1, \texttt{co}\langle q_1, q_\forall\rangle \to q_2, \texttt{ns}\langle q_\forall, q_2\rangle \to q_{\text{ra}}\}$.

So armed, we can build a *reduction automaton* which enforces the "split-cons" tree structure's equalities. Adding the constraint $1.2 = 2$ to the rule targeting q_{F} and the constraint $2.1.2 = 2.2$ to the rule targeting q_{ra} is sufficient. Every root-to-leaf path through the tree crosses through exactly one ro-annotated position and at most one rs-annotated position, so the lattice ordering needs only three ranks: $\{q_\forall, q_{\text{nil}}, q_0, q_1, q_2\} < \{\mathrm{ra}, \mathrm{rs}\} < \{\mathrm{F}\}$. (See the "Split-cons root" and "Split-cons right-adjacent structure" machines in Figure 3; the construction described here is their intersection.)

Within such a "split-cons" tree, the ra-annotated states have access to adjacent components of the history at positions 1 and 2.2.1 (contrast 1 and 2.1, earlier). As with TAC+ above, it is a simple matter to ensure that these positions are

related by a 2CM transition rule. In fact, we no longer need to carry the $P \times \{\mathrm{s}, \mathrm{z}\}^2$ summary of the 2CM state within the Q annotations to ensure that a tree represents a computation history; it now suffices to check that every (ra-annotated, i.e., right-adjacent) $\mathtt{ns}/2$ node has appropriately related children. However, we must still carry a single bit of state, differentiating the final state $f \in P$ from all others. This will enable the root node to enforce that the top-most entry in the computational history is an accepting configuration of the 2CM. Thus, replace the single $\mathrm{ra} \in Q$ above with a pair $\mathrm{ra}_{\mathrm{final}}$ and $\mathrm{ra}_{\mathrm{nonfinal}}$. By also changing the base-case of ra-annotations to be $\mathtt{ns}\langle \mathtt{m}_0\langle z, z\rangle, \mathtt{nil}\langle\rangle\rangle$ (with $0 \in P$ again the initial state), we continue to require that the computation start from the initial state. Thus we build a non-determinstic reduction automaton capable of recognizing only accepting execution histories of a 2CM, demonstrating undecidability of the emptiness problem for non-deterministic RAs.

5 Simple Tree Languages With Complex Intersection

The various stages of the above construction can be broken out to their own machines, each of which are straightforward and describe simple languages. The first three of these machines are exhibited in Figure 3.

5.1 End-state Machine

To check that the computational history ends with the 2CM in an accepting state, a *deterministic top-down regular* machine accepting $L = \{\mathtt{co}\langle \mathtt{ns}\langle \mathtt{m}_f\langle x_1, x_2\rangle, x_3\rangle, x_4\rangle \mid x_i \in \mathcal{T}(\mathcal{F} \setminus \{\mathtt{m}_f\})\}$ suffices. We are justified in the exclusion of $\mathtt{m}_f$ from the x_i structures by the supposition that the 2CM state $f \in P$ was non-reentrant, so it will occur nowhere else in a valid tree, even given the rampant history duplication of the split-cons structure. Presuming that the state $q_{\bar{\forall}}$ recognizes the language $\mathcal{T}(\mathcal{F} \setminus \{\mathtt{m}_f\})$, then the additional rules (over fresh q_1, q_2, q_{F}) $\{\mathtt{m}_f\langle q_{\bar{\forall}}, q_{\bar{\forall}}\rangle \to q_1, \mathtt{ns}\langle q_1, q_{\bar{\forall}}\rangle \to q_2, \mathtt{co}\langle q_2, q_{\bar{\forall}}\rangle \to q_{\mathrm{F}}\}$ suffice to define q_{F} as accepting L.

5.2 Split-Cons Tree Machines

The shape of the split-cons tree is enforcable as the intersection of (the languages described by) two non-deterministic TAC+ machines (both of which are also non-deterministic reduction automata):

- one to enforce the $1.2 = 2$ constraint at the root, i.e., the language $\{\mathtt{co}\langle \mathtt{ns}\langle x_1, x_2\rangle, x_2\rangle \mid x_i \in \mathcal{TF}\}$, and
- another to enforce the $2.1.2 = 2.2$ constraint at all (but the last) right-adjacent nodes, i.e., the inductive language $L = \{\mathtt{co}\langle \mathtt{ns}\langle x, \mathtt{nil}\langle\rangle\rangle, \mathtt{nil}\langle\rangle\rangle \mid x \in \mathcal{TF}\} \cup \{\mathtt{co}\langle t, l\rangle \mid l \in L, t \in \{\mathtt{ns}\langle x_1, \mathtt{co}\langle \mathtt{ns}\langle x_2, x_3\rangle, x_3\rangle\rangle \mid x_i \in \mathcal{TF}\}\}$.

5.3 Transition Machine

As outlined at the end of the last section, the 2CM's transitions can themselves be encoded using a TAC+ (again, also a non-deterministic reduction automaton). Letting $\rightsquigarrow$ denote the transition relation between tree-encoded 2CM configurations, we can easily build a TAC+ machine which recognizes $L_{\rightsquigarrow} \stackrel{\mathrm{def}}{=} \{\mathtt{ns}\langle m_1, \mathtt{co}\langle \mathtt{ns}\langle m_2, x_1\rangle, x_2\rangle\rangle \mid x_i \in \mathcal{TF}, m_2 \rightsquigarrow m_1\}$. To be compatible with the other machines requires constructing a TAC+ which accepts the inductive language $L = \{\mathtt{co}\langle \mathtt{ns}\langle x, \mathtt{nil}\langle\rangle\rangle, \mathtt{nil}\langle\rangle\rangle, \mid x \in \mathcal{TF}\} \cup \{\mathtt{co}\langle t, l\rangle \mid l \in L, t \in L_{\rightsquigarrow}\}$. As this machine checks only at the right-adjacent positions in the split-cons structure it is therefore are amenable to the RA condition, as each root-leaf path will transit at most one node enforcing the constraints within $L_{\rightsquigarrow}$.

6 Conclusion

We have exhibited a series of simple languages, recognized by non-deterministic TAC+ machines, whose intersection describes the language of accepting computational histories of a particular 2CM. Any class of non-deterministic tree automata with equalities (including that of non-deterministic reduction automata) that includes machines capable of recognizing these languages may have at most one of a decidable emptiness test or closure under intersection, because their emptiness of intersection problem is thus shown to be undecidable.

7 Acknowledgements

We would like to thank several anonymous TTATT reviewers for feedback on a previous version of this paper.

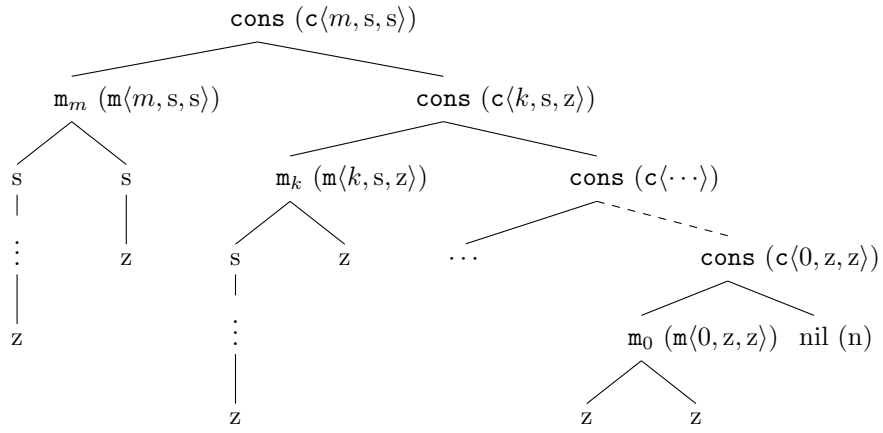

Figure 1: A computational history of a 2CM, as recognizable by a TAC+ machine. State labels are given in parentheses to the right of nodes and are as per the construction of section 3.

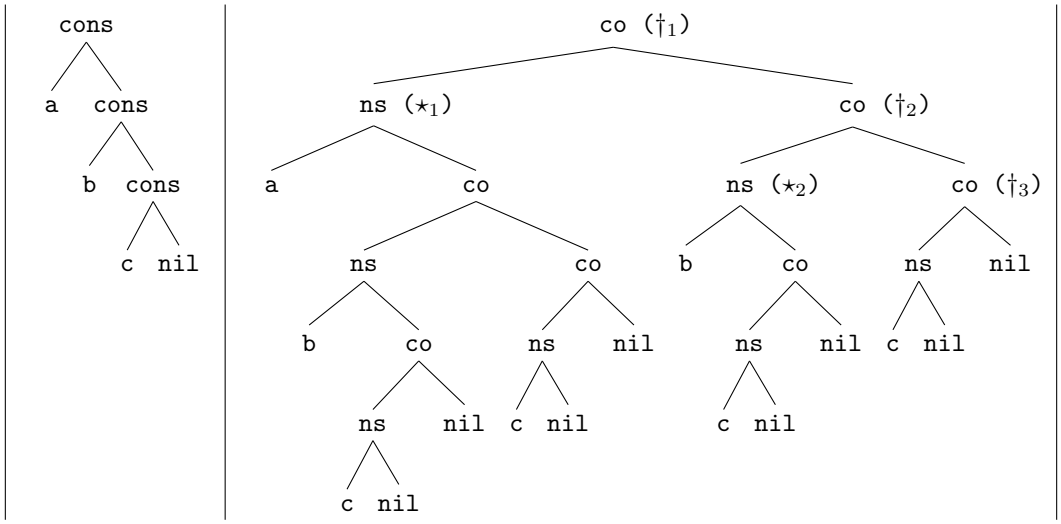

Figure 2: Comparison between a cons/2 list with its linear history and a "split-cons" tree which duplicates history. Every element doubles the size of the tree below it: there is one a in this tree, two bs, four cs, and eight nils. The shape of the tree may be enforced in two different ways: the first ensures that $1.2 = 2$ at all co/2 nodes along the right spine (marked †s), while the second ensures that $1.2 = 2$ *at the root* and $2.2 = 2.1.2$ at all "right-adjacent" ns/2 nodes (marked ⋆s). The two key observations are (1) the requisite equalities can be enforced even without constraints on the entire right spine ($\dagger_2$ is implied by $\dagger_1$ and $\star_1$, and $\dagger_3$ is implied by $\dagger_1$, $\star_1$, and $\star_2$) and (2) that any root-leaf path will go through the root ($\dagger_1$) and at most one $\star_i$-labeled node. To verify that $\dagger_3$ is indeed implied, one can readily verify the chain of equalities through all eight nil/0 nodes as named from the root: $2.2.2 \overset{\dagger_1}{=} 1.2.2.2 \overset{\star_1}{=} 1.2.1.2.2 \overset{\dagger_1}{=} 2.1.2.2 \overset{\star_2}{=} 2.1.2.1.2 \overset{\dagger_1}{=} 1.2.1.2.1.2 \overset{\star_1}{=} 1.2.2.1.2 \overset{\dagger_1}{=} 2.2.1.2$. Note that this contains two applications of the proof of $\dagger_2$'s derivation: $2.2 \overset{\dagger_1}{=} 1.2.2 \overset{\star_1}{=} 1.2.1.2 \overset{\dagger_1}{=} 2.1.2$. This recursive structure for implied equality constraints continues for deeper elements in taller split-cons structures.

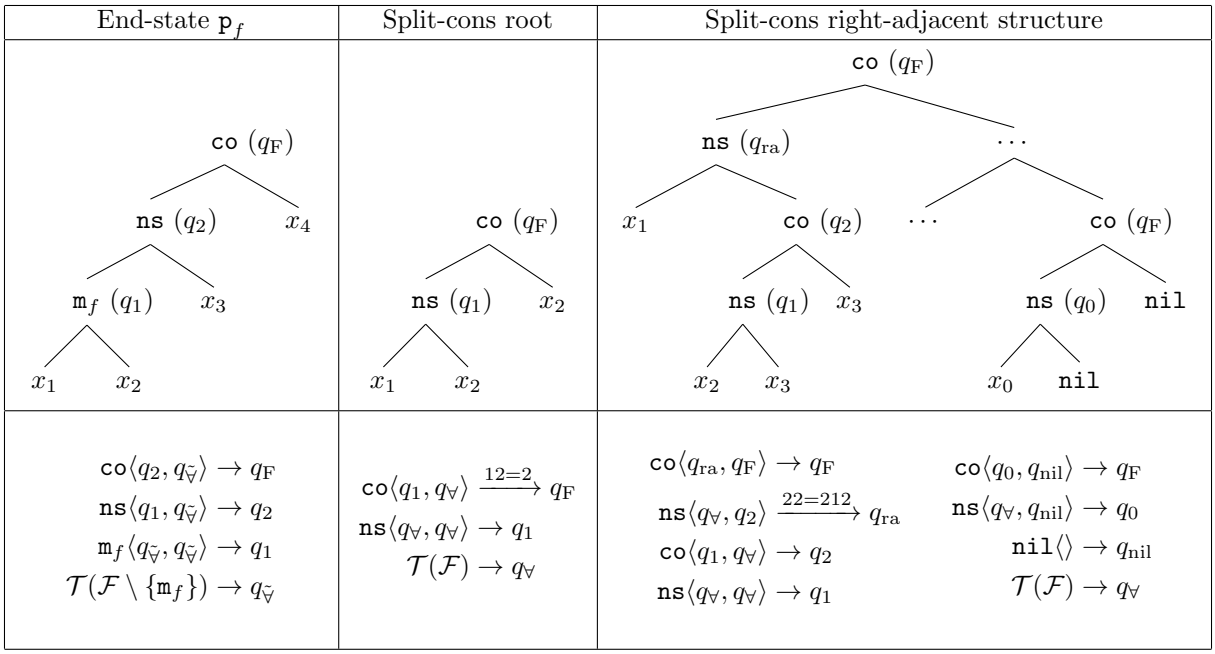

Figure 3: Schematics and machines for 2CM-by-intersection construction. All x_i trees are analyzed as $q_{\tilde\forall}$ or $q_\forall$, as appropriate; equal subscripts indicate equal trees enforced by the machines' equality constraints.

References

[1] Max Dauchet, Anne-Cécile Caron, and Jean-Luc Coquidé. Automata for reduction properties solving. 20(2):215–233.

[2] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[3] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Loding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.*

[4] Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[5] Jocelyne Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, 1981.

[6] Alan J. Perlis. Special feature: Epigrams on programming. *SIGPLAN Not.*, 17(9):7–13, September 1982.

[7] Rich Schroeppel. A two counter machine cannot calculate 2^n. Technical Report 257, MIT, May 1972.