Safraless Compositional Synthesis*

Orna Kupferman[1], Nir Piterman[2], and Moshe Y. Vardi[3]

[1] Hebrew University
[2] Ecole Polytechnique Fédéral de Lausanne (EPFL)
[3] Rice University and Microsoft Research

Abstract. In automated synthesis, we transform a specification into a system that is guaranteed to satisfy the specification. In spite of the rich theory developed for system synthesis, little of this theory has been reduced to practice. This is in contrast with of model-checking theory, which has led to industrial development and use of formal verification tools. We see two main reasons for the lack of practical impact of synthesis. The first is algorithmic: synthesis involves Safra's determinization of automata on infinite words, and a solution of parity games with highly complex state spaces; both problems have been notoriously resistant to efficient implementation. The second is methodological: current theory of synthesis assumes a single comprehensive specification. In practice, however, the specification is composed of a set of properties, which is typically evolving – properties may be added, deleted, or modified.

In this work we address both issues. We extend the Safraless synthesis algorithm of Kupferman and Vardi so that it handles LTL formulas by translating them to nondeterministic generalized Büchi automata. This leads to an exponential improvement in the complexity of the algorithm. Technically, our algorithm reduces the synthesis problem to the emptiness problem of a nondeterministic Büchi tree automaton $\mathcal{A}$. The generation of $\mathcal{A}$ avoids determinization, avoids the parity acceptance condition, and is based on an analysis of runs of universal generalized co-Büchi tree automata. The clean and simple structure of $\mathcal{A}$ enables optimizations and a symbolic implementation. In addition, it makes it possible to use information gathered during the synthesis process of properties in the process of synthesizing their conjunction.

1 Introduction

One of the most significant developments in the area of program verification over the last two decades has been the development of of algorithmic methods for verifying temporal specifications of *finite-state* programs; see [5]. A frequent criticism against this approach, however, is that verification is done *after* significant resources have already been invested in the development of the program. Since programs invariably contain errors, verification simply becomes part of the debugging process. The critics argue that the desired goal is to use the specification in the program development process in order to guarantee the design of correct programs. This is called *program synthesis*.

* A full version can be downloaded from www.cs.huji.ac.il/∼ornak/cav06.pdf. The references to the appendix in this extended abstract refer to this version.

In the late 1980s, several researchers realized that the classical approach to program synthesis, where a program is extracted from a proof that the specification is satisfiable, is well suited to *closed* systems, but not to *open* (also called *reactive*) systems [1, 6, 21]. In reactive systems, the program interacts with the environment, and a correct program should then satisfy the specification with respect to all environments. These researchers argued that the right way to approach synthesis of reactive systems is to consider the situation as a (possibly infinite) game between the environment and the program. A correct program can be then viewed as a winning strategy in this game. It turns out that satisfiability of the specification is not sufficient to guarantee the existence of such a strategy. Abadi et al. called specifications for which a winning strategy exists *realizable*. Thus, a strategy for a program with inputs in I and outputs in O maps finite sequences of inputs (words in $(2^I)^*$ – the actions of the environment so far) to an output in 2^O – a suggested action for the program. A strategy can then be viewed as a labeling of a tree with directions in 2^I by labels in 2^O. The traditional algorithm for finding a winning strategy transforms the specification into a parity automaton over such trees such that a program is realizable precisely when this tree automaton is nonempty, i.e., it accepts some infinite tree [21]. A finite generator of an infinite tree accepted by this automaton can be viewed as a finite-state program realizing the specification. This is closely related to the approach taken, e.g., in [23], to solve Church's *solvability problem* [4]. Several works during the 1990s showed how this approach to program synthesis can be carried out in a variety of settings.

In spite of the rich theory developed for program synthesis, little of this theory has been reduced to practice. Some people argue that this is because the realizability problem for linear-temporal logic (LTL) specifications is 2EXPTIME-complete [21, 24], but this argument is not compelling. First, experience with verification shows that even nonelementary algorithms can be practical, since the worst-case complexity does not arise often (cf., the model-checking tool MONA [7]). Furthermore, in some sense, synthesis is not harder than verification. This may seem to contradict the known fact that while verification is "easy" (linear in the size of the model and at most exponential in the size of the specification [16]), synthesis is hard (2EXPTIME-complete). There is, however, something misleading in this fact: while the complexity of synthesis is given with respect to the specification only, the complexity of verification is given with respect to the specification and the program, which can be much larger than the specification. In particular, it is shown in [24] that there are temporal specifications for which every realizing program must be at least doubly exponentially larger than the specifications. Clearly, the verification of such programs is doubly exponential in the specification, just as the cost of synthesis.

We believe that there are two reasons for the lack of practical impact of synthesis theory. The first is algorithmic and the second is methodological. Consider first the algorithmic problem. First, constructing tree automata for realizing strategies uses Safra's construction for determinizing Büchi automata. This construction has been notoriously resistant to efficient implementations [2, 27] (An alternative construction is equally hard [2].) Second, Safra's determinization results in automata with a very complicated state space. The best-known algorithms for parity-tree-automata emptiness [13] are nontrivial already when applied to simple state spaces. Implementing them on top

of the messy state space that results from Safra's determinization is awfully complex, and is not amenable to optimizations and a symbolic implementation.

Another major issue is methodological. The current theory of program synthesis assumes that one gets a comprehensive set of temporal assertions as a starting point. This cannot be realistic in practice. A more realistic approach would be to assume an *evolving* formal specification: temporal assertions can be added, deleted, or modified. Since it is rare to have a complete set of assertions at the very start of the design process, there is a need to develop *compositional* synthesis algorithms. Such algorithms can, for example, refine designs when provided with additional temporal properties.

In this paper we address both issues. We focus on the case where forbidden behaviors are described by nondeterministic generalized Büchi automata on infinite words, which are Büchi automata with multiple acceptance sets (corresponding to the *impartiality* fairness condition of [17]). Our interest in specifying forbidden behaviors and in using the generalized Büchi condition is motivated by the fact that LTL formulas (and their negation) can be conveniently translated to nondeterministic generalized Büchi automata [10]. Equivalently, one can specify allowed behavior by universal generalized co-Büchi automata. Following [15], we offer an alternative to the standard automata-theoretic approach. The crux of our approach is avoiding the use of Safra's construction and of nondeterministic parity tree automata. In the approach described here, one checks whether the specification ψ is realizable using the following steps: (1) construct a universal generalized co-Büchi tree automaton $\mathcal{A}_\psi$ that accepts all realizing strategies for ψ, (2) reduce[4] $\mathcal{A}_\psi$ to an alternating weak tree automaton $\mathcal{A}^w_\psi$, (3) translate $\mathcal{A}^w_\psi$ to a nondeterministic Büchi tree automaton $\mathcal{A}^n_\psi$, and (4) check that the language of $\mathcal{A}^n_\psi$ is nonempty. The key is avoiding Safra's construction, by using universal generalized co-Büchi automata instead of deterministic parity automata.[5]

The difference between our approach here and the approach in [15] is that here we use *generalized* co-Büchi automata, unlike the co-Büchi automata used there. This leads to an exponential improvement in the complexity of our algorithm, as we describe below. Extending the framework of [15] to generalized co-Büchi automata requires two key technical steps. First, as our Safraless approach used a "Safraful" bound on the size of the realizing strategies, we need to extend Safra's construction to nondeterministic generalized Büchi automata, obtaining an exponential improvement (with respect to an approach that first translates the generalized Büchi automaton to a Büchi automaton) in that construction. Second, we need to show how the co-Büchi ranks devised in [14] for the analysis of runs of universal automata on words can be applied to the analysis of runs of universal automata on finitely generated trees.

[4] We use "reduce A_1 to A_2", rather than "translate A_1 to A_2" to indicate that A_1 need not be equivalent to A_2, yet the language of $\mathcal{A}_1$ is empty iff the language of $\mathcal{A}_2$ is empty.

[5] A note to readers who are discouraged by the fact our method goes via several intermediate automata: it is possible to combine the reductions into one construction, and in fact we describe here also a direct translation of universal generalized co-Büchi automata into nondeterministic Büchi automata. In practice, however, it is beneficial to have many intermediate automata, as each intermediate automaton undergoes optimization constructions that are suitable for its particular type, cf. [12].

Beyond the improvement in complexity, the advantage of the Safraless approach is that we get tree automata with cleanly described state spaces, which enables the application of symbolic algorithms for Büchi tree automata emptiness. Further, we can now obtain a *compositional* algorithm. Given a specification ψ, we first check its realizability. Suppose now that we get an additional specification ψ'. We can, of course, simply check the realizability of $\psi \wedge \psi'$ from scratch. Instead, we suggest to first check also the realizability of ψ'. We then show how, thanks to the simple structure of the tree automata, much of the work used in checking the realizability of ψ and ψ' in isolation can be reused in checking the realizability of $\psi \wedge \psi'$. The compositional algorithm we suggest can be combined with an *incremental* algorithm, in which we iteratively increase the bound on the size of the realizing strategy. In addition, we explain how it can be implemented symbolically.

2 Preliminaries

We assume familiarity with the basic notions of alternating automata on infinite trees, cf. [11]. We include basic definitions in Appendix A.

Given an alphabet Σ and a set D of directions, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$, where $T \subseteq D^*$ is a tree over D and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. A *transducer* is a labeled finite graph with a designated start node, where the edges are labeled by D and the nodes are labeled by Σ. A Σ-labeled D-tree is *regular* if it is the unwinding of some transducer. More formally, a transducer is a tuple $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$, where D is a finite set of directions, Σ is a finite alphabet, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times D \to S$ is a deterministic transition function, and $L : S \to \Sigma$ is a labeling function. We define $\eta : D^* \to S$ in the standard way: $\eta(\varepsilon) = s_{in}$, and for $x \in D^*$ and $d \in D$, we have $\eta(x \cdot d) = \eta(\eta(x), d)$. Intuitively, A Σ-labeled D-tree $\langle D^*, \tau \rangle$ is regular if there exists a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ such that for every $x \in D^*$, we have $\tau(x) = L(\eta(x))$. We then say that the size of the regular tree $\langle D^*, \tau \rangle$, denoted $\|\tau\|$, is $|S|$, the number of states of $\mathcal{T}$.

We denote an alternating tree automaton by a tuple $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition A run of $\mathcal{A}$ is accepting if all its infinite paths satisfy the acceptance condition. For a path π, we denote the set of automaton states visited infinitely often along this path by $inf(\pi)$. exactly all the states that appear infinitely often in π. We consider here four acceptance conditions defined as follows[6].

- A path π satisfies a *generalized Büchi* acceptance condition $\alpha = \{F_1, F_2, \ldots, F_k\} \subseteq 2^Q$ iff for all $1 \leq i \leq k$ we have $inf(\pi) \cap F_i \neq \emptyset$. The number k of sets in α is called the *index* of the automaton. If $|\alpha| = 1$ we call α a *Büchi* condition.
- A path π satisfies a *generalized co-Büchi* acceptance condition $\alpha = \{F_1, F_2, \ldots, F_k\} \subseteq 2^Q$ iff for some $1 \leq i \leq k$ such that $inf(\pi) \cap F_i = \emptyset$. The number k of sets in α is called the *index* of the automaton. If $|\alpha| = 1$ we call α a *co-Büchi* condition.

[6] We also refer to the Streett condition, but its definition is not important here.

– A path π satisfies a *Rabin* acceptance condition $\alpha = \{\langle L_1, U_1\rangle, \ldots, \langle L_k, U_k\rangle\}$ with $L_i, U_i \subseteq Q$ for all $1 \leq i \leq k$ iff for some $1 \leq i \leq k$ for which $inf(\pi) \cap L_i \neq \emptyset$ and $inf(\pi) \cap U_i = \emptyset$.

For the three conditions, an automaton accepts a tree iff there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts. We also refer to a fourth condition, which is a special case of the Büchi condition, and is referred to as the *weak* condition [20].

Below we discuss some special cases of alternating automata. The alternating automaton $\mathcal{A}$ is *nondeterministic* if for all the formulas that appear in δ, if (c_1, q_1) and (c_2, q_2) are conjunctively related, then $c_1 \neq c_2$. (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of $\{c\} \times Q$, for each $c \in D$, in each disjunct). The automaton $\mathcal{A}$ is *universal* if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$, and $\mathcal{A}$ is *deterministic* if it is both nondeterministic and universal. The automaton $\mathcal{A}$ is a *word* automaton if $|D| = 1$. Then, we can omit D from the specification of the automaton and denote the transition function of $\mathcal{A}$ as $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$. If the word automaton is nondeterministic or universal, then $\delta : Q \times \Sigma \to 2^Q$.

We denote each of the different types of automata by an acronym in $\{D, N, U, A\} \times \{B, GB, C, GC, R\} \times \{W, T\}$, where the first letter describes the branching mode of the automaton (deterministic, nondeterministic, universal, or alternating), the second letter describes the acceptance condition (Büchi, generalized Büchi, co-Büchi, generalized co-Büchi, or Rabin), and the third letter describes the object over which the automaton runs (words or trees). For example, ART are alternating Rabin tree automata and UGCT are universal generalized co-Büchi tree automata.

3 Synthesis

Consider an UGCW $\mathcal{S}$ over the alphabet $2^{I \cup O}$, for sets I and O of input and output signals. The *realizability problem* for $\mathcal{S}$ [21] is to decide whether there is a *strategy* $f : (2^I)^* \to 2^O$, generated by a transducer[7] such that all the computations of the system generated by f are in $L(\mathcal{S})$. We call such a strategy, a *good* strategy. A computation $\rho \in (2^{I \cup O})^\omega$ is *generated by* f if $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots$ and for all $j \geq 1$, we have $o_j = f(i_0 \cdot i_1 \cdots i_{j-1})$.

In practice, the UGCW $\mathcal{S}$ originates from an LTL formula ψ that specifies the desired properties of the program we synthesize. In order to get $\mathcal{S}$, we first translate $\neg\psi$ to an NGBW $\mathcal{A}_{\neg\psi}$, and then dualize $\mathcal{A}_{\neg\psi}$ by viewing it as a UGCW. By [29, 10], $\mathcal{A}_{\neg\psi}$, and thus also $\mathcal{S}$, have $2^{O(|\psi|)}$ states and index $O(|\psi|)$. Alternatively, one can define properties directly using UGCW, as done, for example, in the framework of Generalized Symbolic Trajectory Evaluation [30], by means of *fair assertion graphs*.

Theorem 1. *The realizability problem for a given UGCW can be reduced to the nonemptiness problem of a UGCT with the same state space and index.*

[7] It is known that if some transducer that generates f exists, then there is also a finite-state transducer.

Proof: A strategy $f : (2^I)^* \to 2^O$ can be viewed as a 2^O-labeled 2^I-tree. Given S, we define a UGCT $\mathcal{S}'$ such that $\mathcal{S}'$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau\rangle$ iff τ is a good strategy for $\mathcal{S}$.

Let $\mathcal{S} = \langle 2^{I\cup O}, Q, q_{in}, \delta, \alpha\rangle$. Then, $\mathcal{S}' = \langle 2^O, 2^I, Q, q_{in}, \delta', \alpha\rangle$, where for every $q \in Q$ and $o \in 2^O$, we have $\delta'(q, o) = \bigwedge_{i\in 2^I} \bigwedge_{q'\in\delta(q,i\cup o)}(i, q')$. Thus, from state q, reading the output assignment $o \in 2^O$, the automaton $\mathcal{S}'$ branches to each direction $i \in 2^I$, with all the states q' to which δ branches when it reads $i \cup o$ in state q. It is not hard to see that $\mathcal{S}'$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau\rangle$ iff for all the paths $\{\varepsilon, i_0, i_0 \cdot i_1, i_0 \cdot i_1 \cdot i_2, \ldots\}$ of T, the infinite word $(i_0 \cup \tau(\varepsilon)), (i_1 \cup \tau(i_0)), (i_2 \cup \tau(i_0 \cdot i_1)), \ldots$ is accepted by the UGCW $\mathcal{S}$ as required. □

We now describe an emptiness preserving translation of UGCT to NBT. The correctness proof of the construction is given in Sections 4.1 and 4.2. There, we also suggest to use ABT as an intermediate step in the construction. While this adds a step to our chain of reductions, it enables further optimizations of the result.

For an integer c, let $[c]$ denote the set $\{0, 1, \ldots, c\}$, and let $[c]^{odd}$ and $[c]^{even}$ denote the set of odd and even members of $[c]$, respectively. Also, let $R_k(c) = [2c]^{even} \cup ([2c]^{odd} \times \{1, \ldots, k\})$, and $\leq$ be the lexicographical order on the elements of $R_k(c)$. We refer to the members of $R_k(c)$ in $[2c]^{even}$ as *even ranks* and refer to the members of $R_k(c)$ in $[2c]^{odd} \times \{j\}$ as *odd ranks with index* j. Note that the size of $R_k(c)$ is $c(k+1)$. Our construction refers to a function $Safra(n, k)$, which, as we show later, is bounded from above by $12^n n^{2n} k^n$.

Theorem 2. *Let $\mathcal{A}$ be a UGCT with n states and index k. There is an NBT $\mathcal{A}'$ over the same alphabet such that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n^2(\log n+\log k))}$.*

Proof: Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \{F_1, \ldots, F_k\}\rangle$, and let $c = Safra(n, k)n!n^3$. Note that c is $2^{O(n(\log n+\log k))}$. Let $\mathcal{R}_k(c)$ be the set of functions $f : Q \to R_k(c)$ in which $f(q)$, for all $q \in F_j$, is not odd with index j. For $g \in \mathcal{R}_k(c)$, let $odd(g) = \{q : g(q) \text{ is odd}\}$. We define $\mathcal{A}' = \langle \Sigma, D, Q', q'_{in}, \delta', \alpha'\rangle$, where

- $Q' = 3^Q \times \mathcal{R}_k(c)$. For technical convenience, we refer to the states of Q' as triples $\langle S, O, f\rangle$ with $O \subseteq S \subseteq Q$ and $f \in \mathcal{R}_k(c)$.
- $q'_{in} = \langle\{q_{in}\}, \emptyset, g_0\rangle$, where g_0 maps all states to $2c$.
- For $q \in Q$, $\sigma \in \Sigma$, and $d \in D$, let $\delta(q, \sigma, d) = \delta(q, \sigma) \cap (\{d\} \times Q)$. For two functions g and g' in $\mathcal{R}_k(c)$, a letter σ, and direction $d \in D$, we say that g' *covers* $\langle g, \sigma, d\rangle$ if for all q and q' in Q, if $q' \in \delta(q, \sigma, d)$, then $g'(q') \leq g(q)$. Then, for all $\langle S, O, g\rangle \in Q'$ and $\sigma \in \Sigma$, we define δ as follows.
  - If $O \neq \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma) = \bigwedge_{d\in D} \bigvee_{g_d \text{ covers } \langle g,\sigma,d\rangle} \langle \delta(S, \sigma, d), \delta(O, \sigma, d) \setminus odd(g_d), g_d\rangle$.
  - If $O = \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma) = \bigwedge_{d\in D} \bigvee_{g_d \text{ covers } \langle g,\sigma,d\rangle} \langle \delta(S, \sigma, d), \delta(S, \sigma, c) \setminus odd(g_d), g_d\rangle$.
- $\alpha' = 2^Q \times \{\emptyset\} \times \mathcal{R}_k(c)$.

□

Corollary 1. *The realizability problem for an NGBW with n states and index k can be reduced to the nonemptiness problem of an NBT with $2^{O(n^2(\log n + \log k))}$ states.*

These bounds are exponentially better than those established in [15]. There, the NGBW is converted to an NBW with nk states and the overall resulting complexity is $2^{O((nk)^2(\log k + \log n))}$.

The *synthesis problem* for $\mathcal{S}$ is to find a transducer that generates a strategy realizing $\mathcal{S}$. Known algorithms for the nonemptiness problem can be easily extended to return a transducer [22]. The algorithm we present here also enjoys this property, thus it can be used to solve not only the realizability problem but also the synthesis problem. (For a comparison of the Safraless and the Safraful approaches to synthesis from a the perspective of program size, see [15].)

4 From UGCT to NBT

Recall that runs of alternating tree automata are labeled trees. By merging nodes that are roots of identical subtrees, it is possible to maintain runs in graphs. In Section 4.2, we prove a bounded-size run graph property for UGCT. In Section 4.2, we show how the bounded-size property enables a simple translation of UGCT to ABT, which we then translate to an NBT. Combining the translations results in the UGCT to NBT construction described in Theorem 2. While our construction avoids Safra's determinization construction, the proof of the bounded-size run-graph property makes use of the bound the construction provides to the blow up involved in determinization. Since we handle the generalized co-Büchi construction, we need a bound on the blow involved in the determinization of NGBW. We provide such a bound in Section 4.1.

4.1 NGBW to DRW

There are two known approaches to determinization of NGBW. The first is to convert the NGBW to an NBW [3] and then use Safra's determinization [25]. The second is to view the NGBW as a Streett automaton and apply Safra's determinization of Streett automata [26]. Both approaches produce automata with $(nk)^{O(nk)}$ states. In this section we show how to extend Safra's determinization construction for the case of generalized Büchi automata. Our construction below produces a DRW with $(nk)^{O(n)}$ states, exponentially fewer states than the approaches described.

We offer here a succinct description of the improvement. The key is to augment Safra trees with an indexing function. In Safra's constructions, the DRW refers to a visit in the set of accepting states as a good event. In our extension, a good event occurs only after visits to all the sets in the generalized Büchi condition. Thus, the idea is similar to the indexing used in the translation of NGBW to NBW [10], but the challenge is to combine this indexing in the state space of the DRW in a way that minimizes the blow up in terms of k. A correctness proof is provided in Appendix B. There, we also describe Safra's determinization construction for NBW. Note that the improved construction is used only to generate the improved bound. The synthesis algorithm uses this bound but it does *not* use Safra's construction.

Theorem 3. *Given an NGBW with n states and index k, we can construct an equivalent DRW with at most $12^n n^{2n} k^n$ states and n pairs.*

Proof: Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ be an NGBW with $|S| = n$ and $\alpha = \{F_1, \ldots, F_k\}$. Let $V = [n]$ and $V' = \{n+1, \ldots, 2n\}$. We construct the DRW $\mathcal{D}$ equivalent to $\mathcal{N}$. Let $\mathcal{D} = \langle \Sigma, D, \rho, d_0, \alpha \rangle$, where the components of $\mathcal{D}$ are as follows.

- A *generalized Safra tree* t is $\langle N, 1, p, \psi, l, h, R, G \rangle$ where $N \subseteq V$ is a set of nodes, $1 \in N$ is the root node, $p : N \to N$ is the parenthood function, $\psi : N \to N$ is a partial order defining "older than" on siblings, $l : N \to 2^S$ is a labeling of the nodes with subsets of S, $h : N \to [k]$ is an indexing function associating with every node an index in $[k]$, and R and G are two disjoint subsets of V. In addition the label of every node is a proper superset of the union of the labels of its children. The labels of two siblings are disjoint. The set D of states is the set of *generalized Safra trees* over S and k.
- $d_0 \in D$ has a unique node 1 where $l(1)=\{s_0\}$, $h(1)=1$, $R=V-\{1\}$, and $G=\emptyset$.
- The Rabin acceptance condition is $\alpha=\{\langle L_1, U_1 \rangle, \ldots, \langle L_n, U_n \rangle\}$ where $L_i = \{d \in D \mid i \in G_d\}$ and $U_i = \{d \in D \mid i \in R_d\}$.
- For every tree $d \in D$ and letter $\sigma \in \Sigma$ the transition $d' = \rho(d, \sigma)$ is the result of the following transformations on d. We use temporarily the set of nodes V'. (1) For every node v with label S' replace S' by $\delta(S', \sigma)$ and set R and G to the empty set. (2) For every node v with label S' such that $h(v) = i$ and $S' \cap F_i \neq \emptyset$, create a new youngest child $v' \in V'$. Set its label to $S' \cap F_i$ and its index to 1. (3) For every node v with label S' and state $s \in S'$ such that s belongs also to an older sibling v' of v, remove s from the label of v and all its descendants. (4) For every node v whose label is equal to the union of the labels of its children, remove all descendants of v. If $h(v) = k$, change $h(v)$ to 1 and add v to G. If $h(v) < k$, increase $h(v)$ by one. (5) Remove all nodes with empty labels and add all unused names to R. (6) Change nodes in V' to nodes in V.

□

Let $Safra(n, k)$ be the number of generalized Safra trees for NGBW with n states and index k. By the above theorem, $Safra(n, k)$ is bounded from above by $12^n n^{2n} k^n$.

4.2 From UGCT to NBT

A bounded-size run graph property for UGCT Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$ be a UGCT with $\alpha = \{F_1, \ldots, F_k\}$. Recall that a run $\langle T_r, r \rangle$ of $\mathcal{A}$ on a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled tree in which a node y with $r(y) = \langle x, q \rangle$ stands for a copy of $\mathcal{A}$ that visits the state q when it reads the node x. Assume that $\langle T, \tau \rangle$ is regular, and is generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$. For two nodes y_1 and y_2 in T_r, with $r(y_1) = \langle x_1, q_1 \rangle$ and $r(y_2) = \langle x_2, q_2 \rangle$, we say that y_1 and y_2 are *similar* iff $q_1 = q_2$ and $\eta(x_1) = \eta(x_2)$. By merging similar nodes into a single vertex, we can represent the run $\langle T_r, r \rangle$ by a finite graph $G = \langle V, E \rangle$, where $V = S \times Q$ and $E(\langle s, q \rangle, \langle s', q' \rangle)$ iff there is $c \in D$ such that $(c, q') \in \delta(q, L(s))$ and $\eta(s, c) = s'$. We restrict G to vertices reachable from the vertex $\langle s_{in}, q_{in} \rangle$. We refer to G as the *run graph of $\mathcal{A}$ on $\mathcal{T}$*. A run graph of $\mathcal{A}$ is then a run graph of $\mathcal{A}$ on some transducer $\mathcal{T}$. We say that G is accepting iff

every infinite path of G has only finitely many F_j-vertices (vertices in $S \times F_j$), for some $1 \leq j \leq k$. Since $\mathcal{A}$ is universal and $\mathcal{T}$ is deterministic, the run $\langle T_r, r\rangle$ is *memoryless* in the sense that the merging does not introduce to G paths that do not exist in $\langle T_r, r\rangle$, and thus, it preserves acceptance. Formally, we have the following:

Lemma 1. *Consider a UGCT $\mathcal{A}$. Let $\langle T, \tau\rangle$ be a tree generated by a transducer $\mathcal{T}$. The run tree $\langle T_r, r\rangle$ of $\mathcal{A}$ on $\langle T, \tau\rangle$ is accepting iff the run graph G of $\mathcal{A}$ on $\mathcal{T}$ is accepting.*

Note that G is finite, and its size is bounded by $S \times Q$. We now bound S and get a bounded-size run-graph property for UGCT. The bound on S depends on the blow-up involved in NGBW determinization, which we studied in Section 4.1. Essentially, the bound depends on the side of an NRT equivalent to the UGCT, and in order to get such an NRT we have to determinize an NGBW that accepts bad paths in runs of the UGCT.

Theorem 4. *A UGCT $\mathcal{A}$ with n states and index k is not empty iff $\mathcal{A}$ has an accepting run graph with at most $Safra(n,k)n!n^3$ vertices.*

From UGCT to NBT via ABT Consider a graph $G' \subseteq G$. We say that a vertex $\langle s, q\rangle$ is *finite* in G' iff all the paths that start at $\langle s, q\rangle$ are finite. For $1 \leq j \leq k$, we say that a vertex $\langle s, q\rangle$ is F_j*-free* in G' iff all the vertices in G' that are reachable from $\langle s, q\rangle$ are not F_j-vertices. Note that, in particular, an F_j-free vertex is not an F_j-vertex.

Given a run $\langle T_r, r\rangle$, we define an infinite sequence of graphs $G_0 \supseteq G^1_1 \supseteq G^2_1 \supseteq \ldots G^k_1 \supseteq G^{k+1}_1 \supseteq G^1_3 \supseteq \ldots G^{k+1}_3 \supseteq G^1_5 \ldots$ as follows. To simplify notations, we sometimes refer to G^1_{2i+1} as G_{2i+1} and to G^{k+1}_{2i+1} as G_{2i+2}. Thus, $G_1 = G^1_1$, $G_2 = G^{k+1}_1$, $G_3 = G^1_3$, $G_4 = G^{k+1}_3$, and so on.

- $G_0 = G$.
- $G^1_{2i+1} = G_{2i} \setminus \{\langle s, q\rangle \mid \langle s, q\rangle \text{ is finite in } G_{2i}\}$.
- $G^{j+1}_{2i+1} = G^j_{2i+1} \setminus \{\langle s, q\rangle \mid \langle s, q\rangle \text{ is } F_j\text{-free in } G^j_{2i+1}\}$, for $1 \leq j \leq k$.

Lemma 2. *A run graph $G = \langle V, E\rangle$ is accepting iff there is $i \leq |V|$ for which G_{2i} is empty.*

Let G be an accepting run graph. Given a vertex $\langle s, q\rangle$ in G, the *rank* of $\langle s, q\rangle$, denoted $rank(s, q)$, is defined as follows:

$$rank(s,q) = \begin{cases} 2i & \text{If } \langle s, q\rangle \text{ is finite in } G_{2i}. \\ \langle 2i+1, j\rangle & \text{If } \langle s, q\rangle \text{ is } F_j\text{-free in } G^j_{2i+1}. \end{cases}$$

Recall that, for an integer c, we have defined $R_k(c) = [2c]^{even} \cup ([2c]^{odd} \times \{1, \ldots, k\})$, as a set of $c(k+1)$ ranks, and defined $\leq$ as the lexicographical order on the elements of $R_k(c)$. For an odd rank $\rho = \langle 2i+1, j\rangle$, we refer to G^j_{2i+1} as G_ρ. Let $c = |V|$. By Lemma 2, there is $i \leq c$ for which G_{2i} is empty. Therefore, every vertex gets a well-defined rank in $R_k(c)$.

Lemma 3. *In every infinite path in an accepting run graph G, there exists a vertex $\langle s, q\rangle$ with an odd rank such that all the vertices $\langle s', q'\rangle$ on the path that are reachable from $\langle s, q\rangle$ have $rank(s', q') = rank(s, q)$.*

We can now use the analysis of ranks in order to translate UGCT to NBT. In order to enable further optimizations, we use ABT as an intermediate step in the construction.

Theorem 5. *Let $\mathcal{A}$ be a UGCT with n states and index k. There is an ABT $\mathcal{A}'$ over the same alphabet such that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n(\log n + \log k))}$.*

As detailed in the proof of the Theorem, the ABT $\mathcal{A}'$ accepts all the regular trees $\langle T, \tau \rangle \in \mathcal{L}(\mathcal{A})$ that are generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ with at most $Safra(n,k)n!n^2$ states. Note that the run graph of $\mathcal{A}$ on such $\langle T, \tau \rangle$ is accepting and is of size most $Safra(n,k)n!n^3$. By Theorem 4, we have that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

The state space of $\mathcal{A}'$ is $Q' = Q \times R_k(c)$. Intuitively, when $\mathcal{A}'$ is in state $\langle q, \rho \rangle$ as it reads the node $x \in T$, it guesses that the rank of the vertex $\langle \eta(x), q \rangle$ of G is ρ. The transitions of $\mathcal{A}'$ allows the guessed ranks to decrease, but makes sure that if a state is in F_j, then guessed rank for it cannot be odd with index j. By Lemma 3, the guessed ranks should eventually converge to some odd rank, which is checked by the acceptance condition of $\mathcal{A}'$.[8]

In [18], Miyano and Hayashi describe a translation of ABW to NBW. In Theorem 6 below (see also [19]), we present (a technical variant of) their translation, adapted to tree automata,

Theorem 6. *Let $\mathcal{A}$ be an ABT with n states. There is an NBT $\mathcal{A}'$ with $2^{O(n)}$ states, such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Combining Theorems 5 and 6, one can reduce the nonemptiness problem for UGCT to the nonemptiness problem for NBT. Consider a UGCT $\mathcal{A}$ with n states and index k. If we translate $\mathcal{A}$ to an NBT by going through the ABT we have obtained in Theorem 5, we end up with an NBT with $2^{2^{O(n(\log n + \log k))}}$ states, as the ABT has $2^{O(n(\log n + \log k))}$ states. In order to complete the construction, and get the NBT described in the proof of Theorem 2, we exploit the special structure of the ABT and show that only $2^{O(n^2(\log n + \log k))}$ states of the NBT constructed in Theorem 6 may participate in an accepting run.

5 Compositional Synthesis

A serious drawback of current synthesis algorithms is that they assume a comprehensive set of temporal assertions as a starting point. In practice, however, specifications are evolving: temporal assertions are added, deleted, or modified during the design process. In this section we describe how our synthesis algorithm can support *compositional* synthesis, where the temporal assertions are given one by one. We show how the Safra-less approach enables us, when we check the realizability of $\psi \wedge \psi'$, to use much of the work done in checking the realizability of ψ and ψ' in isolation. Devising a compositional synthesis algorithms to other forms of composition, e.g., $\psi' \rightarrow \psi$, is an interesting research problem.

[8] Readers familiar with weak automata [20], would note that our automaton is in fact an alternating weak tree automaton. It is the special structure of weak automata that enables some of the optimizations we describe below.

Our compositional algorithm extends the *incremental-synthesis* algorithm described in [15]. Essentially, we show that when we construct and check the emptiness of the NBT to which realizability of $\psi \wedge \psi'$ is reduced, we can use much of the work done in the process of checking the emptiness of the two (much smaller) NBTs to which realizability of ψ and ψ' is reduced (in isolation).

We first review the incremental-synthesis idea from [15]. Recall that our construction is based on the fact we can bound the maximal rank that a vertex in an accepting run graph G gets. Often, the sequence $G_0, G_1, G_2, \ldots$ of graphs described in Section 4.2 converges to the empty graph very quickly, making the bound on the maximal rank much smaller (see [12] for an analysis and experimental results for the case of UCW). Accordingly, one can regard the bound c as a parameter in the construction: start with a small parameter, and increase it if necessary.

To see how this is done, consider the combined construction described in Theorem 2. Starting with a UGCT $\mathcal{A}$ with state space Q of size n, we took $c = Safra(n,k)n!n^3$ (an upper bound on the size of the minimal accepting run graph of $\mathcal{A}$), and constructed an NBT $\mathcal{A}'$ with state space $3^Q \times \mathcal{R}_k(c)$, where $\mathcal{R}(c)$ is the set of functions $f : Q \rightarrow R_k(c)$ in which $f(q)$ is not odd with index j for all $q \in F_j$. For $l \leq c$, let $\mathcal{R}_k[l]$ be the restriction of $\mathcal{R}_k$ to functions with range $R_k(l)$, and let $\mathcal{A}'[l]$ be the NBT $\mathcal{A}'$ with c being replaced by l. Recall that the NBT $\mathcal{A}'[l]$ is empty iff all the run graphs of $\mathcal{A}$ of size at most l are not accepting. Thus, coming to check the emptiness of $\mathcal{A}$, the incremental approach proceeds as follows: start with a small l and check the nonemptiness of $\mathcal{A}'[l]$. If $\mathcal{A}'[l]$ is not empty, then $\mathcal{A}$ is not empty, and we can terminate with a "nonempty" output. Otherwise, increase l, and repeat the procedure. When $l = c$ and $\mathcal{A}'[l]$ is still empty, we can terminate with an "empty" output.

As argued for UCTs in [15], it is possible to take advantage of the work done during the emptiness test of $\mathcal{A}'[l_1]$, when testing emptiness of $\mathcal{A}'[l_2]$, for $l_2 > l_1$. To see this, note that the state space of $\mathcal{A}'[l_2]$ consists of the union of $3^Q \times \mathcal{R}_k[l_1]$ (the state space of $\mathcal{A}'[l_1]$) with $3^Q \times (\mathcal{R}_k[l_2] \setminus \mathcal{R}_k[l_1])$ (states whose $f \in \mathcal{R}_k[l_2]$ has a state that is mapped to a rank greater than l_1). Also, since ranks can only decrease, once the NBT $\mathcal{A}'[l_2]$ reaches a state of $\mathcal{A}'[l_1]$, it stays in such states forever. So, if we have already checked the nonemptiness of $\mathcal{A}'[l_1]$ and have recorded the classification of its states to empty and nonempty, the additional work needed in the nonemptiness test of $\mathcal{A}'[l_2]$ concerns only states in $3^Q \times (\mathcal{R}[l_2] \setminus \mathcal{R}_k[l_1])$.

We can now describe how the incremental approach can be extended to a compositional one. Let $\mathcal{S} = \langle \Sigma, Q, \delta, q_{in}, \{F_1, \ldots, F_k\}\rangle$ and $\mathcal{S}' = \langle \Sigma, Q', \delta', q'_{in}, \{F'_1, \ldots, F'_{k'}\}\rangle$ be UGCWs specifying required behaviors. Let $n = |Q|$ and $n' = |Q'|$. Without loss of generality, assume that the state spaces Q and Q' are disjoint. We can define the intersection of $\mathcal{S}$ and $\mathcal{S}'$ as the UGCW P obtained by putting $\mathcal{S}$ and $\mathcal{S}'$ "side by side"; thus[9] $P = \langle \Sigma, Q \cup Q', \delta \cup \delta', \{q_{in}, q'_{in}\}, \{F_1 \cup Q', \ldots, F_k \cup Q', F'_1 \cup Q, \ldots, F'_{k'} \cup Q\}\rangle$. Note that it is indeed the case that P has an accepting run on a word w iff both $\mathcal{S}$ and $\mathcal{S}'$ has an accepting run on w.

Let $\mathcal{A}$ and $\mathcal{A}'$ be the NBTs to which realizability of $\mathcal{S}$ and $\mathcal{S}'$ is reduced, respectively. A non-compositional approach generates the NBT that corresponds to P. By

[9] For technical simplicity, we allow P to have two initial states. This can be easily ovoided by adding a new initial state whose transitions are the union of the transitions from q_{in} and q'_{in}.

Theorem 2, this results in an NBT $\mathcal{U}$ with state space $3^{Q\cup Q'} \times R_{k+k'}(p)^{Q\cup Q'}$, for $p = Safra(n+n', k+k')(n'_n)!(n+n')^3$. On the other hand, the state spaces of $\mathcal{A}$ and $\mathcal{A}'$ are much smaller, and are $3^Q \times R_k(c)^Q$ and $3^{Q'} \times R_{k'}(c')^{Q'}$, for $c = Safra(n,k)n!n^3$ and $c' = Safra(n',k')n'!n'^3$. respectively.

Let us examine the structure of the state space of $\mathcal{U}$ more carefully. Each of its states can be viewed as a triplet $\langle S \cup S', O \cup O', f\rangle$, for $O \subseteq S \subseteq Q$, $O' \subseteq S' \subseteq Q'$, and $f : Q \cup Q' \rightarrow R_{k+k'}(p)$. For f as above, let $f_{|Q}$ and $f_{|Q'}$ denote the restrictions of f to Q and Q', respectively. Note that if f maps the states in S to ranks in $R_k(c)$ and maps states in S' to ranks in $R_{k'}(c')$, then the state $\langle S \cup S', O \cup O', f\rangle$ corresponds to the states $\langle S, O, f_{|Q}\rangle$ of $\mathcal{A}$ and $\langle S', O', f_{|Q'}\rangle$ of $\mathcal{A}'$. Moreover, if one of these states is empty, so is $\langle S \cup S', O \cup O', f\rangle$. This observation is the key to our compositional algorithm.

For $l \leq c$ and $l' \leq c'$, let $\mathcal{U}[l, l']$ denote the NBT $\mathcal{U}$ restricted to states $\langle S \cup S', O \cup O', f\rangle$ in which $f(q)$, for $q \in S$, is in $R_k(l)$ and $f(q')$, for $q' \in S'$, is in $R_{k'}(l')$. We check the emptiness of $\mathcal{U}$ incrementally and compositionally as follows. We start with small l_1 and l'_1 and check the emptiness of $\mathcal{U}[l_1, l'_1]$. Doing so, we first mark as empty all states $\langle S \cup S', O \cup O', f\rangle$ for which either $\langle S, O, f_{|Q}\rangle$ is empty in $\mathcal{A}$ or $\langle S', O', f_{|Q'}\rangle$ is empty in $\mathcal{A}'$, and continue the emptiness check only in the (expectedly much smaller) state space. If $\mathcal{U}[l_1, l'_1]$ is not empty, we are done. Otherwise, we increase our parameters to l_2 and l'_2, with $l_2 \geq l_1$ and $l'_2 \geq l'_1$. Note that we need not increase both parameters. Checking the emptiness of $\mathcal{U}[l_2, l'_2]$, we make use of the information gathered in the emptiness checks of $\mathcal{A}[l_2]$, $\mathcal{A}'[l'_2]$, as well as $\mathcal{U}[l_1, l'_1]$. The procedure continues until we either reach l_j and l'_j for which $\mathcal{U}[l_j, l'_j]$ is not empty, in which case the specification is realizable, or we find that $\mathcal{U}[p, p]$ is empty, in which case the specification is not realizable.

We note that, as with the incremental approach, the significant advantage of the compositional approach is when the specification is realizable, and especially when $\mathcal{U}[l, l']$ is not empty for l and l' smaller than c and c' – thus we can use information about $\mathcal{A}$ and $\mathcal{A}'$ all the way to the positive response. We also note that the incremental approach is possible due to the simple structure of the state spaces of the NBTs to which we have reduced the realizability problem. This simple structure also makes it easy to implement our approach symbolically: the state space of the NBT consists of sets of states and a ranking function, it can be encoded by Boolean variables, and the NBT's transitions can be encoded by relations on these variables and a primed version of them. The fixpoint solution for the nonemptiness problem of NBT (c.f., [28]) then yields a symbolic solution to the original UGCT nonemptiness problem. Moreover, checking the emptiness of $\mathcal{U}[l_j, l'_j]$, we can use BDDs for the empty states in $\mathcal{A}[l_j]$, $\mathcal{A}[l'_j]$, and $\mathcal{U}[l_{j-1}, l'_{j-1}]$. Finally, as discussed in [15], the BDDs that are generated by the symbolic nonemptiness procedure can be used to generate a symbolic witness strategy, from which we can synthesize a sequential circuit implementing the strategy.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *16th ICALP*, LNCS 372, pp 1–17. Springer-Verlag, 1989.

2. C. S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. In *10th CIAA*, LNCS. Springer-Verlag, 2005.
3. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *JCSS*, 8:117–141, 1974.
4. A. Church. Logic, arithmetics, and automata. In *ICM, 1962*, pp 23–35, 1963.
5. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
6. D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.
7. J. Elgaard, N. Klarlund, and A. Möller. Mona 1.x: new techniques for WS1S and WS2S. In *10th CAV*, LNCS 1427, pp 516–520. Springer-Verlag, 1998.
8. E.A. Emerson. Automata, tableaux, and temporal logics. In *WLP*, LNCS 193, pp 79–87. Springer-Verlag, 1985.
9. E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *32nd FOCS*, pp 368–377, 1991.
10. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pp 3–18. 1995.
11. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS 2500. Springer-Verlag, 2002.
12. S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *12th CHARME*, LNCS 2860, pp 96–110. Springer-Verlag, 2003.
13. M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pp 290–301. Springer-Verlag, 2000.
14. O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th TACAS*, LNCS 2988, pp 591–606. Springer-Verlag, 2004.
15. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *46th FOCS*, 2005.
16. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *12th POPL*, pp 97–107, 1985.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
18. S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *TCS*, 32:321–330, 1984.
19. A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *CT*, LNCS 208, pp 157–168. Springer-Verlag, 1984.
20. D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *13th ICALP*, LNCS 226. Springer-Verlag, 1986.
21. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th POPL*, pp 179–190, 1989.
22. M.O. Rabin. Weakly definable relations and special automata. In *Symp. Math. Logic and Foundations of Set Theory*, pp 1–23. 1970.
23. M.O. Rabin. Automata on infinite objects and Church's problem. *AMS*, 1972.
24. R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
25. S. Safra. On the complexity of ω-automata. In *29th FOCS*, pp 319–327, 1988.
26. S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *24th STOC*, 1992.
27. S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic omega-automata. In *8th CHARME*, LNCS 987, pp 261–277, 1995. Springer-Verlag.
28. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *JCSS*, 32(2):182–221, 1986.
29. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *IC*, 115(1):1–37, 1994.

30. J. Yang and C.J.H. Seger. Introduction to generalized symbolic trajectory evaluation. In *19th DAC*, pp 360–367. IEEE, 2001.

A Definition of Trees and Alternating Automata

Given a set D of directions, a *tree over* D, or D*-tree*, for short, is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. If $T = D^*$, we say that T is a full D-tree. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$.

For a set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** (an empty conjunction) and **false** (an empty disjunction). For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. An *Alternating tree automaton* is $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω; several types of acceptance conditions can be defined).

The alternating automaton $\mathcal{A}$ runs on Σ-labeled full D-trees. A *run* of $\mathcal{A}$ over a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$.
2. Let $y \in T_r$ with $r(y) = \langle x, q \rangle$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$, such that S satisfies θ, and for all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

For example, if $\langle T, \tau \rangle$ is a $\{0,1\}$-tree with $\tau(\varepsilon) = a$ and $\delta(q_{in}, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then, at level 1, the run $\langle T_r, r \rangle$ includes a node labeled $(0, q_1)$ or a node labeled $(0, q_2)$, and includes a node labeled $(0, q_3)$ or a node labeled $(1, q_2)$. Note that if, for some y, the transition function δ has the value **true**, then y need not have successors. Also, δ can never have the value **false** in a run. A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. That is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π. Acceptance is defined by placing conditions on $inf(\pi)$ for all paths π ofthe run.

In [20], Muller et al. introduce *alternating weak tree automata*. In a weak automaton, we have a Büchi acceptance condition $\alpha \subseteq Q$ and there exists a partition of Q into disjoint sets, $Q_1, \ldots, Q_m$, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$, for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of an alternating weak automaton ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set.

B From NGBW to DRW

We give an exposition of Safra's construction [25] and then proceed with our extension that handles NGBW.

B.1 Determinization of NBW

Here we describe the determinization of NBW [25]. We construct an ordered tree of subset constructions. Every node in the tree is labeled by the states it follows. The labels of siblings are disjoint and the label of a node is a strict superset of the labels of its descendants. The transition of a tree replaces the label of every node by the set of possible successors. If the label now includes some accepting states, we add a new youngest child to the node with all these accepting states. Intuitively, the states that label the children of a node have already visited an accepting state. Thus, the states in the label of a node that are not in the labels of its descendants are states that still owe a visit to the acceptance set. We move states occurring in more than one child of a node to the older child. If the label of a node becomes equal to the union of labels of its children then we mark this node as accepting and remove all its descendants. We associate a Rabin pair with every node in the tree. Erasing a node is a bad event, which should occur finitely often. Finding that a node is accepting is a good event. If some node eventually remains in the tree and is accepting infinitely often the run is accepting.

Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ be an NBW with $|S| = n$. Let $V = [n]$ and $V' = \{n{+}1, \ldots, 2n\}$. A *Safra tree* t over S is a tuple $\langle N, 1, p, \psi, l, R, G \rangle$ where the components of t are as follows.

- $N \subseteq V$ is a set of nodes.
- $1 \in N$ is the root node.
- $p{:}N{\to}N$ is the parenthood function. We call children of the same node *siblings*.
- $\psi{:}N{\to}N$ is a partial order defining "older than" on siblings.
- $l{:}N{\to}2^S$ is a labeling of the nodes with subsets of S.
- $R, G \subseteq V$ are two disjoint subsets of V, defining the set of *red* and *green* nodes.

In addition we require that the label of every node is a proper superset of the union of the labels of its children and that the labels of two siblings are disjoint.

Proposition 1. [25, 15] *The number of nodes in a Safra tree is at most n. The number of Safra trees over S is at most $12^n n^{2n}$.*

Proof: As the labels of siblings are disjoint and the union of labels of children is a proper subset of the label of the parent it follows that every node is the minimal (according to the subset order on the labels) to contain (at least) some state $s \in S$. It follows that there are at most n nodes.

The number of ordered trees on n nodes is the nth Catalan number. That is $Cat(n) = \frac{(2n)!}{n!(n+1)!} \leq 4^n$. We represent the naming of nodes by $f : [n] \to [n]$ that associates the ith node with its name $f(i)$. There are at most n^n such functions. The labeling function is $l : S \to [n]$ where $l(s) = i$ means that s belongs to the ith node and all its ancestors. Finally, we represent R and G by a function $a : V \to \{0, 1, 2\}$ such that $a(i) = 0$ means that $i \notin R \cup G$, $a(i) = 1$ means that $i \in R$, and $a(i) = 2$ means that $i \in G$. There are at most 3^n such functions.

To summarize, the number of trees is at most $4^n \cdot 3^n \cdot n^n \cdot n^n = 12^n n^{2n}$. □

We construct the DRW $\mathcal{D}$ equivalent to $\mathcal{N}$. Let $\mathcal{D} = \langle \Sigma, D, \rho, d_0, \alpha \rangle$ where the components of $\mathcal{D}$ are as follows.

- D is the set of Safra trees over S.
- d_0 is the tree with a single node 1 labeled $\{s_0\}$ where $R{=}V{-}\{1\}$ and $G = \emptyset$.
- The Rabin acceptance condition α is $\{\langle L_1, U_1 \rangle, \ldots, \langle L_n, U_n \rangle\}$ where $L_i = \{d \in D \mid i \in G_d\}$ and $U_i = \{d \in D \mid i \in R_d\}$.
- For every tree $d \in D$ and letter $\sigma \in \Sigma$ the transition $d' = \rho(d, \sigma)$ is the result of the following transformations on d. We temporarily use the set V' of nodes.
  1. For every node v with label S' replace S' by $\delta(S', \sigma)$ and set R and G to the empty set.
  2. For every node v with label S' such that $S' \cap \alpha \neq \emptyset$, create a new youngest child $v' \in V'$. Set its label to $S' \cap \alpha$.

3. For every node v with label S' and state $s \in S'$ such that s belongs also to an older sibling v' of v, remove s from the label of v and all its descendants.
4. For every node v whose label is equal to the union of the labels of its children, remove all descendants of v. Add v to G.
5. Remove all nodes with empty labels and add all unused names to R.
6. Change the nodes in V' to nodes in V.

Proposition 2. [25] $L(\mathcal{D}) = L(\mathcal{N})$.

B.2 Determinization of NGBW

We now present the determinization of NGBW. Intuitively, we take the above construction and add indices to each node. Whenever all the runs followed by a node v indexed i (where i is not the maximal index) visit the set F_i, we increase the index of v to $i+1$. Whenever all the runs followed by a node v indexed by the maximal index k visit the set F_k, we mark v as accepting and change its index to 1. Applying our construction results in a DRW with $12^n n^{2n} k^n$ states and n pairs.

Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ be an NBW with $|S| = n$ and $\alpha = \{F_1, \ldots, F_k\}$. Let $V = [n]$ and $V' = \{n+1, \ldots, 2n\}$. A *generalized Safra tree* t over S and k is $\langle N, 1, p, \psi, l, h, R, G \rangle$ where N, 1, p, ψ, l, R, and G are as in Safra trees, and $h : N \to [k]$ is an indexing function associating with every node an index in $[k]$.

The following proposition is proven much like the similar proposition for Safra trees (Proposition 1).

Proposition 3. *The number of generalized Safra trees over S and k is at most $12^n n^{2n} k^n$.*

Proof: Just like Safra trees there are at most n nodes. We have to add the function $h : N \to [k]$, which associates an index with every node. This multiplies the number of states by k^n. □

We show that the DRW $\mathcal{D}$ constructed in Section 4.1 is equivalent to $\mathcal{N}$. The proof is an extension of the proof in [25].

Proposition 4. $L(\mathcal{D}) = L(\mathcal{N})$.

Proof: Consider $w \in L(\mathcal{N})$. We have to show $w \in L(\mathcal{D})$. Let $r = s_0 s_1 \cdots$ be an accepting run of $\mathcal{N}$ on w. Let $r' = d_0 d_1 \cdots$ be the run of $\mathcal{D}$ on w and let $d_i = \langle N_i, 1, p_i, \psi_i, l_i, h_i, R_i, G_i \rangle$. It is simple to see that for all $i \geq 0$ we have $s_i \in l_i(1)$. It follows that for all $i \geq 0$ we have $1 \notin R_i$. If step 4 is applied infinitely often to node 1 (equivalently, $1 \in G_i$ infinitely often, or during the transformation of the trees the label of 1 equals the labels of its children) then r' satisfies the pair $\langle L_1, U_1 \rangle$.

Otherwise, from some point onwards in r' we have that step 4 is not applied to node 1 and the index of node 1 is constant. Let j_1 be this point and let i_1 be the index of node 1. There exists a point $j' > j_1$ such that $s_{j'} \in F_{i_1}$. It follows that for all $j > j'$ we have s_j belongs to some son v_1 of 1. Notice, that the run r may start in some son of 1 and move to an older son. This, however, can happen only finitely often and hence we treat v_1 as constant. There exists a point after which r remains in v_1 forever. Let o_1 be the point such that for all $o > o_1$ we have $s_i \in l_i(v_1)$. It follows that for all $o > o_1$ we have $v_1 \notin R_o$ and $d_o \notin U_{v_1}$.

Suppose that step 4 is applied to v_1 infinitely often (equivalently, $v_1 \in G_i$ infinitely often). The the run r' satisfies the Rabin pair $\langle L_{v_1}, U_{v_1} \rangle$ and r' is accepting. Otherwise, step 4 is applied to v_1 finitely often. We construct by induction a sequence $v_1, \ldots, v_k$ such that eventually $v_1, \ldots, v_k$ all remain in the tree and r belongs to all of them. As the number of active nodes in a

tree (nodes v such that $l(v) \neq \emptyset$) is bounded by n we can repeat the process only finitely often. Hence, w is accepted by $\mathcal{D}$.

In the other direction, consider $w \in L(\mathcal{D})$. Let $r' = d_0 d_1 \cdots$ be the accepting run of $\mathcal{D}$ on w where $d_i = \langle N_i, 1, p_i, \psi_i, l_i, h_i, R_i, G_i \rangle$. Let $\langle L_b, U_b \rangle$ be the Rabin pair for which L_b is visited infinitely often and U_b is visited finitely often.

We first prove three propositions.

Proposition 5. *For every $i \in \mathbb{N}$, $v \in V$, and every state $s \in l_i(v)$, we have s is reachable from s_0 reading $w[0, i-1]$.*

Proof: We prove the proposition for all $v \in V$ by induction on i. Clearly, it holds for $i = 0$. Suppose that it holds for i. As $l_{i+1}(v) \subseteq \delta(l_i(v'), w_i)$ for some $v' \in V$, it follows that every state in $l_{i+1}(v)$ is reachable from s_0 reading $w[0, i]$. □

Proposition 6. *Consider $i, i' \in \mathbb{N}$ such that $i < i'$. Suppose that (a) $v \in V$ is a leaf in the trees $d_i, d_{i'}$ (b) for all $i \leq a \leq i'$ we have $d_a \notin U_v$ and (c) $h_{i'}(v) = h_i(v) + 1$ and for all $i < a < i'$ we have $h_a(v) = h_i(v)$. Then every state s in $l_{i'}(v)$ is reachable from some state in $l_i(v)$ reading $w[i, i'-1]$ with a run that visits $F_{r_i(v)}$.*

Proof: By assumption, the set U_v is not visited between i and i'. Hence, the node v is not removed in the transition from d_a to d_{a+1} for all $i \leq a < i'$. In addition from minimality of i' it follows that for all $i \leq a < i'$ we have $r_a(v) = r_i(v)$.

We prove that for every $i < a < i'$ and every descendant v' of v, every state in $l_a(v')$ is reachable from some state in $l_i(v)$ along a run visiting $F_{r_i(v)}$. In the case that $i' = i + 1$ then, all the states in $l_{i'}(v)$ are in $F_{r_i(v)}$ and we are done. Otherwise, consider the descendant v' of v appearing in d_{i+1} (there is at most one, it must exist as i' is minimal). As $l_{i+1}(v') \subseteq \delta(l_i(v), w_i) \cap F_{r_i(v)}$ the claim follows for $i + 1$. Suppose that the claim is true for a and prove for $a + 1$. We know that for every descendant v' of v either $l_{a+1}(v') \subseteq \delta(l_a(v), w_a) \cap F_{r_a(v)}$ or for some descendant v'' of v we have $l_{a+1}(v') \subseteq \delta(l_a(v''), w_a)$ (and v' may be v''). As during the transformation from $d_{i'-1}$ to $d_{i'}$ the label $l_{i'}(v)$ equals the union of the labels of children of v (from i' being minimal and $r_{i'}(j) = r_i(j) + 1$) the proposition follows. □

Proposition 7. *Consider $i, i' \in \mathbb{N}$ such that $i < i'$. Suppose that (a) for some $v \in V$ we have $d_i, d_{i'} \in L_v$ and (b) for all $i < a < i'$ we have $d_a \notin U_v$. Then every state s in $l_{i'}(v)$ is reachable from some state in $l_i(v)$ reading $w[i, i'-1]$ with a run that visits all sets in α.*

Proof: As $d_i \in L_v$ we know that v appears in d_i. By assumption, the set U_v is not visited between i and i'. Hence, the node v is not removed in the transition from d_a to d_{a+1} for all $i \leq a < i'$.

In addition, consider the sequence of indices $r_a(j+1)$ between i and i'. As both d_i and $d_{i'}$ and in L_v we know that $r_i(v) = 1$ and $r_{i'}(v) = 1$. Then there exist $i = i_1 < i_2 < \cdots < i_k < i'$ such that for all $i_l \leq a < i_{l+1}$ we have $r_a(v) = l$.

From Proposition 6 it follows that every state in $l_{i_{l+1}}(v)$ is reachable from some state in $l_{i_l}(v)$ reading $w[i_l, i_{l+1} - 1]$ with a run that visits F_l. It follows that every state in $l_{i'}(v)$ is reachable from some state in $l_i(v)$ reading $w[i, i'-1]$ with a run that visits F_m for every $1 \leq m \leq k$. □

We construct an infinite tree with finite branching degree. The root of the tree corresponds to the initial state of $\mathcal{N}$. Every node in the tree is labeled by some state of $\mathcal{N}$ and a time stamp i. An edge between the nodes labeled (s, i) and (t, j) corresponds to a run starting in s, ending in t, reading $w[i, j-1]$, and visiting F_i for all $1 \leq i \leq k$. From König's Lemma this tree contains

an infinite branch. The composition of all the run segments in this infinite branch is an infinite accepting run of $\mathcal{N}$ on w.

Let $(s_0, 0)$ label the root of T. Let $\langle L_v, U_v \rangle$ be the pair such that L_v is visited infinitely often and U_v finitely often. Let i_0 be the minimal location such that U_v is not visited after i_0. Let i_1 be the minimal location such that $i_1 > i_0$ and $d_{i_1} \in L_v$ (that is step 4 was applied to v). For every state s in $l_{i_1}(v)$ we add a node to T, label it by (s, i_1) and connect it to the root. We extend the tree by induction. We have a tree with leafs labeled by the states in $l_a(v)$ stamped by time a, and $d_a \in L_v$ (step 4 was applied to v). That is, for every state s in $l_a(v)$ there exists a leaf labeled (s, a). We know that L_v is visited infinitely often. Hence, there exists $a' > a$ such that $d_{a'} \in L_v$ (step 4 is applied to v). For every state s' in $l_{a'}(v)$ we add a node to T and label it (s', a'). From Proposition 7 state s' is reachable from state $s \in l_a(v)$ reading $w[a, a' - 1]$. We connect the node (s', a') to the node (s, a). From Proposition 5 it follows that every edge $(s_0, 0), (s', i')$ corresponds to some run starting in s_0, ending in s', and reading $w[0, i' - 1]$. From Proposition 7, every other edge in the tree $(s, a), (s', a')$ corresponds to some run starting in s, ending in s', reading $w[a, a' - 1]$, and visiting F_m for all $1 \leq m \leq k$. From König's Lemma there exists an infinite branch in the tree. This infinite branch corresponds to an accepting run of $\mathcal{N}$ on w. □

Theorem 7. *Given an NGBW with n states and index k, it is possible to construct an equivalent DRW with at most $12^n n^{2n} k^n$ states and n pairs.*

C Proofs from Section 4.2

C.1 Proof of Lemma 1

Proof: We say that a path $\pi = y_0 \cdot y_1 \cdot y_2 \cdots$ of $\langle T_r, r \rangle$ corresponds to a path $\pi' = \langle s_0, q_0 \rangle, \langle s_1, q_1 \rangle, \langle s_2, q_2 \rangle, \ldots$ of G iff $s_0 = s_{in}$, $q_0 = q_{in}$, and there is a path $x_0, x_1, x_2, \ldots$ of T, with $x_{i+1} = x_i \cdot c_i$, such that for all $i \geq 0$, we have that $r(y_i) = \langle x_i, q_i \rangle$ and $\eta(s_i, c_i) = s_{i+1}$. Thus, π' describes the states of $\mathcal{T}$ and $\mathcal{A}$ that the copy of $\mathcal{A}$ whose evolution is recorded in the path π visits. Clearly, for all $1 \leq j \leq k$, we have that π has infinitely many nodes y_i with $r(y_i) \in T \times F_j$ iff π' visits infinitely many F_j-vertices. By the definition of G, each path of $\langle T_r, r \rangle$ corresponds to a single path of G. Also, each path π' of G has at least one path π of $\langle T_r, r \rangle$ such that π corresponds to π'. To see this, note that since $\langle T, \tau \rangle$ is induced by $\mathcal{T}$, then $T = D^*$ and for all $x \in D^*$, we have that $\tau(x) = L(\eta(x))$. In addition, by the definition of G, for all $i \geq 0$ there is $c_i \in D$ such that $(c_i, q_{i+1}) \in \delta(q_i, L(s_i))$ and $\eta(s_i, c_i) = s_{i+1}$; the sequence of these x_i's induces a path $x_0, x_1, x_2, x_3, \ldots$ of T, with $x_{i+1} = x_i \cdot c_i$. The run of $\mathcal{A}$ on $\langle T, \tau \rangle$ contains a copy that reads this path and visits $q_0, q_1, q_2, \ldots$, and the path π of $\langle T_r, r \rangle$ describes this copy. Hence, for every $1 \leq j \leq k$, we have that $\langle T_r, r \rangle$ has an infinite path that visits F_j finitely often iff G has an infinite path with finitely many F_j-vertices, and we are done. □

C.2 Proof of Theorem 4

Proof: Assume first that $\mathcal{A}$ has an accepting run graph G (of any size) on some transducer $\mathcal{T}$. Let $\langle T, \tau \rangle$ be the tree generated by $\mathcal{T}$. Thus, $T = D^*$ and for all $x \in D^*$ we have that $\tau(x) = L(\eta(x))$. Consider the run $\langle T_r, r \rangle$ of $\mathcal{A}$ on $\langle T, \tau \rangle$. By Lemma 1, $\langle T_r, r \rangle$ is accepting. Hence, $\mathcal{A}$ is not empty.

For the other direction, consider the UGCT $\mathcal{A}$. By [9], there is a DRT $\mathcal{A}^d$ equivalent to $\mathcal{A}$, which is constructed as follows. Let $\mathcal{A}'$ be an NGBW that runs over a branch of an input tree for $\mathcal{A}$ and checks whether $\mathcal{A}$ has a rejecting path over this branch. The NGBW $\mathcal{A}'$ is obtained by dualizing $\mathcal{A}$ and following the run read in the input. Thus, $\mathcal{A}'$ has the same state space and index as $\mathcal{A}$. Let $\mathcal{A}''$ be a DRW that is equivalent to $\mathcal{A}'$ (by Theorem 3). Now, we construct a DRW $\mathcal{A}'''$

for the complementary language, and run $\mathcal{A}'''$ over all branches of the input tree to check that all paths of the run tree of $\mathcal{A}$ are accepting. This yields the DRT $\mathcal{A}^d$ that is equivalent to $\mathcal{A}$.

By Theorem 3, the DRW $\mathcal{A}''$ has at most $12^n n^{2n} k^n$ states. The DRW $\mathcal{A}'''$ and the DRT $\mathcal{A}^d$ have at most $n' = 12^n n^{2n} k^n n!$ states [26]. By [8], an NRT with n' states is not empty iff it accepts a regular tree generated by a transducer with n' states. The state space of the run graph of $\mathcal{A}$ on such a transducer is then bounded by $nn' = 12^n n^{2n+3} k^n n!$. Since the run of $\mathcal{A}$ on the tree is accepting, Lemma 1 implies that so is the run graph. □

C.3 Proof of Lemma 2

Proof: Assume first that G is accepting. We prove that for all $i \geq 0$, the graph G_{2i} has at most $|V| - i$ vertices. In particular, $G_{2|V|}$ has at most 0 vertices, so there is $i \leq |V|$ for which G_{2i} is empty. The proof proceeds by an induction on i. Clearly, G_0 has at most $|V|$ vertices. For the induction step, we prove that for all $i \geq 0$, if G_{2i} contains vertices that are not finite, then G_{2i+1} contains at least one F_j-free vertex, for some $1 \leq j \leq k$. It follows that the transition from G_{2i} to G_{2i+2} either results in an empty G_{2i+2} or involves a removal of at least one vertex.

Consider the graph G_{2i} and assume by way of contradiction that it contains a vertex $\langle s_0, q_0 \rangle$ that is not finite, and yet no vertex in G_{2i+1} is F_j-free, for all $1 \leq j \leq k$. Consider the graph G_{2i+1}. All the vertices in G_{2i+1} are not finite, and therefore, each of the vertices in G_{2i+1} has at least one successor. Consider the vertex $\langle s_0, q_0 \rangle$ in G_{2i+1}. Since, by the assumption, it is not F_1-free, there exists an F_1-vertex $\langle s'_0, q'_0 \rangle$ reachable from $\langle s_0, q_0 \rangle$. Let $\langle s_1, q_1 \rangle$ be a successor of $\langle s'_0, q'_0 \rangle$. By the assumption, $\langle s_1, q_1 \rangle$ is also not F_2-free. Hence, there exists an F_2-vertex $\langle s'_1, q'_1 \rangle$ reachable from $\langle s_1, q_1 \rangle$. Let $\langle s_2, q_2 \rangle$ be a successor of $\langle s'_1, q'_1 \rangle$. By the assumption, $\langle s_2, q_2 \rangle$ is also not F_3-free. Hence, there exists an F_3-vertex $\langle s'_3, q'_3 \rangle$ reachable from $\langle s_2, q_2 \rangle$. We can continue similarly and construct an infinite sequence of vertices $\langle s_h, q_h \rangle$, $\langle s'_h, q'_h \rangle$ such that for all $h \geq 0$, the vertex $\langle s'_h, q'_h \rangle$ is an $F_{(h \bmod k)+1}$-vertex reachable from $\langle s_h, q_h \rangle$, and $\langle s_{h+1}, q_{h+1} \rangle$ is a successor of $\langle s'_h . q'_h \rangle$. Such a sequence, however, corresponds to a path in G that visits F_j infinitely often, for all $1 \leq j \leq k$, contradicting the assumption that G is accepting.

Assume now that G is rejecting. Then, G contains an infinite path π with infinitely many F_j-vertices, for all $1 \leq j \leq k$. We prove that for all $i \geq 0$, all the vertices $\langle s, q \rangle$ in π are in G_{2i}. The proof proceeds by induction on i. The vertices in π are clearly members of G_0. Also, if all the vertices in π are members of G_{2i}, it must be that they are neither finite nor F_j-free in G_{2i+1}, for all $1 \leq j \leq k$, so they stay in G_{2i+2}. □

C.4 Proof of Lemma 3

Proof: Consider a run graph $G = \langle V, E \rangle$. Let $c = |V|$. We prove the following two claims.

1. For every vertex $\langle s, q \rangle$ in G and $\rho \in R(c)$, we have $\langle s, q \rangle \notin G_\rho$ iff $rank(s, q) < \rho$.
2. For every two vertices $\langle s, q \rangle \neq \langle s', q' \rangle$ in G, if $\langle s', q' \rangle$ is reachable from $\langle s, q \rangle$, then $rank(s', q') \leq rank(s, q)$.

We start with Claim (1): for every vertex $\langle s, q \rangle$ in G and $\rho \in R(c)$, we have $\langle s, q \rangle \notin G_\rho$ iff $rank(s, q) < \rho$.

We first prove that if $rank(s, q) < \rho$ then $\langle s, q \rangle \notin G_\rho$. Let $rank(s, q) = \rho'$. By the definition of ranks, $\langle s, q \rangle$ is finite or F_j-free (for the appropriate $1 \leq j \leq k$) in $G_{\rho'}$. Hence, $\langle s, q \rangle$ is removed from $G_{\rho'}$. Hence, as $\rho > \rho'$, also $\langle s, q \rangle \notin G_\rho$.

For the other direction, we proceed by an induction on ρ. Since $G_0 = G$, the case where $\rho = 0$ is immediate. For the induction step, we distinguish between two cases. For the case $\rho = \langle 2i+1, j \rangle$, consider a vertex $\langle s, q \rangle \notin G^{j+1}_{2i+1}$. If $\langle s, q \rangle \notin G^j_{2i+1}$, the lemma's requirement follows from the induction hypothesis. If $\langle s, q \rangle \in G^j_{2i+1}$, then $\langle s, q \rangle$ is F_j-free in G^j_{2i+1}. Accordingly,

$rank(\langle s,q\rangle) = \langle 2i+1, j\rangle$, meeting the lemma's requirement. For the case $\rho = 2i$, consider a vertex $\langle s,q\rangle \notin G^1_{2i+1}$. If $\langle s,q\rangle \notin G_{2i}$, the lemma's requirement follows from the induction hypothesis. If $\langle s,q\rangle \in G_{2i}$, then $\langle s,q\rangle$ is finite in G_{2i}. Accordingly, $rank(\langle s,q\rangle) = 2i$, meeting the lemma's requirement.

We now prove Claim (2): for every two vertices $\langle s,q\rangle \neq \langle s',q'\rangle$ in G, if $\langle s',q'\rangle$ is reachable from $\langle s,q\rangle$, then $rank(s',q') \leq rank(s,q)$.

We distinguish between two cases. If $rank(\langle s,q\rangle) = 2i$ is even, then $\langle s,q\rangle$ is finite in G_{2i}. Hence, either $\langle s',q'\rangle$ is not in G_{2i}, in which case, by Claim (1), we have that $rank(\langle s',q'\rangle) < 2i$, or $\langle s',q'\rangle$ is in G_{2i}, in which case, being reachable from $\langle s,q\rangle$, it must be finite in G_{2i}, with $f(\langle s',q'\rangle) = 2i$, and we are done.

If $f(\langle s,q\rangle) = \langle 2i+1, j\rangle$ is odd, then $\langle s,q\rangle$ is F_j-free in G^j_{2i+1}. Hence, either $\langle s',q'\rangle$ is not in G^j_{2i+1}, in which case, by Claim (1), we have that $f(\langle s',q'\rangle) < \langle 2i+1, j\rangle$, or $\langle s',q'\rangle$ is in G^j_{2i+1}, in which case, being reachable from $\langle s,q\rangle$, it must be F_j-free in G^j_{2i+1}, in which case $f(\langle s',q'\rangle) = \langle 2i+1, j\rangle$, and we are done.

Since even ranks are given to finite vertices, we are done. □

C.5 Proof of Theorem 5

Proof: Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha\rangle$, and let $c = 12^n n^{2n+3} k^n n!$. The ABT $\mathcal{A}'$ accepts all the regular trees $\langle T,\tau\rangle \in \mathcal{L}(\mathcal{A})$ that are generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$ with at most $12^n n^{2n+2} k^n n!$ states. Note that the run graph of $\mathcal{A}$ on such $\langle T,\tau\rangle$ is accepting and is of size most c. By Theorem 4, we have that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

We define $\mathcal{A}' = \langle \Sigma, D, Q', q'_{in}, \delta', \alpha'\rangle$, where

- $Q' = Q \times R_k(c)$. Intuitively, when $\mathcal{A}'$ is in state $\langle q,\rho\rangle$ as it reads the node $x \in T$, it guesses that the rank of the vertex $\langle \eta(x), q\rangle$ of G is ρ. An exception is the initial state q'_{in} explained below.
- $q'_{in} = \langle q_{in}, 2c\rangle$. That is, q_{in} is paired with $2c$, which is an upper bound on the rank of $\langle \eta(\varepsilon), q_{in}\rangle$.
- We define δ' by means of a function

$$release : \mathcal{B}^+(D \times Q) \times R_k(c) \to \mathcal{B}^+(D \times Q').$$

  Given a formula $\theta \in \mathcal{B}^+(D \times Q)$, and a rank $\rho \in R_k(c)$, the formula $release(\theta, i)$ is obtained from θ by replacing an atom (d,q) by the disjunction $\bigvee_{\rho' \leq \rho}(d, \langle q, i'\rangle)$. For example, if $k = 3$, then $release((1,q) \wedge (2,s), 2) = ((1,\langle q,2\rangle) \vee (1,\langle q,(1,1)\rangle) \vee (1,\langle q,(1,2)\rangle) \vee (1,\langle q,(1,3)\rangle) \vee (1,\langle q,0\rangle)) \wedge ((2,\langle s,2\rangle) \vee (2,\langle s,(1,3)\rangle) \vee (2,\langle s,(1,2)\rangle) \vee (2,\langle s,(1,1)\rangle) \vee (2,\langle s,0\rangle))$.

  Now, $\delta' : Q' \times \Sigma \to \mathcal{B}^+(D \times Q')$ is defined, for a state $\langle q,\rho\rangle \in Q'$ and $\sigma \in \Sigma$, as follows.

$$\delta'(\langle q,\rho\rangle, \sigma) = \begin{cases} release(\delta(q,\sigma),\rho) & \text{If } q \notin F_j \text{ or } \rho \text{ is not odd with index } j. \\ \textbf{false} & \text{If } q \in F_j \text{ and } \rho \text{ is odd with index } j. \end{cases}$$

  That is, if the current guessed rank is ρ then, by employing $release$, the run can move in its successors to every rank that is smaller than or equal to ρ. If, however, $q \in F_j$ and the current guessed rank is odd with index j, then, by the definition of ranks, the current guessed rank is wrong, and the run is rejecting.
- $\alpha' = Q \times [2c]^{odd}$. That is, infinitely many guessed ranks along each path should be odd.

We prove that $\mathcal{A}'$ accepts all the regular trees $\langle T,\tau\rangle \in \mathcal{L}(\mathcal{A})$ that are generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$ with at most $12^n n^{2n+2} k^n n!$ states. Note that the run graph of $\mathcal{A}$

on such $\langle T, \tau \rangle$ is accepting and is of size most c. By Theorem 4, we then have that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

We first prove that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. Consider a tree $\langle T, \tau \rangle$ accepted by $\mathcal{A}'$. Let $\langle T_r, r' \rangle$ be the accepting run of $\mathcal{A}'$ on $\langle T, \tau \rangle$. Consider the $T \times Q$-labeled tree $\langle T_r, r \rangle$ where for all $y \in T_r$ with $r'(y) = (x, \langle q, \rho \rangle)$, we have $r(y) = (x, q)$. Thus, $\langle T_r, r \rangle$ projects out the $R_k(c)$ element of the labels of $\langle T_r, r' \rangle$. It is easy to see that $\langle T_r, r \rangle$ is a run of $\mathcal{A}$ on $\langle T, \tau \rangle$. Indeed, the transitions of $\mathcal{A}'$ only annotate transitions of $\mathcal{A}$ by ranks. We show that $\langle T_r, r \rangle$ is an accepting run. Since $\langle T_r, r' \rangle$ is accepting, then, by the definition of α', each infinite path of $\langle T_r, r' \rangle$ gets trapped in a set $Q \times \{\rho\}$ for some odd ρ. By the definition of δ', no accepting run can visit a state $\langle q, \rho \rangle$ with an odd η of index j and $q \in F_j$. Hence, the infinite path actually gets trapped in the subset $(Q \setminus F_j) \times \{\rho\}$ of $Q \times \{\rho\}$. Consequently, in $\langle T_r, r \rangle$, all the paths visit states in F_j only finitely often, and we are done.

It is left to prove that if $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ is a transducer with at most $12^n n^{2n+2} k^n n!$ states and the run graph of $\mathcal{A}$ on $\mathcal{T}$ is accepting, then $\mathcal{A}'$ accepts the regular tree generated by $\mathcal{T}$. Let $\mathcal{T}$ be as above and let G be the accepting run graph of $\mathcal{A}$ on $\mathcal{T}$. Consider the $(T \times Q')$-labeled $\mathbb{N}$-tree $\langle T'_r, r' \rangle$ defined as follows.

- $\varepsilon \in T'_r$ and $r'(\varepsilon) = (\epsilon, \langle q_{in}, 2c \rangle)$.
- Let $y \in T'_r$ be such that $r'(y) = (x, \langle q, \rho \rangle)$. By the definition of $\langle T'_r, r' \rangle$ so far, $\langle \eta(x), q \rangle$ is a vertex in G . Let $\delta(q, \tau(x)) = \{\langle d_1, q_1 \rangle, \ldots, \langle d_m, q_m \rangle\}$. By the definition of G, the vertex $\langle \eta(x), q \rangle$ has successors $\langle s_1, q_1 \rangle, \ldots, \langle s_m, q_m \rangle$ such that for all $1 \leq i \leq m$, we have that $\eta(\eta(x), d_i) = s_i$. Then, for all $1 \leq i \leq m$, we have $y \cdot i \in T'_r$, and $r'(y \cdot i) = (x \cdot d_i, \langle q_i, \mathit{rank}(\eta(x_i), q_i) \rangle)$.

We claim that $\langle T'_r, r' \rangle$ is an accepting run of $\mathcal{A}'$ on $\langle T, \tau' \rangle$. We first prove that $\langle T'_r, r' \rangle$ is a legal run. Since $q'_{in} = \langle q_{in}, 2c \rangle$, the root of T'_r is labeled legally. We now consider the other nodes of T'_r. Let $\{(s_1, q_1), \ldots, (s_m, q_m)\}$ be the successors of (ε, q_{in}) in G, with $s_i = \eta(s_{in}, d_i)$. As c is the maximal rank that a vertex can get, each successor (s_i, q_i) has $\mathit{rank}(s_i, q_i) \leq k$. Thus, as $2c$ is even, the set $\{(c_1, \langle q_1, \mathit{rank}(x_1, q_1) \rangle), \ldots, (c_m, \langle q_m, \mathit{rank}(x_m, q_m) \rangle)\}$ satisfies $\delta'(\langle q_{in}, 2c \rangle, \tau(\varepsilon))$. Hence, the first level of T'_r is labeled legally. For the other levels, consider a node $y \in T'_r$ such that $y \neq \varepsilon$. Let $r'(y) = (x, \langle q, \rho \rangle)$. By the definition of $\langle T'_r, r' \rangle$, we have that $(\eta(x), q)$ is a vertex of G with $\mathit{rank}(\eta(x), q) = \rho$. Let $\{(s_1, q_1), \ldots, (s_m, q_m)\}$ be the successors of $(\eta(x), q)$ in G with $s_i = \eta(s_{in}, d_i)$. As argued in the proof of Lemma 3, for all $1 \leq i \leq m$, we have $\mathit{rank}(s_i, q_i) \leq \rho$. Also, by the definition of ranks, it cannot be that $q \in F_j$ and ρ is odd with index j. Therefore, the set $\{(d_1, \langle q_1, \mathit{rank}(\eta(x_1), q_1) \rangle), \ldots, (d_m, \langle q_m, \mathit{rank}(\eta(x_m), q_m) \rangle)\}$ satisfies $\delta'(\langle q, \rho \rangle, \tau(x))$. Hence, the tree $\langle T'_r, r' \rangle$ is a legal run of $\mathcal{A}'$ on $\langle T, \tau' \rangle$. Finally, by Lemma 3, each infinite path of $\langle T'_r, r' \rangle$ gets trapped in a set with an odd rank, thus $\langle T'_r, r' \rangle$ is accepting.

We now analyze the size of Q'. Recall that the size of $R_k(c)$ is $c(k+1)$, with $c = 12^n n^{2n+3} k^n n!$. Thus, the size of $Q' = Q \times R_k(c)$ is $12^n n^{2n+4} k^n n!(k+1) = 2^{O(n(\log n + \log k))}$. □

C.6 Proof of the construction in Theorem 2

Proof: We prove that the construction described Theorems 5 and 6 result in the NBT described in Theorem 2.

Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$ with $|Q| = n$. Let $c = 12^n n^{2n+3} k^n n!$. Consider a state $\langle S, O \rangle$ of the NBT constructed from $\mathcal{A}$ as described above. Each of the sets S and O is a subset of $Q \times R_k(c)$. We say that a set $P \subseteq Q \times R_k(c)$ is *consistent* iff for every two states $\langle q, \rho \rangle$ and $\langle q', \rho' \rangle$ in P, if $q = q'$ then $\rho = \rho'$. We claim the following: (1) Restricting the states of the NBT to pairs $\langle S, O \rangle$ for which S is a consistent subset of $Q \times R_k(c)$ is allowable; that is, the resulting NBT is equivalent. (2) There are $2^{O(n^2(\log n + \log k))}$ consistent subsets of $Q \times R_k(c)$.

In order to prove Claim (1), recall that the ABT visiting a state $\langle q, \rho \rangle$ when reading a node $x \in T$ corresponds to a guess that the rank of the vertex $\langle \eta(x), q \rangle$ of an accepting run graph G is ρ. Since every vertex in G has a unique rank, the copies of ABT that are generated in an accepting run that corresponds to G are consistent, in the sense that the different copies that read the same node x agree on the rank that $\langle \eta(x), q \rangle$ has in G. When the NBT visits a state $\langle S, O \rangle$, all the states in S correspond to copies of the ABT that read the same node. Hence, a state $\langle S, O \rangle$ for which S is inconsistent corresponds to a node in the run of the ABT whose copies are inconsistent. Hence, the NBT can ignore states $\langle S, O \rangle$ with inconsistent S.

In order to prove Claim (2), observe that we can characterize a consistent set by the projection of its pairs on Q, augmented by an assignment $f : Q \to R_k(c)$. The size of $R_k(c)$ is bounded by ck. Since there are 2^n such projections and $(ck)^n = 2^{O(n^2(\log n + \log k))}$ such assignments, we are done.

By the two claims, as O is always a subset of S, we can restrict the state space of the NBT to $2^{O(n^2(\log n + \log k))}$ states. The construction that follows is described in the proof of Theorem 2. □