Bounded Synthesis for Streett, Rabin, and CTL*

Ayrat Khalimov and Roderick Bloem*

Graz University of Technology, Austria

Abstract. SMT-based bounded synthesis uses an SMT solver to synthesize systems from LTL properties by going through co-Büchi automata. In this paper, we show how to extend the ranking functions used in Bounded Synthesis, and thus the bounded synthesis approach, to Büchi, Parity, Rabin, and Streett conditions. We show that we can handle both existential and universal properties this way, and therefore, that we can extend Bounded Synthesis to CTL*. Thus, we obtain the first Safraless synthesis approach and the first synthesis tool for (conjunctions of) the acceptance conditions mentioned above, and for CTL*.

1 Introduction

Reactive synthesis is the problem of constructing a correct system from a specification given as a temporal logic formula or an automaton. For Linear Temporal Logic [22], the standard approach to synthesis involves Safra's relatively complex construction [26,20] to determinize Büchi automata [23]. Over the last decade, alternatives to this approach have led to a boom in reactive synthesis. Besides methods that limit the expressibility of the logic [5] and complete methods including [12,28], Kupferman and Vardi's *Safraless* approach [17] and Finkbeiner and Schewe's *Bounded Synthesis* [27] provide relatively simple and efficient methods to synthesize LTL formulas.

The idea behind Bounded Synthesis is the following. LTL properties can be translated to Büchi automata [29] and verification of LTL properties can be reduced to deciding emptiness of the product of this automaton and the Kripke structure representing an implementation [19,30]. This product is a nondeterministic Büchi automaton in its own right. Finkbeiner and Schewe made two important observations: (1) Using a ranking function, the emptiness problem of nondeterministic Büchi automata can be encoded as a Satisfiability modulo Theories (SMT) query, and (2) by fixing its size, the Kripke structure can be left uninterpreted, resulting in an SMT query for a system that fulfills the property. Because the size of the system is bounded by Safra's construction, this yields an approach to LTL synthesis that is complete in principle. (Although proofs of unrealizability are usually computed differently.)

The reduction to SMT used by Bounded Synthesis provides two benefits: the performance progress of SMT solvers and the flexibility. With the latter, it is easy

* The authors order was defined by a fair coin.

to adapt the SMT constraints produced by Bounded Synthesis to build semi-complete synthesizers for distributed [27], self-stabilising [6], parameterized [13], assume-guarantee [7], probabilistic [3], and partially implemented systems.

In this paper, we extend Bounded Synthesis in two directions. First, we show how we can directly encode into SMT that some path of a system is accepted by an X automaton, for $X \in \{\text{Büchi, co-Büchi, Parity, Streett, Rabin}\}$. We do this by introducing new ranking functions, circumventing the need to explicitly translate these automata into Büchi automata.

Second, we extend Bounded Synthesis to branching logics. The branching logics allow the user to specify structural properties of the system. For example, if g is system output and r is system input, then the CTL* formula $\mathsf{AG}\,\mathsf{EF}\,g$ says that a state satisfying g is always reachable; and the CTL* formula $\mathsf{EFG}(g \wedge r)$ roughly says that a state satisfying g is reachable and it has a self-loop satisfying r. In both cases, the existential path quantifier E allows us to refrain from specifying the exact path that leads to such states.

We show two approaches to the bounded synthesis for CTL*. First, we show that we can use the ranking functions for X automata to either decide that some path of a system fulfills such a condition, or that *all* paths of the system do. Once we have established this fact, we can extend Bounded Synthesis to logics like CTL* by replacing all state subformulas by fresh atomic propositions and encoding them each by a Büchi automaton. This approach follows the classical construction [9] of model checking CTL*, extending it to synthesis setting. Alternative, we show that we can use a translation of CTL* to Hesitant Alternating Automata [18] to obtain a relatively simple encoding to SMT.

Thus, we obtain a relatively simple, Safraless synthesis procedure to (conjunctions of) various acceptance conditions and CTL*. This gives us a full decision procedure that is efficient when the specification is satisfied by a small system, but is admittedly impractical at showing unrealizability. Just like Bounded Synthesis does for LTL synthesis, it also gives us a semi-decision procedure for undecidable problems such as distributed [24] or parameterized synthesis [13,15]. We have implemented the CTL* synthesis approach in a tool which to our knowledge is the only tool that supports CTL* synthesis.

2 Definitions

Notation: $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ is the set of Boolean values, $\mathbb{N}$ is the set of natural numbers (excluding 0), $[k]$ is the set $\{i \in \mathbb{N} \mid i \leq k\}$ and $[0, k]$ is the set $[k] \cup \{0\}$ for $k \in \mathbb{N}$.

In this paper we consider *finite* systems and automata.

A *(Moore) system* M is a tuple $(I, O, T, t_0, \tau, \mathit{out})$ where I and O are disjoint sets of input and output variables, T is the set of states, $t_0 \in T$ is the initial state, $\tau : T \times 2^I \to T$ is a transition function, $\mathit{out} : T \to 2^O$ is the output function that labels each state with a set of output variables. Note that systems have no dead ends and have a transition for every input.

For the rest of the section, fix a system $M = (I, O, T, t_0, \tau, \mathit{out})$.

A *system path* is a sequence $t_1 t_2 ... \in T^\omega$ such that for every $i \in \mathbb{N}$ there is $e \in 2^I$ with $\tau(t_i, e) = t_{i+1}$. An *input-labeled system path* is a sequence $(t_1, e_1)(t_2, e_2)... \in (T \times 2^I)^\omega$ where $\tau(t_i, e_i) = t_{i+1}$ for every $i \in \mathbb{N}$. A *system trace starting from* $t_1 \in T$ is a sequence $(o_1 \cup e_1)(o_2 \cup e_2)... \in (2^{I \cup O})^\omega$ for which there exists an input-labeled system path $(t_1, e_1)(t_2, e_2)...$ and $o_i = out(t_i)$ for every $i \in \mathbb{N}$. Note that since systems are Moore, the output o_i cannot "react" to input e_i, the outputs are "delayed" with respect to inputs.

A *word automaton* A is a tuple $(\Sigma, Q, q_0, \delta, acc)$ where Σ is an alphabet, Q is a set of states, $q_0 \in Q$ is initial, $\delta : Q \times \Sigma \to 2^Q$ is a transition relation, $acc : Q^\omega \to \mathbb{B}$ is a path acceptance condition. Note that automata have no dead ends and have a transition for every letter of the alphabet.

For the rest of the section, fix automaton $A = (\Sigma, Q, q_0, \delta, acc)$ with $\Sigma = 2^{I \cup O}$.

A path in automaton A is a sequence $q_0 q_1 ... \in Q^\omega$ starting in the initial state such that there exists $a_i \in \Sigma$ for every $i \geq 0$ such that $(q_i, a_i, q_{i+1}) \in \delta(q_i)$. *A sequence* $a_0 a_1 \cdots \in \Sigma^\omega$ *generates a path* $\pi = q_0 q_1 \ldots$ iff for every $i \geq 0$: $(q_i, a_i, q_{i+1}) \in \delta$. A *path* π *is accepted* iff $acc(\pi)$ holds.

We distinguish two types of automata: universal and non-deterministic. The type defines when the automaton accepts a given infinite sequence.

A *non-deterministic automaton* A *accepts an infinite sequence* from Σ^ω iff there exists an accepted path generated by the sequence. Universal automata require *all* paths generated by the sequence to be accepted. For an automaton A, write $L(A)$ for the set of all infinite sequences accepted by A.

The *product* $M \times A$ is the automaton $(Q \times T, (q_0, t_0), \Delta, acc')$ such that for all $(q, t) \in Q \times T$: $\Delta(q, t) = \{(\delta(q, a \cup out(t)), \tau(q, a)) \mid a \in 2^I\}$. Define acc' to return true for a given $\pi \in (Q \times T)^\omega$ iff acc returns true for the corresponding projection of π into Q. Note that $M \times A$ has the 1-letter alphabet (not shown in the tuple).

We distinguish between two *path quantifiers*, E and A: $M \models \mathsf{E}(A)$ iff there is a system trace $(o_0 \cup e_0)(o_1 \cup e_1)...$ accepted by the automaton; $M \models \mathsf{A}(A)$ iff every system trace is accepted by the automaton.

Finally, we define different acceptance conditions. For a given infinite sequence $\pi \in Q^\omega$, let $Inf(\pi)$ be the elements of Q appearing in π infinitely often and let $Fin(\pi) = Q \setminus Inf(\pi)$. Then:

- *Büchi acceptance* is defined by a set $F \subseteq Q$: $acc(\pi)$ holds iff $Inf(\pi) \cap F \neq \emptyset$.
- *Co-Büchi acceptance* is defined by a set $F \subseteq Q$: $acc(\pi)$ holds iff $F \subseteq Fin(\pi)$.
- *Streett acceptance* is defined by pairs $\{(A_i \subseteq Q, G_i \subseteq Q)\}_{i \in [k]}$: $acc(\pi)$ holds iff $\forall i \in [k] : Inf(\pi) \cap A_i \neq \emptyset \to Inf(\pi) \cap G_i \neq \emptyset$.
- *Rabin acceptance* is defined by pairs $\{(F_i, I_i)\}_{i \in [k]}$: $acc(\pi)$ holds iff $\exists i \in [k] : F_i \subseteq Fin(\pi) \wedge Inf(\pi) \cap I_i \neq \emptyset$.
- *Parity acceptance* is defined by a priority function $p : Q \to [0, k]$: $acc(\pi)$ holds iff the minimal priority appearing infinitely often in $p(\pi)$ is even.

In addition to the above acceptance conditions, we define generalized versions. Generalized Büchi acceptance condition is defined by a set $\{F_i\}_{i \in [k]}$: $acc(\pi)$

holds iff the Büchi condition holds wrt. every F_i where $i \in [k]$. Similarly define Generalized co-Büchi, Streett, Rabin, and Parity conditions.

Abbreviations. We use the standard three letter abbreviation for automata $\{U, N, D\} \times \{B, C, S, P, R\} \times \{W\}$. For example, NBW means Nondeterministic Büchi Word automaton, UCW — Universal co-Büchi Word automaton.

3 Synthesis from Büchi, Streett, Rabin, Parity Automata

In this section we describe how to verify and synthesize properties described by Büchi, co-Büchi, Parity, Streett, and Rabin conditions. For each acceptance condition $X \in \{\text{Büchi, co-Büchi, Parity, Streett, Rabin}\}$, we can handle the question whether (the word defined by) some path of a system is in the language of a nondeterministic X automata, as well as the question of whether all paths of the system are in the language defined by a universal X automaton. There does not appear to be an easy way to mix these queries ("do all paths of the system fulfill the property defined by a given nondeterministic automaton?").

3.1 Ranking Functions

In the following, given a system $M = (I, O, T, t_0, \tau, out)$ and a nondeterministic (universal) automaton $A = (2^{I \cup O}, Q, q_0, \delta, acc)$, we will describe how to build an SMT query $\Phi_{M,A}$ that is satisfiable iff some path (all paths, resp.) of M are in $L(A)$. That is, we focus on the verification problem. When the verification problem is solved, we obtain the solution to the synthesis problem easily, following the Bounded Synthesis approach: Given an automaton A, we ask the SMT solver whether there is a system M such that $\Phi_{M,A}$ is satisfiable. More precisely, for increasing k, we fix a set of k states and ask the SMT solver for a transition relation τ and a labeling out for which $\Phi_{M,A}$ is satisfiable.

For the following, fix a system $M = (I, O, T, t_0, \tau, out)$ and an automaton $A = (2^{I \cup O}, Q, q_0, \delta, acc)$.

Our constructions use ranking functions. A *ranking function* is a function $\rho : Q \times T \to D$ for some well-founded domain D. A *rank comparison relation* is a relation $\triangleright \subseteq Q \times D \times D$. In the following, we write $\rho(q,t) \triangleright_q \rho(q',t')$ to mean $(q, \rho(q,t), \rho(q',t')) \in \triangleright$.

We will first establish how to use these to check existential and universal properties and then define the ranking functions for the different acceptance conditions. Given a rank comparison function $\triangleright$, we define the following formula to check an existential property:

$$\Phi_E^{\triangleright}(M,A) = rch(q_0,t_0) \wedge \bigwedge_{q,t \in Q \times T} rch(q,t) \rightarrow \bigvee_{(q,i \cup o,q') \in \delta} out(t) = o \wedge rch(q', \tau(t,i)) \wedge \rho(q,t) \triangleright_q \rho(q', \tau(t,i)).$$

Similarly, to check universal properties, we define

$$\Phi^{\triangleright}_A(M,A) = rch(q_0,t_0) \wedge \bigwedge_{q,t \in Q \times T} rch(q,t) \rightarrow \bigwedge_{(q,i \cup o,q') \in \delta} out(t) = o \rightarrow rch(q', \tau(t,i)) \wedge \rho(q,t) \triangleright_q \rho(q', \tau(t,i)).$$

In these formulas,

- the free variable $rch : Q \times T \rightarrow \mathbb{B}$ is an uninterpreted function that marks reachable states in the product of M and A, and
- the free variable $\rho : Q \times T \rightarrow \mathbb{N}$ is an uninterpreted ranking function.

Intuitively, $\Phi^{\triangleright}_E$ encodes that there is an accepting loop in the product automaton, while $\Phi^{\triangleright}_A$ ensures that all loops are accepting.

Given a path $\pi = (q_1,t_1)(q_2,t_2)\cdots \in (Q \times T)^\omega$ and a rank comparison relation $\triangleright$, we say that π *satisfies* $\triangleright$, denoted by $\pi \models \triangleright$, iff there exists a ranking function ρ such that $\rho(q_i,t_i) \triangleright_q \rho(q_{i+1},t_{i+1})$ holds for every $i > 0$.

Let us look at the properties of these equations.

Lemma 1. *For any rank comparison relation $\triangleright$: $\Phi^{\triangleright}_E(M,A)$ is satisfiable iff $M \times A$ has an infinite path $\pi = (q_1,t_1)(q_2,t_2)\ldots$ that satisfies $\triangleright$.*

Proof idea. Direction $\Leftarrow$. Assume that the product contains a path $\pi = (q_1,t_1)\ldots$ such that $\pi \models \triangleright$. By definition of $\pi \models \triangleright$, there is ρ such that $\rho(q_i,t_i) \triangleright \rho(q_{i+1},t_{i+1})$ holds for every $i > 0$. If we use the same ρ for $\Phi^{\triangleright}_E(M,A)$ and set $rch(q,t)$ to true for $(q,t) \in \pi$ and to false for the other states, then the query is satisfied.

Direction $\Rightarrow$. Assume that $\Phi^{\triangleright}_E$ is satisfied. We use the ranking function ρ from the model of $\Phi^{\triangleright}_E$ to construct a lasso-shaped infinite path that satisfies $\triangleright$. $\square$

Let *acc* be an acceptance condition on $(Q \times T)^\omega$. We say that $\triangleright$ *expresses an acceptance condition acc* iff

$$\forall \pi \in (Q \times T)^\omega : \quad \pi \models \triangleright \quad \leftrightarrow \quad acc(\pi).$$

In words: (1) if an infinite path $\pi = (q_1,t_1)\ldots$ is accepted by *acc*, then there is $\rho : Q \times T \rightarrow \mathbb{N}$ such that $\rho(q_i,t_i) \triangleright_q \rho(q_{i+1},t_{i+1})$ holds for all i; (2) if for an infinite path there is ρ such that $\rho(q_i,t_i) \triangleright_q \rho(q_{i+1},t_{i+1})$ holds for all i, then the path is accepted by *acc*.

The following theorem connects the previous lemma with automata.

Theorem 1 (Soundness and completeness for $\mathsf{E}(\boldsymbol{NXW})$). *Given a nondeterministic X automaton A and a system M. Let $\triangleright$ express acc. Then: $\Phi^{\triangleright}_E(M,A)$ is satisfiable iff $M \models \mathsf{E}(A)$.*

The formula $\Phi^{\triangleright}_A$ has similar properties, but in general only direction $\Rightarrow$ holds. The other direction we prove separately for each $\triangleright$ presented in the paper.

Lemma 2. *Given a system M, an automaton A, and any $\triangleright$: if $\Phi^{\triangleright}_A(M,A)$ is satisfiable, then all paths of $M \times A$ satisfy $\triangleright$.*

Proof idea. Assume that $\Phi^{\triangleright}_A(M, A)$ is satisfied by ranking function ρ. Consider an arbitrary path π in $M \times A$. By the deinfition of $\Phi^{\triangleright}_A$, *rch* holds for every state of π and $\triangleright$ holds for every two consecutive states. Thus, π satisfies $\triangleright$. □

Theorem 2 (Soundness for $\mathsf{A}(\boldsymbol{UXW})$). *Given a universal X automaton $A = (2^{I \cup O}, Q, q_0, \delta, acc)$ and a system $M = (I, O, T, t_0, \tau, out)$, let $\triangleright$ express acc. Then: if $\Phi^{\triangleright}_A(M, A)$ is satisfiable, then $M \models \mathsf{A}(A)$.*

3.2 Encoding for Büchi Automata

In the next sections, we describe rank comparison relations $\triangleright$ for the acceptance conditions Büchi, co-Büchi, Streett, Rabin, and Parity. We prove correctness only for Streett acceptance — the cases of Büchi, co-Büchi, and Parity follows as special cases. The correctness of $\triangleright$ constraints for Rabin acceptance follows from the work of Piterman et al. [21]. For didactical purposes, let us start with the relatively simple Büchi and co-Büchi conditions.

Büchi conditions were also presented in [8] and implicitly in [3]. Given a Büchi automaton $A = (2^{I \cup O}, Q, q_0, \delta, F)$, we define the rank comparison relation $\triangleright^A_B$ as

$$\rho(q,t) \triangleright^A_B \rho(q',t') = \begin{cases} true & \text{if } q \in F \\ \rho(q,t) > \rho(q',t') & \text{if } q \notin F \end{cases} \tag{1}$$

We have that $\triangleright_B$ expresses the Büchi condition. Intuitively, given an infinite path, we can use the natural numbers as the ranking domain and associate each state with the distance to the next acceptance condition on the path. If the path does not contain infinitely many accepting states, this ranking is not possible for a well-founded domain.

3.3 Encoding for co-Büchi Automata

This case was presented in the original paper [27] on Bounded Synthesis. Given a co-Büchi automaton $A = (2^{I \cup O}, Q, q_0, \delta, F)$, the ranking constraint relation $\triangleright^A_C$ for co-Büchi is defined as

$$\rho(q,t) \triangleright^A_C \rho(q',t') = \begin{cases} \rho(q,t) > \rho(q',t) & \text{if } q \in F \\ \rho(q,t) \geq \rho(q',t') & \text{if } q \notin F. \end{cases} \tag{2}$$

We have that $\triangleright_C$ expresses the co-Büchi condition. Intuitively, a ranking of a path with infinitely many rejecting states is not possible, because the domain does not have an infinitely descending chain. If the number is finite, we can label each state with the number of rejecting states that are yet to come.

3.4 Encoding for Streett Automata

Given a Streett automaton $A = (2^{I \cup O}, Q, q_0, \delta, \{(A_i, G_i)\}_{i \in [k]})$, we take the domain $D = \mathbb{N}$ and define the rank comparison relation $\triangleright_S = \bigwedge_i \rho(q,t) \triangleright^{A,i}_S \rho(q',t')$,

where

$$\rho(q,t) \rhd_S^{A,i} \rho(q',t') = \begin{cases} true & \text{if } q \in G_i \\ \rho_i(q,t) > \rho_i(q',t') & \text{if } q \in A_i \wedge q \notin \mathsf{G}_i \\ \rho_i(q,t) \geq \rho_i(q',t') & \text{if } q \notin A_i \cup G_i \end{cases} \tag{3}$$

Theorem 3. *The rank comparison relation $\rhd_S$ expresses the Streett condition.*

In Section 3.1, we have shown that we can check for the existence of a path satisfying an acceptance condition if we are given a rank comparison relation that expresses the condition. We have also shown the soundness of the construction for checking whether all paths satisfy the condition. It remains to show that this construction is complete.

Theorem 4. *Given a Streett automaton $A = (2^{I \cup O}, Q, q_0, \delta, \{(A_i, G_i)\}_{i \in [k]})$ and a system $M = (I, O, T, t_0, \tau, out)$: if $M \models \mathsf{A}(A)$, then $\Phi_A^{\rhd_S^A}(M, A)$ is satisfiable.*

Proof. Let $M \models \mathsf{A}(A)$. We construct a model (ρ and rch) that satisfies the query. Let $rch(q,t) = \mathsf{true}$ iff $(q,t) \in Q \times T$ is reachable in the product $\Gamma = M \times A$, and let $\rho_i(q,t) = 0$ for every unreachable $(q,t) \in Q \times T$. Now let us remove all unreachable states from Γ. Then for each $i \in [k]$, ρ_i is defined as follows.

- For every $(q,t) \in \cup_{i \in [k]} G_i \times T$, let $\rho_i(q,t) = 0$.
- Define an *SCC S of a graph* to be any maximal subset of the graph states such that for any $s \in S$, $s' \in S$, the graph has a path $\pi = s, ..., s'$ of length ≥ 2, where the length is the number of states appearing on the path. Thus, a single-state SCC can appear only if the state has a self-loop.
- Remove all outgoing edges from every state (q,t) of Γ with $q \in G_i$. The resulting graph Γ' has no SCCs that have a state (q,t) with $q \in \cup_{i \in [k]} A_i$.
- Let us define the graph Γ''. Let $\mathcal{S}$ be the set of all SCCs of Γ'. Then Γ'' has the states $V_{\Gamma''} = \mathcal{S} \cup \{\{s\} \mid s \notin \cup_{S \in \mathcal{S}} S\}$, i.e., each state is either an SCC or a singleton-set containing a state outside of any SCC (but in both cases, a state of Γ' is a set of states of Γ). The edges $E_{\Gamma''}$ of Γ'' are: $(S_1, S_2) \in E_{\Gamma''}$ iff $\exists s_1 \in S_1, s_2 \in S_2 : S_1 \neq S_2 \wedge (s_1, s_2) \in E_{\Gamma'}$. Intuitively, Γ'' is a graph derived from Γ by turning all accepting states into leafs, and by making SCCs the new states. Note that the graph Γ'' is a DAG.
- Given a path $\pi = S_1, ..., S_m$ in Γ'', let $nb(\pi)$ be the number of "bad" states visited on the path, i.e., $nb = |\pi \cap \{\{(q,t)\} : q \in \cup_{i \in [k]} A_i\}|$. Such a number exists since all paths of Γ'' are finite.
- For all $(q,t) \in S \in V_{\Gamma''}$ with $q \notin G_i$, let $\rho_i(q,t)$ be the max number of "bad" states visited on any path from S: $\rho_i(q,t) = max(\{nb(\pi) \mid \pi \textit{ is a path from } S\})$. Such a number exists since the number of paths in Γ'' is finite. □

Remark 1 (Comparison with ranking from [21]). Piterman et al. [21] introduced ranking functions to solve Streett *games*. Our ranking functions can be adapted to solve games, too. (Recall that our SMT encoding describes *model checking* with an uninterpreted system.) It may seem that in the case of games, our

construction uses fewer counters than [21], but that is not the case. Given a DSW with k Streett pairs and n states, a winning strategy in the corresponding Streett game may require a memory of size $k!$. In this case, the size of system$\times$automaton is $k!n$. Our construction introduces $2k$ counters with the domain $[k!n] \to [k!n]$ to associate a rank with each state. In contrast, [21] introduces $k!k$ counters with the domain $[n] \to [0, n]$. Encoding these counters into SAT would require $2k \cdot k!n \cdot log_2(k!n)$ bits for our construction, and $k!k \cdot n \cdot log_2(n)$ bits for the construction of [21]. Thus, our construction introduces $2(1 + log_2(k!)/log_2(n))$ times more bits. On the positive side, our construction is much simpler.

3.5 Encoding for Parity Automata

Given a Parity automaton $A = (2^{I \cup O}, Q, q_0, \delta, p)$ with indices $0, \ldots, k-1$, it is well known that we can translate it into the Streett automaton with pairs $(A_1, G_1), \ldots, (A_{m/2}, G_{m/2})$, where $A_i = \{q \mid p(q) = 2i - 1\}$, $G_i = \{q \mid p(q) \in \{0, 2, \ldots, 2i-2\}\}$. We can then apply the encoding for Streett automata. The resulting ranking is essentially Jurdziński's progress measure [14].

3.6 Encoding for Rabin Automata

Given a Rabin automaton $A = (2^{I \cup O}, Q, q_0, \delta, \{F_i, I_i\}_{i \in [k]})$ and a system $M = (I, O, T, t_0, \tau, out)$, we use ranking contraints described by Piterman et al. [21] to construct a rank comparison relation. We define the ranking domain D to consist of tuples of numbers $(b,\ j_1, d_1, \ldots, j_k, d_k)$, where $\rho(q,t)$ has the following meaning. For each $l \in [k]$,

- $j_l \in [k]$ is the index of a Rabin pair,
- $b \in [0, |Q \times T|]$ is an upper bound on the number of times the set F_{j_1} can be visited from (q,t),
- $d_l \in [0, |Q \times T|]$ is the maximal distance from (q,t) to the set I_{j_l},

We define $\rhd_R^A$ as $\rho(q,t) \rhd_q \rho(q',t')$ iff there exists $l \in [k]$ such that one of the following holds:

$$\begin{array}{rcl} b > b', & & \\ (b, \ldots, j_{l-1}, d_{l-1}) = (b', \ldots, j'_{l-1}, d'_{l-1}) & \wedge \quad j_l > j'_l & \wedge \quad q \notin \bigcup\limits_{m \in [l-1]} F_{j_m}, \\ (b, \ldots, j_l) = (b', \ldots, j'_l) & \wedge \quad d_l > d'_l & \wedge \quad q \notin \bigcup\limits_{m \in [l]} F_{j_m} \\ (b, \ldots, j_l) = (b', \ldots, j'_l) & \wedge \quad q \in I_{j_l} & \wedge \quad q \notin \bigcup\limits_{m \in [l]} F_{j_m} \end{array} \tag{4}$$

Here is the intutition. The first line bounds the number of visits to F_{j_1} (b decreases each time F_{j_1} is visited). The second line limits the changes of order $j_1, \ldots, j_k$ in rank b, $j_1, d_1, \ldots, j_k, d_k$ to a finite number. Together, these two lines ensure that on any path some F_m is not visited infinitely often. The third and fourth lines require I_{j_l} to be visited within d_l steps; once it is visited, the distance d_l can be reset to any number $\leq |Q \times T|$.

We can encode rank comparison constraints $\triangleright$ in Eq. 4 into SMT as follows. For each of $j_1 \dots j_k$ introduce an uninterpreted function: $Q \times T \to [k]$. For each of $b, d_1, \dots, d_k$ introduce an uninterpreted function: $Q \times T \to [0, |Q \times T|]$. Finally, replace in Eq. 4 counters b, j, d, b', j', d' with expressions $b(q,t)$, $j(q,t)$, $d(q,t)$, $b(q',t')$, $j(q',t')$, $d(q',t')$ resp.

3.7 Generalized Automata

The extension to generalized automata is simple: replace $\rho(q,t) \triangleright \rho(q',t')$ with $\bigwedge_i \rho_i(q,t) \triangleright \rho_i(q',t')$ where ρ_i describes the ranking of ith automaton component. Note that all components use the same *rch* variables.

4 Synthesis for CTL*

We describe two ways to encode model checking for CTL* into SMT. The first one, direct encoding (Sec. 4.2), resembles bottom-up CTL* model checking [9]. The second encoding (Sec. 4.3) follows the automata-theoretic approach [18] and goes via hesitant tree automata. As usual, replacing a concrete system function with an uninterpreted one of a fixed size gives a bounded synthesis procedure.

Let us compare the approaches. In the direct encoding, the main difficulty is the procedure that generates the constraints: we need to walk through the formula and generate constraints for nondeterministic Büchi or universal co-Büchi subformulas. In the approach via hesitant tree automata, we first translate a given CTL* into a hesitant tree automaton A, and then encode the non-emptiness problem of the product of A and the system into an SMT query. In contrast to the direct encoding, the difficult part is to construct the automaton, while the definition of the rank comparison relation is very easy.

In the next section we define CTL* with inputs and then describe two approaches. The approaches are conceptually the same, thus automata fans are invited to read Sec. 4.3 about the approach using hesitant automata, while the reader who prefers bottom-up CTL* model checking is welcomed to Sec. 4.2.

4.1 CTL* with Inputs

For this section, fix a system $M = (I, O, T, t_0, \tau, \mathit{out})$.

Below we define CTL* with inputs. The definition is slightly unusual (it differentiates inputs and outputs, see Remark 2) and is specific to Moore machines.

Syntax of CTL* with inputs. *State formulas* have the grammar:

$$\Phi = \mathsf{true} \mid \mathsf{false} \mid o \mid \neg o \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathsf{A}\varphi \mid \mathsf{E}\varphi$$

where $o \in O$ and φ is a path formula. *Path formulas* are defined by the grammar:

$$\varphi = \Phi \mid i \mid \neg i \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \varphi\,\mathsf{R}\,\varphi,$$

where $i \in I$. The temporal operators G and F are defined as usual.

The above grammar describes the CTL* formulas in the positive normal form. The general CTL* formula (in which negations can appear anywhere) can be converted into the formula of this form with no size blowup, using equivalence $\neg(a \,\mathsf{U}\, b) \equiv \neg a \,\mathsf{R}\, \neg b$.

Semantics of CTL* with inputs. We define the semantics of CTL* with respect to a system M. The definition is very similar to the standard one [2], except for a few cases involving inputs (marked with "+").

Let $t \in T$, and $o \in O$. Then:

- $t \not\models \Phi$ iff $t \models \Phi$ does not hold
- $t \models \mathsf{true}$ and $t \not\models \mathsf{false}$
- $t \models o$ iff $o \in out(t)$, $t \models \neg o$ iff $o \notin out(t)$
- $t \models \Phi_1 \wedge \Phi_2$ iff $t \models \Phi_1$ and $t \models \Phi_2$. Similarly for $\Phi_1 \vee \Phi_2$.
- \+ $t \models \mathsf{A}\varphi$ iff for all *input-labeled* system paths π starting from t: $\pi \models \varphi$. For $\mathsf{E}\varphi$, replace "for all" with "there exists".

Let $\pi = (t_1, e_1)(t_2, e_2)... \in (T \times 2^I)^\omega$ be an input-labeled system path and $i \in I$. For $k \in \mathbb{N}$, define $\pi_{[k:]} = (t_k, e_k)...$, i.e., the suffix of π starting from (t_k, e_k). Then:

- $\pi \models \Phi$ iff $t_1 \models \Phi$
- \+ $\pi \models i$ iff $i \in e_1$, $\pi \models \neg i$ iff $i \notin e_1$
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$. Similarly for $\varphi_1 \vee \varphi_2$.
- $\pi \models \mathsf{X}\varphi$ iff $\pi_{[2:]} \models \varphi$
- $\pi \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff $\exists l \in \mathbb{N} : (\pi_{[l:]} \models \varphi_2 \wedge \forall m \in [1, l-1] : \pi_{[m:]} \models \varphi_1)$
- $\pi \models \varphi_1 \,\mathsf{R}\, \varphi_2$ iff $(\forall l \in \mathbb{N} : \pi_{[l:]} \models \varphi_2) \vee (\exists l \in \mathbb{N} : \pi_{[l:]} \models \varphi_1 \wedge \forall m \in [1, l] : \pi_{[m:]} \models \varphi_2)$

A system M satisfies a CTL* state formula Φ, written $M \models \Phi$, iff the initial state satisfies it.

Note that $M \models i \wedge o$ is not defined, since $i \wedge o$ is not a state formula.

Remark 2 (Inputs vs. outputs). Let $r \in I$ and $g \in O$. According to the semantics, $\mathsf{E}r \wedge \mathsf{E}\neg r$ is valid, while $\mathsf{E}g \wedge \mathsf{E}\neg g$ ($\equiv g \wedge \neg g$) is unsatisfiable. Vice versa, $\mathsf{A}r \vee \mathsf{A}\neg r$ is unsatisfiable, while $\mathsf{A}g \vee \mathsf{A}\neg g$ ($\equiv g \vee \neg g$) is valid. This distinction is a consequence of the way we group inputs and outputs into traces.

4.2 Direct Encoding

We can encode CTL* model checking in SMT following the classical model checking approach [9] with the exception that system is described by uninterpreted functions.

Let Φ be a CTL* state formula. We define the SMT query as follows.

1) Replace each subformula f_i of Φ that starts with A or E with a new proposition p_i, and let $P = \{p_1, \ldots, p_k\}$. Thus, each p_i corresponds to some $\mathsf{E}/\mathsf{A}\varphi_i$ where φ_i is a path formula over $I \cup O \cup P$.

2) For each $f \in \{f_1, \ldots, f_k\}$, we do the following. If f is of the form $\mathsf{A}\varphi$, we translate φ into a UCW[1], otherwise into an NBW; let the resulting automaton be $A_\varphi = (2^{I \cup O \cup P}, Q, q_0, \delta, F)$. Then for all $(q,t) \in Q \times T$ the query contains the constraints:

   2a) If A_φ is an NBW, then:

$$rch(q,t) \;\rightarrow \bigvee_{(i,q') \in \delta(q, out(t), P(t))} rch(q',t') \wedge \rho(q,t) \rhd_B \rho(q',t')$$

   2b) If A_φ is a UCW, then:

$$rch(q,t) \;\rightarrow \bigwedge_{(i,q') \in \delta(q, out(t), P(t))} rch(q',t') \wedge \rho(q,t) \rhd_C \rho(q',t')$$

   In both cases, we have: $P(t) = \{p_i \in P \mid rch(q_0^{p_i}, t) = \mathsf{true}\}$, $q_0^{p_i}$ is the initial state of A_{φ_i}, $\rhd_B$ and $\rhd_C$ are the Büchi and co-Büchi rank comparison functions wrt. A_φ (see Eq. 1 and 2), and $t' = \tau(t,i)$. Intuitively, $P(t)$ under-approximates the subformulas that hold in t: if $p_i \in P(t)$, then $t \models f_i$.

3) Let $\widetilde{\Phi}$ be the top-level Boolean formula of Φ. Then the query contains the constraint $\widetilde{\Phi}[p_i \leftarrow rch(q_0^{p_i}, t_0)]$, where $q_0^{p_i}$ is the initial state of A_{φ_i}. For example, for $\Phi = \neg g \wedge \mathsf{AG}\,\mathsf{EF}\,\neg g$ the constraint is $\neg g(t_0) \wedge rch(q_0^{p_2}, t_0)$ where: $g \in O$, p_2 corresponds to $\mathsf{AG}\,p_1$, p_1 corresponds to $\mathsf{EF}\,\neg g$.

Theorem 5 (Correctness of direct encoding). *Given a CTL* formula Φ over inputs I and outputs O and a system $M = (I, O, T, t_0, \tau, out)$: $M \models \Phi$ iff the query is satisfiable.*

Here is the intuition behind the proof. The standard bottom-up model checker marks every system state with state subformulas it satisfies. The model checker returns "Yes" iff the initial state satisfies the top-level Boolean formula. The direct encoding conceptually follows that approach. If for some system state t, $rch(q_0^{p_i}, t)$ holds, then t satisfies the corresponding to p_i state formula f_i. Thus, if the top-level Boolean constraint (3) holds, then $t_0 \models \Phi$. And vice versa: if model checker returns "Yes", then the marking it produced can be used to satisfy the SMT constraints. Finally, the positive normal form of Φ allows us to get away with encoding of positive obligations only ($rch(q_0^{p_i}, t) \Rightarrow t \models f_i$), eliminating the need to encode $\neg rch(q_0^{p_i}, t) \Rightarrow t \models \neg f_i$.

Example 1. Let $I = \{r\}$, $O = \{g\}$, $\Phi = g \wedge \mathsf{AG}\,\mathsf{EF}\,\neg g$. We associate p_1 with $\mathsf{EF}\,\neg g$ and p_2 with $\mathsf{AG}\,p_1$. Automata for p_1 and p_2 are in Fig. 1, the SMT constraints are in Fig. 2.

[1] To translate φ into a UCW, translate $\neg\varphi$ into an NBW and treat it as a UCW.

(a) NBW for $\mathsf{F}\,\neg g$ (b) UCW for Gp_1

Fig. 1: Automata for Example 1

$$
\begin{aligned}
& g(t) \wedge rch(q_0, t_0) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(v_0, t) \wedge rch(q_0, t) \rightarrow rch(v_0, \tau(t, r)) \wedge \rho(v_0, t) \geq \rho(v_0, \tau(t, r)) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(v_0, t) \wedge \neg rch(q_0, t) \rightarrow rch(v_1, \tau(t, r)) \wedge \rho(v_0, t) \geq \rho(v_1, \tau(t, r)) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(v_1, t) \rightarrow rch(v_1, \tau(t, r)) \wedge \rho(v_1, t) > \rho(v_1, \tau(t, r)) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(q_0, t) \wedge g(t) \rightarrow rch(q_0, \tau(t, r)) \wedge \rho(q_0, t) > \rho(q_0, \tau(t, r)) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(q_0, t) \wedge \neg g(t) \rightarrow rch(q_1, \tau(t, r)) \wedge \rho(q_0, t) > \rho(q_1, \tau(t, r)) \\
\bigwedge_{t \in T, r \in \mathbb{B}} & \; rch(q_1, t) \rightarrow rch(q_1, \tau(t, r))
\end{aligned}
$$

Fig. 2: SMT constraints for Example 1

4.3 Encoding via Alternating Hesitant Tree Automata

A CTL* property can be converted into a hesitant tree automaton [18]. Then model checking a system with respect to a CTL* property is equivalent to checking non-emptiness of the product of the system and the hesitant automaton. The non-emptiness question can be reduced to solving a 1-Rabin game, and that is what the SMT query will express. The query will be satisfiable iff the product is non-empty. Below we define hesitant tree automata and describe the encoding.

Definitions. Intuitively, for a given CTL* formula, the alternating hesitant automaton expresses proof obligations encoded in (1)–(4) in Sec. 4.2, but in the form of an automaton. Thus, it is a mix of Büchi and co-Büchi automata.

Let $B^+(Q)$ be the set of all positive Boolean formulas over variables Q. Fix two disjoint finite sets, I and O. An *alternating hesitant tree automaton (AHT)* is a tuple $(\Sigma, D, Q, q_0, \delta, Acc)$, where $\Sigma = 2^O$, $D = 2^I$, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is the transition relation, $Acc \subseteq Q$ is the acceptance condition, and the following restrictions hold.

– Q can be partitioned into $Q^N_1, \ldots, Q^N_{k_N}$, $Q^U_1, \ldots, Q^U_{k_U}$, $Q^T_1, \ldots, Q^T_{k_T}$, where superscript N means nondeterministic, U means universal, and T means transient. Also, let $Q^N = \bigcup Q^N_i$, $Q^U = \bigcup Q^U_i$, and $Q^T = \bigcup Q^T_i$.
– There is a partial order on $\{Q^N_1, \ldots, Q^N_{k_N}, Q^U_1, \ldots, Q^U_{k_U}, Q^T_1, \ldots, Q^T_{k_T}\}$.
– The transition function δ satisfies: for any $q \in Q$, $a \in \Sigma$
  - if $q \in Q^T_i$, then: $\delta(q,a)$ contains no elements of Q^T_i; every element of $\delta(q,a)$ belongs to a lower set (with respect to the partial order);
  - if $q \in Q^N_i$, then: $\delta(q,a)$ contains only disjunctively related[1] elements of Q^N_i; every element of $\delta(q,a)$ outside of Q^N_i belongs to a lower set;
  - if $q \in Q^U_i$, then: $\delta(q,a)$ contains only conjunctively related[1] elements of Q^U_i; every element of $\delta(q,a)$ outside of Q^U_i belongs to a lower set.

We later define the acceptance of a system by such an automaton.

A *1-letter alternating hesitant word automaton (1-AHW)* is an AHW $(Q, q_0, \delta : Q \to \mathcal{B}^+(Q), Acc \subseteq Q)$. Its alphabet has only one letter (not shown in the tuple) and the automaton satisfies restrictions on δ and Q similar to those for AHTs.

A *run of a 1-AHW* $(Q, q_0, \delta : Q \to \mathcal{B}^+(Q), Acc \subseteq Q)$ is a labeled tree defined in a standard way. Its nodes are from Q^*, the root is q_0, the labeling l maps a node (in Q^*) to the last element (in Q), and for any reachable node, $l(succ(n)) \models \delta(l(n))$ where $l(succ(n))$ is the set of labels of the successors of node n. A *run is accepting* if all paths of the tree satisfy the acceptance condition. A run tree *path satisfies the acceptance condition Acc* iff one of the following holds:

– the corresponding path in the 1-AHW gets trapped in some Q^U and visits $Acc \cap Q^U$ only finitely often, or
– the corresponding path in the 1-AHW gets trapped in some Q^N and visits some state of $Acc \cap Q^N$ infinitely often.

Intuitively, the 1-AHW acceptance condition is a mix of Büchi and co-Büchi acceptance conditions. It can also be seen as a Rabin acceptance with one pair (F, I) where $F = Acc \cap Q^U$ and $I = (Acc \cap Q^N) \cup (Q^U \setminus Acc)$.

Note that any path of a run tree of a 1-AHW is trapped in some Q^N_i or Q^U_i.

The *non-emptiness question of 1-AHW* is "does the automaton has an accepting run?".

A *product of an AHT* $(2^O, 2^I, Q, q_0, \delta, Acc)$ *and a system* $(I, O, T, t_0, \tau, out)$ is a 1-AHW $(Q \times T, (q_0, t_0), \Delta, Acc')$ s.t. $Acc' = \{(q,t) \mid q \in Acc\}$ and for all (q,t):

$$\Delta(q,t) = \delta(q, out(t))[(d, q') \mapsto (\tau(t,d), q')].$$

A *system is accepting by an AHT* iff their product (1-AHW) is non-empty.

Encoding. Given a system M and a CTL* formula φ, we convert ϕ into an AHT A [18]. Then, we encode the non-emptiness of the 1-AHW $M \times A$ into an

[1] In a Boolean formula, atoms E are disjunctively (conjunctively) related iff the formula can be written into DNF (CNF) in such a way that each cube (clause) has at most one element from E.

SMT query:

$$rch(q_0, t_0) \wedge$$
$$\bigwedge_{q,t} rch(q,t) \rightarrow \delta(q, out(t))\ [(d, q') \ \mapsto\ rch(q', \tau(t,d)) \wedge \rho(q,t) \triangleright_{q,q'} \rho(q', \tau(t,d))] \tag{5}$$

where $\triangleright_{q,q'}$ is:

- if q and q' are in the same Q_i^N, then the Büchi rank comparison $\triangleright_B^{Q_i^N}$;
- if q and q' are in the same Q_i^U, then the co-Büchi rank comparison $\triangleright_C^{Q_i^N}$;
- otherwise, true.

Note: For ease of explanation, $\triangleright_{q,q'}$ depends on q *and* q', but it can also be defined to depend on q only. Intuitively, states from different Q_i correspond to different state subformulas of Φ.

Theorem 6. *Given a system $(I, O, T, t_0, \tau, out)$ and CTL* formula Φ over inputs I and outputs O: system $\models \Phi$ iff the SMT query in Eq. 5 is satisfiable.*

Proof idea. Direction $\Rightarrow$. Let (Q, q_0, δ, Acc) be the 1-AHW representing the product system$\times$AHT (AHT of Φ). We will use the following observation.

> Observation: *The 1-AHW non-emptiness can be reduced to solving the following 1-Rabin game.* The game states are Q, the game graph corresponds to δ, there is one Rabin pair (F, I) with $F = Acc \cap Q^U$, $I = (Acc \cap Q^N) \cup (Q^U \setminus Acc)$. Let us view δ to be in the DNF. Then, in state q of the game, the "existential" player (Automaton) chooses a disjunct in $\delta(q)$, while the "universal" player (Pathfinder) chooses a state in that disjunct. Automaton's strategy is winning iff for any Pathfinder's strategy the resulting play satisfies the Rabin acceptance (F, I). Note that Automaton has a winning strategy iff the 1-AHW is non-empty; also, memoryless strategies suffice for Automaton.

Since the 1-AHW is non-empty, Automaton has a memoryless winning strategy. We will construct rch and ρ from this strategy. For rch: set it to true if there is a strategy for Pathfinder such that the state will reached. Let us prove that ρ exists.

Since states from different Q_i can never form a cycle (due to the partial order), ρ of states from different Q_i are independent. Hence we consider two cases separately: ρ for some Q_i^N and for some Q_i^U.

- The case of Q_i^N is simple: by the definition of the 1-AHW, we can have only simple loops within Q_i^N. Any such reachable loop visits some state from $Acc \cap Q_i^N$. Consider such a loop: assign ρ for state q of the loop to be the minimal distance from any state $Acc \cap Q_i^N$.
- The case of Q_i^U: in contrast, we can have simple and non-simple loops within Q_i^U. But none of such loops visits $Acc \cap Q_i^U$. Then, for each $q \in Q_i^U$ assign ρ to be the maximum bad-distance from any state of Q_i^U. The bad-distance between q and q' is the maximum number of $Acc \cap Q^U$ states visited on any path from q to q'.

Direction $\Leftarrow$. The query is satisfiable means there is a model for *rch*. Note that the query is Horn-like ($\ldots \rightarrow \ldots$), hence there is a minimal marking *rch* of states that still satisfies the query[2][3]. Wlog., assume *rch* is minimal. Consider the subset of the states of the 1-AHW that are marked with *rch*, and call it U. Note that U is a 1-AHW and it has only universal transitions (i.e., we never mark more than one disjunct of δ on the right side of $\ldots \rightarrow \delta(\ldots)$). Intuitively, U represents a finite-state folding of the run tree of the original 1-AHW.

> Claim: *the run tree (the unfolding of U) is accepting.* Suppose it is not: there is a run tree path that violates the acceptance. Consider the case when the path is trapped in some Q_i^U. Then the path visits a state in $Q_i^U \cap Acc$ infinitely often. But this is impossible since we use co-Büchi ranking for Q_i^U. Contradiction. The case when the path is trapped in some Q_i^N is similar — the Büchi ranking prevents from not visiting $Acc \cap Q_i^N$ infinitely often.

Thus, the 1-AHW is non-empty since it has an accepting run (U unfolded). □

5 Prototype Synthesizer for CTL*

We implemented both approaches to CTL* synthesis described in Sections 4.2 and 4.3 inside the tool PARTY [15]: https://github.com/5nizza/party-elli (branch "cav17"). In this section we illustrate the approach via AHTs.

The synthesizer works as follows:

1) Parse the specification that describes inputs, outputs, and CTL* formula Φ.
2) Convert Φ into a hesitant tree automaton using the procedure described in [18], using LTL3BA [1] to convert path formulas into NBWs.
3) For each system size k in increasing order:
   - encode "$\exists M_k : M_k \times AHT \neq \emptyset$?" into SMT using Eq. 5 where $|M_k| = k$
   - call Z3 solver [11]: if the solver returns "unsatisfiable", goto next iteration; otherwise print the model in the dot graph format.

This procedure is complete, because there is a $2^{2^{|\Phi|}}$ bound on the size of the system, although reaching it is impractical.

Running example: resettable 1-arbiter. Let $I = \{r\}$, $O = \{g\}$. Consider a simple CTL* property of an arbiter

$$\mathsf{EG}(\neg g) \wedge \mathsf{AG}(r \rightarrow \mathsf{F}\, g) \wedge \mathsf{AG}\,\mathsf{EF}\,\neg g.$$

The property says: there is a path from the initial state where the system never grants (including the initial state); every request should be granted;

[2] Minimal in the sense that it is not possible to reduce the number of *rch* truth values by falsifying some of *rch*.

[3] Non-minimality appears when δ of the alternating automaton has OR and the SMT solver marks with *rch* more than one OR argument. Another case is when the solver marks some state with *rch* but there is no antecedent requiring that.

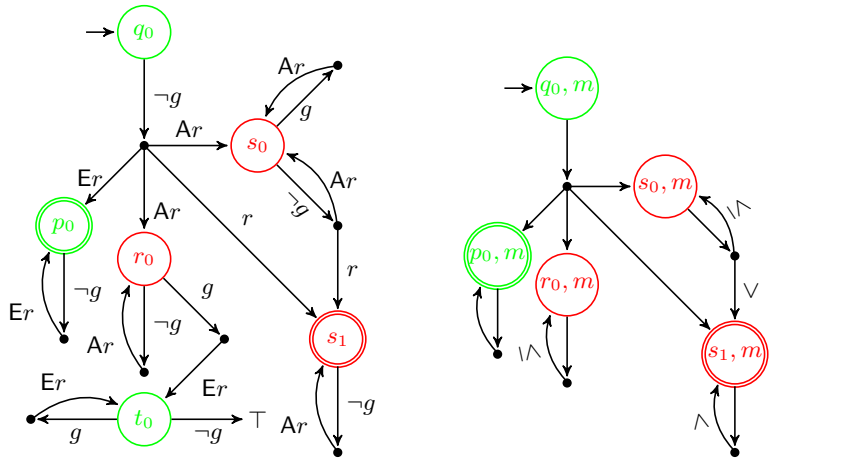
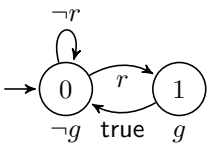


Fig. 3: On the left: AHT for the CTL* formula $\mathsf{EG}\,\neg g \wedge \mathsf{AG}\,\mathsf{EF}\,\neg g \wedge \mathsf{AG}(r \rightarrow \mathsf{F}\,g)$. Green states are from the nondeterministic partion, red states are from the universal partition, double states are final (a red final state is rejecting, a green final state is accepting). Falling out of red (universal) states is allowed, falling out of green (nondeterministic) states is not allowed. State $\top$ denotes an accepting state. In this automaton all transitions out of black dots are conjuncted. For example, $\delta(q_0, \neg g) = ((r, p_0) \vee (\neg r, p_0)) \wedge ((r, r_0) \wedge (\neg r, r_0)) \wedge (r, s_1) \wedge ((r, s_0) \wedge (\neg r, s_0))$. States s_0 and s_1 describe the property $\mathsf{AG}(r \rightarrow \mathsf{F}\,g)$, state p_0 — $\mathsf{EG}\,\neg g$, states r_0 and t_0 — $\mathsf{AG}\,\mathsf{EF}\,\neg g$, state t_0 — $\mathsf{EF}\,\neg g$.
On the right side is the product (1-AHW) of the AHT with the one state system that never grants (thus it has $m \xrightarrow{\mathsf{true}} m$ and $out(m) = \neg g$). The edges are labeled with the relation $\triangleright_{q,q'}$ defined in Eq. 5. The product has no plausible annotatation due to the cycle $(s_1, m) \xrightarrow{\geq} (s_1, m)$, thus the system does not satisfy the property.

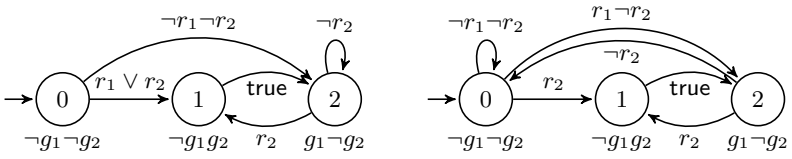
and finally, a state without the grant should always be reachable. We now invite the reader to Figure 3. It contains the AHT produced by our tool, and on its right side we show the product of the AHT with the one-state system that does not satisfy the property. The correct system needs at least two states and is on the right.



Resettable 2-arbiter. Let $I = \{r_1, r_2\}$, $O = \{g_1, g_2\}$. Consider the formula

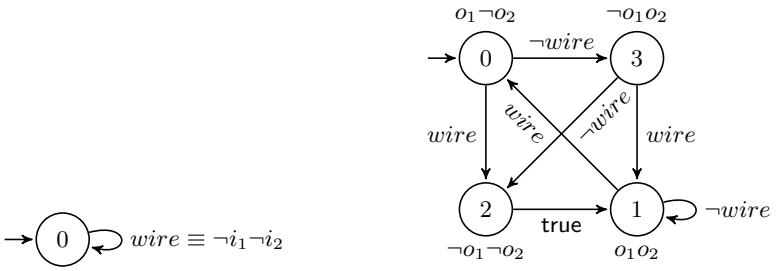
$$\mathsf{EG}(\neg g_1 \wedge \neg g_2) \ \wedge \mathsf{AG}\,\mathsf{EF}(\neg g_1 \wedge \neg g_2) \wedge$$
$$\mathsf{AG}(r_1 \rightarrow \mathsf{F}\,g_1) \wedge \mathsf{AG}(r_2 \rightarrow \mathsf{F}\,g_2) \wedge \mathsf{AG}(\neg(g_1 \wedge g_2)).$$

Note that without the properties with E, the synthesizer can produce the system in Figure 4a which starts in the state without grants and then always grants one or another client. Our synthesizer output the system in Figure 4b (in one second).



(a) The case without E properties (b) The case with E properties

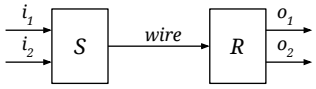
Fig. 4: Synthesized systems for the resettable arbiter example



(a) The sender (Mealy machine) (b) The receiver (Moore machine)

Fig. 5: The synthesized system for the sender-receiver example

Sender-receiver system. Consider a sender-receiver system of the following structure. It has two modules, the sender (S) with inputs $\{i_1, i_2\}$ and output *wire* and the receiver (R) with input *wire* and outputs $\{o_1, o_2\}$. The sender can send one bit over the wire to the receiver. We would like to synthesize the sender and receiver modules that satisfy the following CTL* formula over $I = \{i_1, i_2\}$ and $O = \{o_1, o_2\}$:



$$\mathsf{AG}((i_1 \wedge i_2) \rightarrow \mathsf{F}(o_1 \wedge o_2)) \wedge$$
$$\mathsf{AG}((i_1 \wedge i_2 \wedge o_1 \wedge o_2) \rightarrow \mathsf{X}(o_1 \wedge o_2)) \wedge$$
$$\mathsf{AG}(\ \mathsf{EF}(o_1 \wedge \neg o_2) \wedge \mathsf{EF}(\neg o_1 \wedge o_2) \wedge \mathsf{EF}(\neg o_1 \wedge \neg o_2) \wedge \mathsf{EF}(o_1 \wedge o_2)\ ).$$

Our tool does not support the distributed synthesis, so we manually adapted the SMT query it produced, by introducing the following uninterpreted functions.

- For the sender: the transition function $\tau_s : T_s \times 2^{\{i_1,i_2\}} \rightarrow T_s$ and the output function $out_s : T_s \times 2^{\{i_1,i_2\}} \rightarrow \mathbb{B}$. We set T_s to have a single state.
- For the receiver: the transition function $\tau_r : T_r \times 2^{\{wire\}} \rightarrow T_r$ and the output functions $o_1 : T_r \rightarrow \mathbb{B}$ and $o_2 : T_r \rightarrow \mathbb{B}$. We set T_r to have four states.

It took Z3 solver about 1 minute to find the solution shown in Figure 5.

6 Related Work and Conclusion

The closest is the work by Beyene et al. [4] on solving *infinite*-state games using SMT solvers. Conceptually, they use co-Büchi and Büchi ranking functions to encode game winning into SMT, which was also partially done by Schewe and Finkbeiner [27] a few years earlier (for finite-state systems). The authors focused on co-Büchi and Büchi automata, while we also considered Rabin and Streett automata (for finite-state systems). Although they claimed their approach can be extended to μ-calculus (and thus to CTL*), they did not elaborate beyond noting that CTL* verification can be reduced to games.

In this paper, we showed how the research on ranking functions [21,14] can be used to easily derive synthesis procedures. We also described two approaches to the CTL* synthesis and the only (to our knowledge) synthesizer supporting CTL*. (For CTL synthesis see [16,10,25], and [3] for PCTL.) The two approaches are conceptually similar. The approach via direct encoding is easier to code. The approach via alternating hesitant automata, for example, hints at how to reduce CTL* synthesis to solving safety games: via bounding the number of visits to co-Büchi final states and bounding the distance to Büchi final states, and then determinizing the resulting automaton. A possible future direction is to extend the approach to the logic ATL* and distributed systems.

Acknowledgements. We thank Swen Jacobs and Bernd Finkbeiner for early discussions on bounded synthesis for GR(1), Nir Piterman for explaining Streett/Rabin ranking constructions and alternating automata. This work was supported by the Austrian Science Fund (FWF) under the RiSE National Research Network (S11406).

References

1. Babiak, T., Kretínský, M., Rehák, V., Strejcek, J.: LTL to Büchi automata translation: Fast and more deterministic. In: TACAS. LNCS, vol. 7214, pp. 95–109. Springer (2012)
2. Baier, C., Katoen, J.P.: Principles of model checking, vol. 26202649. MIT press Cambridge (2008)
3. Bertrand, N., Fearnley, J., Schewe, S.: Bounded Satisfiability for PCTL. In: Cégielski, P., Durand, A. (eds.) CSL. LIPICS, vol. 16, pp. 92–106. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2012), http://drops.dagstuhl.de/opus/volltexte/2012/3666
4. Beyene, T., Chaudhuri, S., Popeea, C., Rybalchenko, A.: A constraint-based approach to solving games on infinite graphs. SIGPLAN Not. 49(1), 221–233 (Jan 2014), http://doi.acm.org/10.1145/2578855.2535860
5. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. Journal of Computer and System Sciences 78, 911–938 (2012)
6. Bloem, R., Braud-Santoni, N., Jacobs, S.: Synthesis of self-stabilising and byzantine-resilient distributed systems. In: Chaudhuri, S., Farzan, A. (eds.) CAV. Lecture Notes in Computer Science, vol. 9779, pp. 157–176. Springer (2016), http://dx.doi.org/10.1007/978-3-319-41528-4_9
7. Bloem, R., Chatterjee, K., Jacobs, S., Könighofer, R.: Assume-guarantee synthesis for concurrent reactive programs with partial information. In: Baier, C., Tinelli, C.

(eds.) TACAS. pp. 517–532. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-46681-0_50
8. Bloem, R., Chockler, H., Ebrahimi, M., Strichman, O.: Synthesizing non-vacuous systems. In: Bouajjani, A., Monniaux, D. (eds.) VMCAI. pp. 55–72. Springer International Publishing, Cham (2017), http://dx.doi.org/10.1007/978-3-319-52234-0_4
9. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986), http://doi.acm.org/10.1145/5397.5399
10. De Angelis, E., Pettorossi, A., Proietti, M.: Synthesizing concurrent programs using answer set programming. Fundamenta Informaticae 120(3-4), 205–229 (2012)
11. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. pp. 337–340 (2008)
12. Filiot, E., Jin, N., Raskin, J.: Antichains and compositional algorithms for LTL synthesis. Formal Methods in System Design 39(3), 261–296 (2011), http://dx.doi.org/10.1007/s10703-011-0115-3
13. Jacobs, S., Bloem, R.: Parameterized synthesis. In: Flanagan, C., König, B. (eds.) TACAS. LNCS, vol. 7214, pp. 362–276. Springer (2012)
14. Jurdziński, M.: Small progress measures for solving parity games. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 290–301. Springer (2000)
15. Khalimov, A., Jacobs, S., Bloem, R.: Party parameterized synthesis of token rings. In: International Conference on Computer Aided Verification. pp. 928–933. Springer (2013)
16. Klenze, T., Bayless, S., Hu, A.J.: Fast, flexible, and minimal ctl synthesis via smt. In: International Conference on Computer Aided Verification. pp. 136–156. Springer (2016)
17. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS. pp. 531–542 (2005)
18. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM 47(2), 312–360 (Mar 2000), http://doi.acm.org/10.1145/333979.333987
19. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. In: Kozen, D. (ed.) Logics of Programs, Workshop, Yorktown Heights, New York, May 1981. Lecture Notes in Computer Science, vol. 131, pp. 253–281. Springer (1981), http://dx.doi.org/10.1007/BFb0025786
20. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: LICS. pp. 255–264. IEEE Computer Society (2006), http://dx.doi.org/10.1109/LICS.2006.28
21. Piterman, N., Pnueli, A.: Faster solutions of Rabin and Streett games. In: 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings. pp. 275–284 (2006), http://dx.doi.org/10.1109/LICS.2006.23
22. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57. IEEE (1977)
23. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989. pp. 179–190. ACM Press (1989), http://doi.acm.org/10.1145/75277.75293
24. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri,

USA, October 22-24, 1990, Volume II. pp. 746–757. IEEE Computer Society (1990), http://dx.doi.org/10.1109/FSCS.1990.89597
25. Prezza, N.: Ctl (computation tree logic) sat solver, https://github.com/nicolaprezza/CTLSAT
26. Safra, S.: On the complexity of omega-automata. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988. pp. 319–327. IEEE Computer Society (1988), http://dx.doi.org/10.1109/SFCS.1988.21948
27. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Automated Technology for Verification and Analysis (ATVA'07). pp. 474–488 (2007)
28. Sohail, S., Somenzi, F.: Safety first: a two-stage algorithm for the synthesis of reactive systems. STTT 15(5-6), 433–454 (2013), http://dx.doi.org/10.1007/s10009-012-0224-3
29. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. 115(1), 1–37 (1994), http://dx.doi.org/10.1006/inco.1994.1092
30. Wolper, P., Vardi, M.Y., Sistla, A.P.: Reasoning about infinite computation paths (extended abstract). In: 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983. pp. 185–194. IEEE Computer Society (1983), http://dx.doi.org/10.1109/SFCS.1983.51