Fully Generalized Reactivity(1) Synthesis[⋆]

Rüdiger Ehlers and Ayrat Khalimov(✉)

TU Clausthal, Clausthal-Zellerfeld, Germany
{rudiger.ehlers,ayrat.khalimov}@tu-clausthal.de

Abstract. Generalized Reactivity(1) (GR(1)) synthesis is a reactive synthesis approach in which the specification is split into two parts: a symbolic game graph, describing the safe transitions of a system, a liveness specification in a subset of Linear Temporal Logic (LTL) on top of it. Many specifications can naturally be written in this restricted form, and the restriction gives rise to a scalable synthesis procedure – the reasons for the high popularity of the approach. For specifications even slightly beyond GR(1), however, the approach is inapplicable. This necessitates a transition to synthesizers for full LTL specifications, introducing a huge efficiency drop. This paper proposes a synthesis approach that smoothly bridges the efficiency gap from GR(1) to LTL by unifying synthesis for both classes of specifications. The approach leverages a recently introduced canonical representation of omega-regular languages based on a chain of good-for-games co-Büchi automata (COCOA). By constructing COCOA for the liveness part of a specification, we can then build a fixpoint formula that can be efficiently evaluated on the symbolic game graph. The COCOA-based synthesis approach outperforms standard approaches and retains the efficiency of GR(1) synthesis for specifications in GR(1) form and those with few non-GR(1) specification parts.

1 Introduction

Reactive synthesis is the process of automatically computing a provably correct reactive system from its formal specification [13]. A safety-critical system is often developed twice: first, when it is described using a formal specification, and second, when a system is implemented according to this specification. The dream of reactive synthesis is to fully eliminate manual implementation phase.

Reactive synthesis is however computationally hard. For specifications in the commonly used linear temporal logic (LTL), checking whether an implementation exists is 2EXPTIME-complete [30]. The classical approach to solve reactive synthesis from LTL is to first translate the LTL formula into a deterministic parity automaton, followed by solving the induced two-player parity game [7]. The system player wins this game if and only if there is an implementation satisfying the specification. It is the first phase of translating LTL to parity automaton that usually represents a bottleneck. This observation spurred a series

[⋆] This work has been partially supported by the DFG through Grant No. 322591867 (GUISynth) and the BMWi through Grant No. 19A21026E (SafeWahr).

of synthesis approaches. For instance, in bounded synthesis, either the maximal number of states that a system can have [22] or the longest system response time [20] is restricted. If there exists a system realizing the specification, then there exists one that adheres to some bounds, and bounded synthesis works well whenever small bounds suffice for realizing the given specification. Another approach is to synthesize implementations for parts of the specification, and to then compose them into one that realizes the whole specification [25,31,21]. The approach of [26] avoids constructing one large deterministic parity automaton and instead constructs many smaller ones that—when composed together—represent the original specification. Such decomposition proved beneficial on practical examples [1]. Finally, there are approaches that consider "synthesis-friendly" subsets of LTL. Alur and La Torre identified a number of such LTL fragments with a simpler synthesis problem [3], and this eventually led to the introduction of Generalized Reactivity(1) synthesis by Piterman et al. [28], GR(1) for short. GR(1) synthesis gained a lot of prominence and was applied in domains such as robotics [34,24], cyber-physical system control [36,35], and chip component design [8,23]. We describe it in more detail.

In GR(1) synthesis, the specification is divided into two parts. The first part represents the *safety properties* of a system and encodes a symbolic game graph. Each graph vertex encodes a valuation of last system inputs and outputs. The transitions in the graph represent how these variables can evolve in one step. For instance, a robot on a grid can move from its current cell to the left, right, up, or down, but cannot jump; this is easily encoded as a symbolic game graph. Secondly, there are *liveness properties* of the following form: if certain vertices are visited infinitely often, then certain other vertices must be visited infinitely often as well. The liveness properties are encoded symbolically using LTL formulas of the shape $\bigwedge_i \mathsf{GF}\varphi_i \rightarrow \bigwedge_j \mathsf{GF}\psi_j$, where φ_i and ψ_j are Boolean formulas over input and output system propositions. Synthesis problems from many domains can be encoded naturally, or after some manual effort, into the GR(1) setting.

Constraining specifications to GR(1) form reduces the synthesis problem's complexity from doubly-exponential to singly-exponential (in the number of propositions), or polynomial when the number of propositions is fixed [8]. The GR(1) synthesis problem can be solved by evaluating a fixpoint formula on the symbolic game arena. The fixpoint formula defines the set of vertices from which the system player satisfies the GR(1) liveness properties while staying in the game arena. The simple shape of GR(1) liveness properties makes the fixpoint formula simple. Moreover, evaluating the fixpoint formula on the symbolic game graph can be done efficiently using Binary Decision Diagrams (BDDs, [12]) as the underlying data structure. These factors together — efficient implementation and relatively expressive specification language — made GR(1) synthesis popular.

GR(1) synthesis has a drawback. A single property outside of GR(1) – for instance, "eventually the robot always stays in some stable zone" ($\mathsf{FG}\ \mathit{inStableZone}$) – makes GR(1) synthesis inapplicable. Switching to full-LTL synthesizers introduces an abrupt efficiency drop, as they do not take advantage of the simple structure of GR(1)-like specifications. For improving the practical applicability of reactive synthesis, a synthesis approach exhibiting a smooth efficiency curve on the way from GR(1) to LTL would hence be useful. While there are are some GR(1) synthesis extensions (e.g., [4,17]), they only extend it by certain specification classes and consequently do not support full LTL.

This paper unifies synthesis for GR(1) and full LTL. Like in GR(1) synthesis, we aim at synthesis for specifications split into the safety part encoded as a symbolic game graph and the liveness part. Unlike the standard GR(1) synthesis, the liveness part can be any LTL or omega-regular property. For standard GR(1) specifications, our approach inherits the efficiency of GR(1) synthesis, including when a specification does not fall syntactically into this class, but is semantically a GR(1) specification. At the same time, for specifications that go beyond GR(1) and only have a few non-GR(1) components, our approach scales well.

Our solution is based on the same fixpoint-evaluation-of-symbolic-game-graph idea. Our starting point is a folklore approach based on solving parity games by evaluating fixpoint equations [11]. We modify it so that it becomes applicable to specifications given in the form of a chain of good-for-games co-Büchi automata (COCOA). Such chains have recently been proposed as a new canonical representation of omega-regular languages [19], and it has been shown how minimal and canonical COCOA can be computed in polynomial time from a deterministic parity automaton of the language. Our COCOA-based synthesis approach converts the liveness part of the specification into a parity automaton, constructs the chain, builds the fixpoint formula from the chain, and finally evaluates it on the symbolic game graph. We show that the fixpoint formula built from the chain has a structure similar to GR(1) fixpoint formulas. This is not the case for the folklore approach via parity games, and as a result, our COCOA-based synthesizer is roughly an order of magnitude faster. The COCOA-based synthesis approach inherits the efficiency of GR(1) synthesis, and it is also efficient on specifications slightly beyond GR(1). Finally, our approach is the first application of the new canonical representation of omega-regular languages.

2 Preliminaries

Automata and languages

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers including 0. Let AP be a set of *atomic propositions*; 2^{AP} denotes the *valuations* of these propositions. A Boolean formula represents a set of valuations: for instance, $\bar{a} \wedge b$, also written $\bar{a}b$, encodes valuations in which proposition a has value false and b is true. A Boolean function maps valuations of propositions to either true or false. Binary decision diagrams (BDDs) are a data structure for manipulating such functions.

A *word* is a sequence of proposition valuations $w = x_0x_1 \ldots \in (2^{\mathsf{AP}})^\omega \cup (2^{\mathsf{AP}})^*$. A word can be finite or infinite. A *language* is a set of infinite words. Given a language L, the *suffix language* of L for some finite word $p \in (2^{\mathsf{AP}})^*$ is $\mathcal{L}(L, p) = \{x_0x_1 \ldots \in (2^{\mathsf{AP}})^\omega \mid p \cdot x_0x_1 \ldots \in L\}$. The words in this set are called *suffix words*. The set of all suffix languages of L is the set $\{\mathcal{L}(L, p) \mid p \in (2^{\mathsf{AP}})^*\}$.

Automata over infinite words are used to finitely represent languages. We consider parity and co-Büchi automata with transition-based acceptance. A *parity automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ with a finite alphabet Σ (usually $\Sigma = 2^{\mathsf{AP}}$), a finite set of states Q, an initial state $q_0 \in Q$, and a finite transition relation $\delta \subseteq Q \times \Sigma \times Q \times \mathbb{N}$ satisfying $(q, x, q', c) \in \delta \Rightarrow (q, x, q', c') \notin \delta$ for all q, x, q' and $c' \neq c$. An automaton is *complete* if for every state q and letter x there exists at least one pair $(q', c) \in Q \times \mathbb{N}$ s.t. $(q, x, q', c) \in \delta$; it is *deterministic* if exactly one such pair (q', c) exists. Wlog. we assume that automata are complete. An automaton is *co-Büchi* if only colors 1 and 2 occur in δ, and then we call the transitions with color 1 *rejecting* and those with color 2 *accepting.*

A *run* of $\mathcal{A}$ on a word $w = x_0 x_1 \ldots \in \Sigma^\omega$ is a sequence $\pi = \pi_0 \pi_1 \ldots \in Q^\omega$ starting in $\pi_0 = q_0$ and such that $(\pi_i, x_i, \pi_{i+1}, c_i) \in \delta$ for some c_i for every $i \in \mathbb{N}$; the induced *color sequence* $c = c_0 c_1 \ldots$ is uniquely defined by w and π. A run is *accepting* if the lowest color occurring infinitely often in the induced color sequence is even ("min-even acceptance"). When this minimal color is uniquely defined, e.g. when there is only one accepting run, it is called *the color of w* wrt. $\mathcal{A}$. A word is *accepted* if it has an accepting run. The automaton's language $\mathcal{L}(\mathcal{A})$ is the set of accepted words. The language of the automaton $\mathcal{A}'$ derived from $\mathcal{A}$ by changing the initial state to q is denoted by $\mathcal{L}(\mathcal{A}, q)$.

A *co-Büchi language* is a language representable by a nondeterministic (equiv., deterministic) co-Büchi automaton. The Co-Büchi languages are a strict subset of the omega-regular languages.

An automaton is *good-for-games* if there exists a strategy $f : \Sigma^* \to Q$ to resolve the nondeterminism to produce accepting runs on the accepted words, formally: for every infinite word $w = x_0 x_1 \ldots$, the sequence $\pi_0 \pi_1 \ldots$ defined by $\pi_i = f(x_0 \ldots x_{i-1})$ for all $i \in \mathbb{N}$ is a run, *and* it is accepting whenever w belongs to the language.

Games and our realizability problem

LTL. A commonly used formalism to represent system specifications is *Linear Temporal Logic* (LTL, [29]). It uses temporal operators U, X, and derived ones G and F, which we do not define here. For details, we refer the reader to [27].

Games. An edge-labelled *game* is a tuple $G = (\mathsf{AP}_I, \mathsf{AP}_O, V, v_0, \delta, obj)$ where V is a finite set of vertices, $v_0 \in V$ is initial, $\delta : V \times 2^{\mathsf{AP}_I} \times 2^{\mathsf{AP}_O} \rightharpoonup V$ is a partial function describing possible moves (safety specification), and obj is a winning objective (liveness specification). A *play* is a maximal (finite or infinite) sequence of transitions of the form $(v_0, i_0, o_0, v_1)(v_1, i_1, o_1, v_2)(v_2, i_2, o_2, v_3)\ldots$; the corresponding sequence $(i_0 \cup o_0)(i_1 \cup o_1)\ldots$ is called the *action sequence.* An infinite play is winning for the system if it satisfies the objective obj; when obj is an LTL objective over $\mathsf{AP}_I \cup \mathsf{AP}_O$, the infinite play satisfies obj iff the action sequence satisfies it. A system *strategy* is a function $f : (2^{\mathsf{AP}_I})^+ \to 2^{\mathsf{AP}_O}$. The game is won by the system if it has a strategy f such that every play $(v_0, i_0, o_0, v_1)(v_1, i_1, o_1, v_2)\ldots$ is infinite and it satisfies the objective, where $o_j = f(i_0 \ldots i_j)$ for all j. To define parity games, the winning objective obj is set to be a parity-assigning function $obj : V \to \mathbb{N}$, and then an infinite play satisfies obj iff the minimal parity visited infinitely often in the sequence $obj(v_0)obj(v_1)\ldots$ is even (min-even acceptance on states).

The *enforceable predecessor operator* $\Box\Diamond$ reads a set of tuples $\Phi \subseteq 2^{\mathsf{AP}} \times V$ and returns the set of positions from which the system can enforce taking one of the transitions into the destination set:

$$\Box\Diamond(\Phi) = \{v \in V \mid \forall i.\exists o : (i \cup o, \delta(v, i, o)) \in \Phi\} \tag{1}$$

Symbolic games with LTL objectives. Games can be represented symbolically. For instance, the vertices can be encoded as valuations of Boolean variables AP, and transitions between the vertices can be encoded using a Boolean formula. This paper focuses on solving symbolic games with LTL objectives:

Given a symbolic game with LTL objective. Who wins the game?

The particular symbolic representation is not important as long as it provides the operations for union, intersection, and complementation of sets of label-position tuples, and the enforceable predecessor operator $\Box\Diamond$. This paper focuses exclusively on the realizability problem; the extraction of compact and efficient implementations merits a separate study.

Mu-calculus fixpoint formulas. For an introduction to using fixpoint formulas in synthesis, we refer the reader to [7], and to [10,5] for mu-calculus in general. The fixpoint formulas use the greatest (ν) and least (μ) fixpoint operators, and the enforceable-predecessor operator $\Box\Diamond$. For instance, the formula $\nu Y.\mu X.\Box\Diamond(Y \wedge (\overline{x} \vee X))$ represents the biggest set of vertices such that from all vertices in the set, the system can enforce that either x does not hold along the next transition and this transition leads back to the same set, or the play gets closer to a position from which this can be enforced. This formula hence characterizes the positions from which the system can enforce that $\overline{x}$ holds infinitely often along a play.

Generalized Reactivity(1)

Generalized Reactivity(1) is a class of assume-guarantee specifications that includes safety and liveness components. It gained popularity because many specifications naturally fall into the GR(1) class, and the restricted nature of GR(1) admits an efficient synthesis approach. For the purpose of this paper, we define a GR(1) specification as a game $G_{\text{gr1}} = (\mathsf{AP}_I, \mathsf{AP}_O, V, v_0, \delta, \Phi)$ with an LTL winning objective of the form $\Phi = \bigwedge_{i=1}^{m} \mathsf{GF} a_i \to \bigwedge_{j=1}^{n} \mathsf{GF} g_j$, where each assumption a_i and guarantee g_j are Boolean formulas over $\mathsf{AP}_I \cup \mathsf{AP}_O$. The original GR(1) specification class [28] uses logical formulas to describe the symbolic arena.

Solving GR(1) games using fixpoints

We now show how to solve GR(1) games by evaluating fixpoint formulas on GR(1) game arenas. Consider a GR(1) game $G_{\text{gr1}} = (\mathsf{AP}_I, \mathsf{AP}_O, V, v_0, \delta, \Phi)$ with

$\Phi = \bigwedge_{i=1}^{m} \mathsf{GF} a_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF} g_j$. The set of positions $W \subseteq V$ from which the system player wins the game is characterized by the fixpoint equation [18,8]:

$$W = \nu Z. \bigwedge_{j=1}^{n} \mu Y. \bigvee_{i=1}^{m} \nu X. \Box\Diamond \left[(g_j \wedge Z) \ \vee \ Y \ \vee \ (\neg a_i \wedge X) \right] \tag{2}$$

This fixpoint formula ensures that the system chooses to move into states of one of the three kinds: (1) states where it waits for an environment goal a_i to be reached, possibly forever $(\neg a_i \wedge X)$, (2) states that move the system closer to reaching its goal number j (Y), or (3) winning states that satisfy system goal number j $(g_j \wedge Z)$. The conjunction over all guarantees to the right of νZ ensures that all liveness guarantees are satisfied from all winning positions (unless some environment liveness assumption is violated). The disjunction over the environment goals permits the system to wait for the satisfaction of any of the environment liveness goals. At the end of evaluating the fixpoint formula, Z consists of the winning positions for the system. The system wins the GR(1) game if and only if W includes v_0.

Example. Consider a GR(1) game with $\mathsf{AP}_I = \{u\}$, $\mathsf{AP}_O = \{x, y\}$, and $\Phi = \mathsf{GF} u \rightarrow (\mathsf{GF} x \wedge \mathsf{GF} y)$. Equation 2 becomes:

$$W = \nu Z. \begin{bmatrix} \mu Y. \nu X. \Box\Diamond (xZ \ \vee \ Y \ \vee \ \bar{u}X) \wedge \\ \mu Y. \nu X. \Box\Diamond (yZ \ \vee \ Y \ \vee \ \bar{u}X) \end{bmatrix} \tag{3}$$

For conciseness, we write xZ instead of $x \wedge Z$, and $\bar{a}$ instead of $\neg a$.

Solving symbolic parity games using fixpoints

Consider a parity game $(\mathsf{AP}_I, \mathsf{AP}_O, V, v_0, \delta, c)$ with colors $\{0, \ldots, n\}$. The winning positions for the system player in such game are characterized by the fixpoint formula from [33,11] adapted to our setting:

$$W = \nu X^0 \mu X^1 \ldots \sigma X^n . \Box\Diamond (\vee_{i=1}^{n} \mathsf{color}_i \wedge X^i) \tag{4}$$

The operators ν and μ alternate, so the symbol σ is μ if n is odd and ν if n is even; $\mathsf{color}_i = \{v \mid c(v) = i\}$ denotes the set of vertices of color i.

Solving symbolic LTL games using fixpoints

Let G be a game with LTL objective Φ. We can construct a deterministic parity automaton $\mathcal{A}$ for Φ, build the product parity game $G \otimes \mathcal{A}$, and solve it with the help of Equation 4. An alternative approach is to embed the product into the fixpoint formula by using vector notation [10].

Consider an example. Let $G = (\mathsf{AP}_I, \mathsf{AP}_O, V, v_0, \delta, \Phi)$ be a game with $\Phi = \mathsf{GF} u \rightarrow (\mathsf{GF} x \wedge \mathsf{GF} y)$. The parity automaton for Φ is shown on Figure 1. It has two states, q_0 and q_1, and uses three colors. For three colors, the parity fixpoint

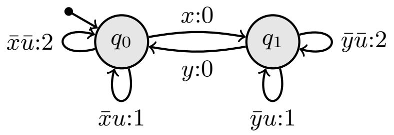

Fig. 1. Parity automaton for $\mathsf{GF}u \to (\mathsf{GF}x \wedge \mathsf{GF}y)$. Transitions are labeled by the proposition valuations for which they can be taken as well as the color of the transition.

formula in Equation 4 has structure $\nu Z.\mu Y.\nu X$. We index each set variable with the state of the automaton, thus Z is split into Z_0 and Z_1, etc. The formula is:

$$\begin{bmatrix} W_0 \\ W_1 \end{bmatrix} = \nu \begin{bmatrix} Z_0 \\ Z_1 \end{bmatrix}.\mu \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix}.\nu \begin{bmatrix} X_0 \\ X_1 \end{bmatrix}.\Box\Diamond \begin{bmatrix} xZ_1 & \vee & \bar{x}uY_0 & \vee & \bar{x}\bar{u}X_0 \\ yZ_0 & \vee & \bar{y}uY_1 & \vee & \bar{y}\bar{u}X_1 \end{bmatrix} \tag{5}$$

The top row encodes the transitions from state q_0 of the parity automaton: $q_0 \overset{x:0}{\to} q_1$ becomes xZ_1, $q_0 \overset{\bar{x}u:1}{\to} q_1$ becomes $\bar{x}uY_1$, $q_0 \overset{\bar{x}\bar{u}:2}{\to} q_0$ becomes $\bar{x}\bar{u}X_0$. After formula evaluation, the variable W_0 contains game positions winning for the system wrt. the parity automaton $\mathcal{A}_{q_0}$, while W_1 does so wrt. $\mathcal{A}_{q_1}$.

In general, suppose we are given a game whose winning objective is a deterministic parity automaton $(2^{\mathsf{AP}}, Q, q_0, \delta)$ with transition function $\delta : Q \times \Sigma \to Q \times \mathbb{N}$ that uses n colors $\{0, \ldots, n-1\}$. The set of winning game positions is characterized by the fixpoint formula:

$$\begin{bmatrix} W_1 \\ \vdots \\ W_{|Q|} \end{bmatrix} = \nu \begin{bmatrix} X_1^0 \\ \vdots \\ X_{|Q|}^0 \end{bmatrix}.\mu \begin{bmatrix} X_1^1 \\ \vdots \\ X_{|Q|}^1 \end{bmatrix} \ldots\ \sigma \begin{bmatrix} X_1^{n-1} \\ \vdots \\ X_{|Q|}^{n-1} \end{bmatrix}.\Box\Diamond \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_{|Q|} \end{bmatrix} \tag{6}$$

$$\text{where for all } j \in \{1, \ldots, |Q|\}, \text{we have } \psi_j = \bigvee_{\substack{x \in 2^{\mathsf{AP}} \\ \text{let } (q,c) = \delta(q_j, x)}} x \wedge X^c_{ind(q)}$$

where $ind : Q \to \{1, \ldots, |Q|\}$ is some state numbering (one-to-one) that maps the initial automaton state q_0 to 1. The game is won by the system if and only if the initial game position belongs to W_1.

3 Chains of Good-for-Games co-Büchi Automata

This section reviews the chain of good-for-games co-Büchi automata representation [19] for ω-regular languages used by our synthesis approach in Section 4.

Like parity automata, a chain of co-Büchi automaton representation of a language assigns colors to words. The central difference is that the chain representation relies on a sequence of automata, each taking care of a single color.

Definition 1. *Let $L \subseteq \Sigma^\omega$ be an omega-regular language. A falling chain of languages $L_1 \supset L_2 \supset \ldots \supset L_n$ is a* chain-of-co-Büchi representation *of L if*

– *every language L_i for $i \in \{1, \ldots, n\}$ is a co-Büchi language, and*

– *for every $w \in \Sigma^\omega$, the word w is in L if and only if $w \notin L_1$ or the highest index i such that $w \in L_i$ is even.*

Examples. The universal language Σ^ω has the singleton-chain $L_1 = \emptyset$, and the empty language has the chain $(L_1 = \Sigma^\omega) \supset (L_2 = \emptyset)$. The language of the LTL formula $\mathsf{GF}a$ over a single atomic proposition a is expressed by the chain $(L_1 = L(\mathsf{FG}\overline{a})) \supset (L_2 = \emptyset)$, and $L(\mathsf{FG}a)$ by $(L_1 = \Sigma^\omega) \supset (L_2 = L(\mathsf{FG}a)) \supset (L_3 = \emptyset)$.

The definition of the natural color of a word from [19] provides a canonical way to represent L as a chain of co-Büchi languages $L_1 \supset L_2 \supset \ldots \supset L_n$, which uses the minimal number of colors. Moreover, Abu Radi and Kupferman describe a procedure to construct a minimal and canonical good-for-games co-Büchi automaton for a given co-Büchi language [2]. Thus, every omega-regular language has a canonical minimal chain-of-co-Büchi-automata representation (***COCOA***).

The canonization procedure in [2, Thm.4.7] ensures the following property.

Lemma 1 ([2]). *Fix a canonical GFG co-Büchi automaton $\mathcal{A}$ computed by [2, Thm.4.7]. For every state q and letter x, either there is*

– *exactly one accepting transition, or there are*
– *one or more rejecting transitions. In this case:*
  • *all successors of q on x share the same suffix language L', i.e., for every two successors s_1 and s_2 of q on x: $L(\mathcal{A}, s_1) = L(\mathcal{A}, s_2)$, and*
  • *for every state q' with suffix language L', there is a rejecting transition to q' from q on x.*

Figure 2 on page 12 shows an example of a COCOA.

Strategies to get back on the track

Every GFG automaton has a strategy to resolve its nondeterminism such that a word is accepted if and only if the run adhering to this strategy is accepting. We allow such strategies to diverge for a finite number of steps, and show that this divergence does not affect the acceptance by canonical GFG automata.

Given a COCOA $\mathcal{A}^1, \ldots, \mathcal{A}^n$, define the *natural color* of a word to be the largest level l such that $\mathcal{A}^l$ accepts the word, or 0 if no such l exists. Thus, a word is accepted by the COCOA if and only if the natural color is even.

GFGness strategies f^l. Let $f^l : \Sigma^* \to Q^l$ be a GFG witness resolving nondeterminism in $\mathcal{A}^l$, for every $l \in \{1, \ldots, n\}$; we call f^l a golden strategy of $\mathcal{A}^l$, and the induced run for some given word is called its *golden run.*

Restrictions g^l. The synthesis approach, which will be described later, considers combined runs of all automata. Its efficiency depends on the number of reachable states in $Q^1 \times \ldots \times Q^n$, so it is beneficial to reduce this number. To this end, we introduce a restriction on successor choices. We first define a helpful notion: for a co-Büchi automaton $\mathcal{A}$ and its state q, let $L^{acc}(q)$ denote the set of words which have a run from q visiting only accepting transitions. For several automata

$\mathcal{A}^1, \ldots, \mathcal{A}^l$ and their states $q^1, \ldots, q^l$, define $L^{acc}(q^1, \ldots, q^l) = \bigwedge_i L^{acc}(q^i)$. Then, for $l \in \{1, \ldots, n\}$, define a *restriction* function $g^l : Q^l \times \Sigma \times Q^1 \times \ldots \times Q^{l-1} \to 2^{Q^l}$: for every q^l, x, $r^1, \ldots, r^{l-1}$, let $g^l(q^l, x, r^1, \ldots, r^{l-1}) = S \subseteq \delta^l(q^l, x)$ be a maximal set such that for every $r^l \in S$ there exists no other $\tilde{r}^l \in S$ with $L^{acc}(r^1, \ldots, r^{l-1}, \tilde{r}^l) \subsetneq L^{acc}(r^1, \ldots, r^l)$. Intuitively, given a current state q^l of the automaton $\mathcal{A}^l$, a letter x, and successor states $r^1, \ldots, r^{l-1}$ of the automata on lower levels, the function g^l returns a set of states among which $\mathcal{A}^l$ should pick a successor. Runs $\rho^1 = q_0^1 q_1^1 \ldots, \ldots, \rho^n = q_0^n q_1^n \ldots$ of $\mathcal{A}^1, \ldots, \mathcal{A}^n$ on a word $x_0 x_1 \ldots$ *satisfy* restrictions $g^1, \ldots, g^n$ if for every level $l \in \{1, \ldots, n\}$ and step $i \in \mathbb{N}$: $q_{i+1}^l \in g^l(q_i^l, x_i, q_{i+1}^1, \ldots, q_{i+1}^{l-1})$. Strategies $f^l : \Sigma^* \to Q^l$ for $l \in \{1, \ldots, n\}$ *satisfy* restrictions $g^1, \ldots, g^n$ if on every word the strategies yield runs satisfying the restrictions.

The following lemma states that requiring runs of $\mathcal{A}^1, \ldots, \mathcal{A}^n$ to satisfy the restrictions $g^1, \ldots, g^n$ preserves the natural colors and the GFGness.

Lemma 2. *There exist strategies $f^l : \Sigma^* \to Q^l$ for $l \in \{1, \ldots, n\}$ satisfying the restrictions $g^1, \ldots, g^n$ such that for every word of a natural color c, the strategies yield accepting runs $\rho^1, \ldots, \rho^c$ of $\mathcal{A}^1, \ldots, \mathcal{A}^c$.*

Proof. Fix a word w of a natural color c. Each automaton $\mathcal{A}^l$ of the chain has a GFG witness in the form of a strategy $h^l : \Sigma^* \to Q^l$ to resolve nondeterminism. From such strategies and the restrictions $g^1, \ldots, g^n$, we construct the sought strategies $f^1, \ldots, f^n$, inductively on the level, starting from the smallest level 1 and proceeding upwards to n.

Fix $l \in \{1, \ldots, n\}$, and suppose the strategies $f^1, \ldots, f^{l-1}$ are already defined; we define the strategy $f^l : \Sigma^* \to Q^l$. Fix a moment $i-1$. Let q_{i-1}^l be the state of the run ρ^l proceeding according to f^l, $\tilde{q}_i^l = h^l(x_0 \ldots x_{i-1})$ the successor state in the original run $\tilde{\rho}^l$ according to h^l, $q_i^1, \ldots, q_i^{l-1}$ the successor states in $\rho^1, \ldots, \rho^{l-1}$ adhering to $f^1, \ldots, f^{l-1}$, and $Q_i^l = g^l(q_{i-1}^l, x_{i-1}, q_i^1, \ldots, q_i^{l-1})$ the allowed successors on level l. Then:

- if $Q_i^l = \{q_i^l\}$ describes a unique choice, then $f^l(x_0 \ldots x_{i-1}) = q_i^l$ takes it,
- else f^l picks any $q_i^l \in Q_i^l$ s.t. $L^{acc}(q_i^1, \ldots, q_i^{l-1}, q_i^l) \supseteq L^{acc}(q_i^1, \ldots, q_i^{l-1}, \tilde{q}_i^l)$. Note that such q_i^l always exists because in canonical GFG co-Büchi automata a choice of a nondeterministic transition does not narrow the subsequent nondeterminism resolution.

We now show that the strategies $f^1, \ldots, f^l$ preserve the natural colors. Fix a word w. It suffices to prove that the original strategy h^l yields an accepting run $\tilde{\rho}^l$ if and only if f^l yields an accepting run ρ^l. If $\tilde{\rho}^l$ is rejecting, then ρ^l is also rejecting, for h^l is a witness of GFGness. Now assume that $\tilde{\rho}^l$ is accepting. After some moment m, the runs $\rho^1, \ldots, \rho^{l-1}, \tilde{\rho}^l$ never make a rejecting transition, hence $w_m w_{m+1} \ldots \in L^{acc}(q_m^1, \ldots, q_m^{l-1}, \tilde{q}_m^l)$. Let $m' \geq m$ be the first moment after m when ρ^l visits a rejecting transition; if no such m' exists, we are done. At moment m', the strategy f^l picks a successor $q_{m'+1}^l$ such that $L^{acc}(q_{m'+1}^1, \ldots, q_{m'+1}^l) \supseteq L^{acc}(q_{m'+1}^1, \ldots, \tilde{q}_{m'+1}^l)$. Since $w_{m'+1} \ldots \in L^{acc}(q_{m'+1}^1, \ldots, q_{m'+1}^{l-1}, \tilde{q}_{m'+1}^l)$, that suffix also belongs to a larger L^{acc} wrt. $q_{m'+1}^l$. Hence the run ρ^l is accepting. $\square$

Get-back strategies $f^l_\star$. We now consider runs that diverge from golden runs. Given an individual strategy $f^l : \Sigma^* \to Q^l$, define $f^l_\star : \Sigma^* \times Q^l \times \Sigma \rightharpoonup Q^l$ to be a strategy-like function which, when presented with a choice, makes the same choice as f^l. Formally: for every $p \in \Sigma^*$, $q \in Q^l$ reachable from the initial state on reading p, and $x \in \Sigma$, the value $f^l_\star(p, q, x) = f^l(p \cdot x)$ if $\mathcal{A}^l$ needs to take a rejecting transition from q on x, otherwise there is no choice to be made and $f^l_\star(p, q, x) = q'$ for the unique successor q' of q on reading x. It follows from properties of canonical GFG automata (Lemma 1) that every successor chosen by $f^l_\star$ satisfies the transition relation of $\mathcal{A}^l$. We now prove that it is sufficent to adhere to $f^l_\star$ only eventually.

Lemma 3. *Fix a COCOA and a word w. For $l \in \{1, \dots, n\}$, suppose $\mathcal{A}^l$ on w has a rejecting run ρ^l that eventually adheres to $f^l_\star$, where $f^l_\star$ is constructed from f^l of Lemma 2. Then $\mathcal{A}^l$ rejects w.*

The proof is based on Lemma 1, which implies that two diverging runs of a canonical GFG automaton on the same word can always be converged once a rejecting transition is taken.

Proof. For $l = 0$ the claim trivially holds; assume $l > 0$. Let $\rho^l_\star$ be the golden run of $\mathcal{A}^l$ on the word. Let m be the moment starting from which ρ^l adheres to the golden strategy of $\mathcal{A}^l$. Let n be the first moment $n \geq m$ when $\mathcal{A}^l$ makes a rejecting transition: by properties of canonical GFG automata (Lemma 1), there must be a rejecting transition to the same state as in $\rho^l_\star$. The strategy $f^l_\star$ moves the automaton $\mathcal{A}^l$ in ρ^l into the same state at moment $n+1$ as it is in $\rho^l_\star$. Afterwards, the strategy $f^l_\star$ ensures that $\mathcal{A}^l$ in ρ^l follows exactly the same transitions as $\mathcal{A}^l$ in $\rho^l_\star$. Hence, the golden run $\rho^l_\star$ is rejecting: $\mathcal{A}^l$ rejects w. □

COCOA product

In this section, we compose individual automata of COCOA into a product which is a good-for-games alternating parity automaton [9]. The results above imply that the languages of a COCOA and its product coincide. Later we use COCOA products to solve games with LTL objectives.

Alternating automata. A *simple*[1] *alternating parity automaton* (Σ, Q, q_0, δ) has a transition function of type $\delta : Q \times \Sigma \to 2^Q \times \mathbb{N} \times \{rej, acc\}$. For instance, $\delta(q, x) = (\{q_1, q_2\}, 1, rej)$ means that from state q on reading letter x there are transitions to q_1 and q_2, both labelled with color 1, and the choice between q_1 and q_2 is controlled by the rejector player. There are two players, rejector and acceptor, and the acceptance of a word $w = x_0 x_1 \dots$ is defined via the following *word-checking game.* Starting in q_0, the two players resolve nondeterminism and build a *play* $(q_0, c_0, pl_0, q_1)(q_1, c_1, pl_1, q_2)\dots$: suppose the play sequence is in state

[1] 'Simple' refers to a simpler form of the transition function. We use $\delta : Q \times \Sigma \to 2^Q \times \mathbb{N} \times \{rej, acc\}$ while the general form is $\delta : Q \times \Sigma \to \mathsf{B}^+(Q)$ plus parity assignment $Q \times \Sigma \times Q \to \mathbb{N}$. We forbid mixing conjunctions and disjunctions.

q_i, let $\delta(q_i, x_i) = (Q_{i+1}, c_i, pl_i)$: if $pl_i = rej$ then the rejector chooses a state $q_{i+1} \in Q_{i+1}$, otherwise the acceptor chooses. The play sequence is then extended by $(q_i, c_i, pl_i, q_{i+1})$ and the procedure repeats from state q_{i+1}. The play is *won* by the acceptor if the minimal color appearing infinitely often in $c_0c_1\ldots$ is even (min-even acceptance), otherwise it is won by the rejector. The word-checking game is *won* by the acceptor if it has a strategy $f_w : Q^* \to Q$ to resolve its nondeterminism to win every play; otherwise the game is won by the rejector, who then also has a winning strategy. Note that although the acceptor strategy does not know the rejector choices beforehand, it knows the word w. The word is *accepted* by the automaton if the word-checking game is won by the acceptor.

A simple alternating automaton is *good-for-games*, abbreviated *A-GFG*, if the acceptor player has a strategy $f_{\text{acc}} : (Q \times \Sigma)^* \to Q$ to win the word-checking game for every accepting word, and the rejector player has a strategy $f_{\text{rej}} : (Q \times \Sigma)^* \to Q$ winning for every rejected word. These strategies depend only on the currently seen word prefix, not the whole word. We remark that our definition of GFGness differs from [9] but they show the equivalence [9, Thm.8].

COCOA product. The product is built in three steps. First, we define a naive product, which combines individual chain automata into A-GFG in a straightforward way. The naive product may contain states whose removal does not affect its language, hence in the second step we define a product with reduced sets of states and transitions. In turn, the reduced product may miss transitions beneficial for synthesis. Therefore, in the last step, we enrich the reduced product with transitions to derive the optimized, and final, COCOA product.

Given a COCOA $\mathcal{A}^l = (\Sigma, Q^l, q_0^l, \delta^l)$ with $l \in \{1, \ldots, n\}$, the *naive COCOA product* is the following simple alternating parity automaton (Σ, Q, q^0, δ). Each state is a tuple from $Q^1 \times \ldots \times Q^n$, $q^0 = (q_0^1, \ldots, q_0^n)$, and the set of states consists of those reachable from the initial state under the transition relation defined next. The transition relation $\delta : Q \times \Sigma \to 2^Q \times \mathbb{N} \times \{rej, acc\}$ simulates individual automata of the COCOA. Consider an arbitrary $(q^1, \ldots, q^n) \in Q$, $x \in \Sigma$; let r be the smallest number such that $\mathcal{A}^r$ has a rejecting transition from q^r on reading x, i.e., $(q^r, x, \tilde{q}^r, 1) \in \delta^r$ for some $\tilde{q}^r \in Q^r$, otherwise set r to $n+1$. By abuse of notation, define $\delta^l(q^l, x) = \{\tilde{q}^l \mid \exists p : (q^l, x, \tilde{q}^l, p) \in \delta^l\}$ to be the set of successor states of q^l on reading x in $\mathcal{A}^l$. Let pl^r be rej for odd r and acc for even r. Then, $\delta((q^1, \ldots, q^n), x) = (\tilde{Q}, r-1, pl^r)$, where:

$$\tilde{Q} = \{(\tilde{q}^1, \ldots, \tilde{q}^n) \mid \tilde{q}^l \in \delta^l(q^l, x) \textit{ for every } l\}.$$

Notice that the automata on levels $l < r$ have unique successors ($\tilde{q}^l$ is unique) as their transitions are accepting and hence deterministic (by Lemma 1 on page 8). The automata on levels $l \geq r$ may need to resolve nondeterminism, which is done by a single player pl^r in the product.

The *reduced COCOA product* is defined by replacing the definition of $\tilde{Q}$ by

$$\tilde{Q} = \{(\tilde{q}^1, \ldots, \tilde{q}^n) \mid \tilde{q}^l \in g^l(\tilde{q}^1, \ldots, \tilde{q}^{l-1}, x, q^l) \textit{ for every } l\}$$

where the restriction function g^l was defined on page 9. As a result, this set $\tilde{Q}$ has no two states $(q^1, \ldots, q^n)$ and $(\tilde{q}^1, \ldots, \tilde{q}^n)$ with $L^{acc}(q^1, \ldots, q^n) \subseteq L^{acc}(\tilde{q}^1, \ldots,$

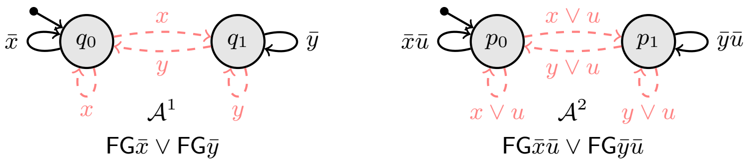

Fig. 2. COCOA for the language $\mathsf{GF}u \to (\mathsf{GF}x \wedge \mathsf{GF}y)$. Rejecting transitions are dashed.

$\tilde{q}^n$). The set of states of the reduced COCOA product is the set of states from $Q^1 \times \ldots \times Q^n$ reachable under the above definition.

Finally, given a reduced COCOA product $(\Sigma, Q, q^0, \delta_R)$, we now define the *optimized COCOA product* $(\Sigma, Q, q^0, \delta_O)$. It has the same states Q as the reduced product but adds transitions. For $(q^1, \ldots, q^n) \in Q$, $x \in \Sigma$, let $(\tilde{Q}_R, r-1, pl^r) = \delta_R((q^1, \ldots, q^n), x)$. Then $\delta_O((q^1, \ldots, q^n), x) = (\tilde{Q}, r-1, pl^r)$, where

$$\begin{aligned}\tilde{Q} = \tilde{Q}_R \cup \big\{ (\tilde{q}^1, \ldots, \tilde{q}^n) \in Q :& \\ &\forall l \in \{1, \ldots, r-1\}\colon q^l \in \delta^l(q^l, x) \wedge \\ &\forall l \in \{r, \ldots, n\}. \exists (\tilde{q}^1_R, \ldots, \tilde{q}^n_R) \in \tilde{Q}_R\colon L(\tilde{q}^l) = L(\tilde{q}^l_R) \big\}.\end{aligned}$$

In the first condition, the successor q^l for $l \le r-1$ is uniquely defined. The second condition on levels higher than $r-1$ allows for state jumping.

Lemma 4. *For every COCOA, the optimized product is A-GFG and has the same language as the COCOA.*

Proof. We describe two strategies, $f_{\text{acc}} : (Q \times \Sigma)^* \to Q$ for the acceptor and $f_{\text{rej}} : (Q \times \Sigma)^* \to Q$ for the rejector, and prove two claims: for every word, (1) if the word is accepted by COCOA, the acceptor wins the word-checking game using f_{acc}, (2) if the word is rejected by COCOA, the rejector wins the word-checking game using f_{rej}. The lemma follows from these claims.

We define f_{acc}. Given a finite history $h = ((q^1_1, ..., q^n_1), x_1)...((q^1_i, ..., q^n_i), x_i)$, let $f_{\text{acc}}(h) = (q^1_{i+1}, ..., q^n_{i+1})$, where for $l = 1, ..., n$:

– if l is even: $q^l_{i+1} = f^l_\star(x_1 \ldots x_{i-1}, q^l_i, x_i)$;
– if l is odd, pick arbitrary $q^l_{i+1} \in g^l(q^1_{i+1}, \ldots, q^{l-1}_{i+1}, q^l_i)$.

The strategy f_{rej} is built similarly but $f^l_\star$ is used for odd l. Finally, the two items are then proven using contraposition and then applying Lemma 3. □

Example. Figure 3 shows the optimized product for COCOA in Figure 2.

4 Solving LTL Games Using Chain of co-Büchi Automata

This section shows how to solve symbolic games with LTL objectives by going through COCOA. For a given LTL specification we construct a deterministic

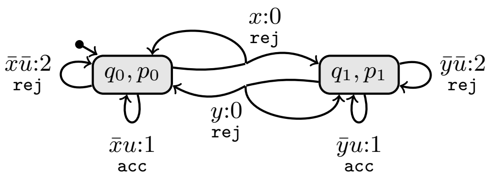

Fig. 3. Optimized COCOA product for $\mathsf{GF}u \to (\mathsf{GF}x \wedge \mathsf{GF}y)$. It has only two nondeterministic transitions, connecting (q_0, p_0) and (q_1, p_1), controlled by the rejector. For instance, $\delta((q_0, p_0), x) = (\{(q_0, p_0), (q_1, p_1)\}, 0, rej)$.

parity automaton and then a COCOA using the effective procedure of [19]. We then compute the COCOA product. Finally, we encode the symbolic game with a COCOA product objective into a fixpoint formula. The latter step is simple because the COCOA product is a good-for-games alternating automaton, and such automata are composable with games [9, Thm.8]. Finally, we show that the GR(1) fixpoint equation is a special case of the COCOA fixpoint formula.

Fixpoint formula for games with COCOA objectives

Given a game with an objective in the form of an optimized COCOA product $(2^{\mathsf{AP}}, Q, q_0, \delta)$, we construct a fixpoint formula that characterizes the set of winning positions. Since the COCOA product is a good-for-games parity automaton, the formula resembles Equation 6. It has the structure $\nu X^0.\mu X^1.\ldots\sigma X^n$ where $n+1$ is the number of colors in the COCOA product, and the operators ν and μ alternate. As before, we use the vector notation, and split each variable X^l into $|Q|$ variables $X^l_1, \ldots, X^l_{|Q|}$, one per state of the COCOA product, and the kth row in the fixpoint formula encodes transitions from state q_k of the product. Let $ind : Q \to \{1, \ldots, |Q|\}$ be some one-to-one state numbering with the initial state of the COCOA product mapped to 1, and let OP^{pl} denote $\bigvee$ when pl is acc otherwise it is $\bigwedge$. The following fixpoint formula computes, for each state q of the COCOA product, the set $W_{ind(q)}$ of game positions from which the system player wins the game wrt. the COCOA product whose initial state is set to q:

$$\begin{bmatrix} W_1 \\ \vdots \\ W_{|Q|} \end{bmatrix} = \nu \begin{bmatrix} X^0_1 \\ \vdots \\ X^0_{|Q|} \end{bmatrix} .\mu \begin{bmatrix} X^1_1 \\ \vdots \\ X^1_{|Q|} \end{bmatrix} \ldots\ \sigma \begin{bmatrix} X^n_1 \\ \vdots \\ X^n_{|Q|} \end{bmatrix} .\Box\Diamond \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_{|Q|} \end{bmatrix}, \quad \text{where for all } j\text{:} \tag{7}$$

$$\psi_j = \bigvee_{\substack{x \in 2^{\mathsf{AP}} \\ \text{let } (\tilde{Q}, c, pl) = \delta(q_j, x)}} \left(x \wedge \mathsf{OP}^{pl}_{q \in \tilde{Q}} X^c_{ind(q)} \right)$$

The game wrt. the COCOA product is won by the system player if and only if $v_0 \in W_1$. Since the languages of COCOA and its optimized product coincide (Lemma 4), we arrive at the following theorem.

Theorem 1. *A game with an LTL objective Φ is won by the system if and only if the initial game position belongs to W_1 computed by Equation 7 for the optimized COCOA product for Φ.*

Example. Consider the LTL specification $\mathsf{GF}u \rightarrow (\mathsf{GF}x \wedge \mathsf{GF}y)$. The optimized product contains only states (q_0, p_0) and (q_1, p_1). The fixpoint formula is:

$$\nu \begin{bmatrix} Z_{00} \\ Z_{11} \end{bmatrix} . \mu \begin{bmatrix} Y_{00} \\ Y_{11} \end{bmatrix} . \nu \begin{bmatrix} X_{00} \\ X_{11} \end{bmatrix} . \Box\Diamond \begin{bmatrix} xZ_{00}Z_{11} \ \vee \ \bar{x}uY_{00} \ \vee \ \bar{x}\bar{u}X_{00} \\ yZ_{00}Z_{11} \ \vee \ \bar{y}uY_{11} \ \vee \ \bar{y}\bar{u}X_{11} \end{bmatrix}$$

where the subscript index ij denotes a state (q_i, p_j) of the optimized COCOA product. The LTL game is won by the system if and only if at the end of evaluation the initial game position v_0 belongs to Z_{00}. This formula has a structure similar to the GR(1) Equation 3, in particular it uses the conjunction over Z variables which leads to a reduction of the number of fixpoint iterations. In contrast, the parity formula in Equation 5 misses this acceleration.

GR(1) synthesis as a special case

We argue that for GR(1) specifications, the COCOA fixpoint Equation 7 becomes similar – in spirit – to GR(1) fixpoint Equation 2. Consider a GR(1) formula $\bigwedge_{i=1}^{m} \mathsf{GF}a_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}g_j$. Its COCOA has two automata, $\mathcal{A}^1$ and $\mathcal{A}^2$. The automaton $\mathcal{A}^1$ accepts exactly the words that violate one of the guarantees, while $\mathcal{A}^2$ accepts exactly the words that violate one of the guarantees and one of the assumptions. In order to reason able number of states in canonical automata, we assume henceforth that in the GR(1) formula, no assumption implies another assumption or guarantee, and no guarantee implies another guarantee. The structures of $\mathcal{A}^1$ and A^2 are as follows. The automaton $\mathcal{A}^1$ has one state per guarantee (n in total), while $\mathcal{A}^2$ has one per combination of liveness assumption and guarantee ($m \cdot n$ in total). The optimized COCOA product has exactly one state for each assumption-guarantee combination, $m \cdot n$ in total, versus $n \cdot (m \cdot n)$ for the non-optimized product. Let $\{1, \ldots, m\} \times \{1, \ldots, n\}$ be the states of the optimized product, and let $(1,1)$ be initial. For each state (i,j):

- for every $x \models \bar{a}_i\bar{g}_j$: $\delta((i,j), x) = \big(\{(i,j)\}, 2, \mathit{rej}\big)$,
- for every $x \models a_i\bar{g}_j$: $\delta((i,j), x) = \big(\{(i',j) \mid i' \in \{1, \ldots, m\}\}, 1, \mathit{acc}\big)$, and
- for every $x \models g_j$: $\delta((i,j), x) = \big(\{1, \ldots, m\} \times \{1, \ldots, n\}, 0, \mathit{rej}\big)$.

The fixpoint formula for such COCOA product has the form:

$$\begin{bmatrix} W_{1,1} \\ \vdots \\ W_{m,n} \end{bmatrix} = \nu \begin{bmatrix} Z_{1,1} \\ \vdots \\ Z_{m,n} \end{bmatrix} . \mu \begin{bmatrix} Y_{1,1} \\ \vdots \\ Y_{m,n} \end{bmatrix} . \nu \begin{bmatrix} X_{1,1} \\ \vdots \\ X_{m,n} \end{bmatrix} . \Box\Diamond \begin{bmatrix} \psi_{1,1} \\ \vdots \\ \psi_{m,n} \end{bmatrix}, \text{ where for all } i, j\text{:}$$

$$\psi_{i,j} = g_j\big(\bigwedge_{\substack{i' \in \{1,\ldots,m\} \\ j' \in \{1,\ldots,n\}}} Z_{i',j'}\big) \ \vee \ a_i\bar{g}_j\big(\bigvee_{i' \in \{1,\ldots,m\}} Y_{i',j}\big) \ \vee \ \bar{a}_i\bar{g}_j X_{i,j}$$

The conjunction $\bigwedge_{i',j'} Z_{i',j'}$ and disjunctions $\bigvee_{i'} Y_{i',j}$ enable faster information propagation which results in smaller number of fixpoint iterations. Such information sharing is present in GR(1) fixpoint Equation 2, and it is in this sense the COCOA approach generalizes GR(1) approach. In contrast, the parity fixpoint formula for GR(1) specifications misses this acceleration.

We now optimize the equation to reduce the number of variables. First, we introduce variables Y_j and Z_j, for $j \in \{1, ..., n\}$, and transform the formula into

$$\begin{bmatrix} W_1 \\ \vdots \\ W_n \end{bmatrix} = \nu \begin{bmatrix} Z_1 \\ \vdots \\ Z_n \end{bmatrix} . \mu \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} . \begin{bmatrix} \bigvee_i \Phi_{i,1} \\ \vdots \\ \bigvee_i \Phi_{i,n} \end{bmatrix}, \textit{ where}$$

$$\begin{bmatrix} \Phi_{1,1} \\ \vdots \\ \Phi_{m,n} \end{bmatrix} = \nu \begin{bmatrix} X_{1,1} \\ \vdots \\ X_{m,n} \end{bmatrix} . \Box\Diamond \begin{bmatrix} \psi_{1,1} \\ \vdots \\ \psi_{m,n} \end{bmatrix}, \textit{ where}$$

$$\psi_{i,j} = g_j\Big(\bigwedge_{j' \in \{1,\ldots,n\}} Z_{j'}\Big) \;\vee\; a_i \bar{g}_j Y_j \;\vee\; \bar{a}_i \bar{g}_j X_{i,j}$$

Note that for every $i \in \{1, \ldots, m\}$, the value $W_{i,j}$ computed by the old formula equals the value W_j computed by the new formula ($W_{i,j} = W_i$), where $j \in \{1, \ldots, n\}$. We then introduce a fresh variable Z, and transform the formula to:

$$W = \nu Z. \bigwedge_{j \in \{1,\ldots,n\}} \Psi_j, \textit{ where}$$

$$\begin{bmatrix} \Psi_1 \\ \vdots \\ \Psi_n \end{bmatrix} = \mu \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} . \begin{bmatrix} \bigvee_i \Phi_{i,1} \\ \vdots \\ \bigvee_i \Phi_{i,n} \end{bmatrix}, \textit{ where}$$

$$\begin{bmatrix} \Phi_{1,1} \\ \vdots \\ \Phi_{m,n} \end{bmatrix} = \nu \begin{bmatrix} X_{1,1} \\ \vdots \\ X_{m,n} \end{bmatrix} . \Box\Diamond \begin{bmatrix} g_1 Z \;\vee\; a_1 \bar{g}_1 Y_1 \;\vee\; \bar{a}_1 \bar{g}_1 X_{1,1} \\ \vdots \\ g_n Z \;\vee\; a_m \bar{g}_n Y_n \;\vee\; \bar{a}_m \bar{g}_n X_{m,n} \end{bmatrix}$$

After this transformation, we have $W = W_j$ for every $j \in \{1, \ldots, n\}$. Finally, the last equations can be folded into the formula

$$W = \nu Z. \bigwedge_{j=1}^{n} \mu Y. \bigvee_{i=1}^{m} \nu X. \Box\Diamond \big[g_j Z \;\vee\; a_i \bar{g}_j Y \;\vee\; \bar{a}_i \bar{g}_j X \big]$$

which is equal to Equation 2 modulo expressions in front of the variables. Our prototype tool implements a generalized version of such formula optimization.

5 Evaluation

Evaluation goals are: (G1) show that standard LTL synthesizers do not fit our synthesis problem, (G2) compare our approach against specialized GR(1) synthesizer, and (G3) compare the COCOA approach against the parity approach.

We implemented COCOA and parity approaches in a prototype tool `reboot`. It uses SPOT [16] to convert LTL specifications (the liveness part of GR(1)) to deterministic parity automata. From it, `reboot` builds COCOA using the construction described in [19]. The COCOA is then compiled into a fixpoint formula in Equation 7, and symbolically evaluated on the game graph. For symbolic encoding of game positions and transitions, we use the BDD library CUDD [32].

We compare our approaches with GR(1) synthesis tool `slugs` [18] and the LTL synthesis tool `strix` [26] which represent the state of the art. The experiments were performed on a Linux machine with AMD EPYC 7502 processor; the timeout was set to 1 hour. To implement the comparison, we collected existing and created new benchmarks: AMBA, lift, and robot on a grid. Each specification is written in an extension of the `slugs` format: it encodes a symbolic game graph using logical formulas over system and environment propositions, and an LTL property on top of it. In total, there are 80 benchmarks, all realizable.

The evaluation data is available at https://doi.org/10.5281/zenodo.10448487

AMBA and lift. We use two parameterized benchmarks inspired by [8], each having two versions, a GR(1) and an LTL version. The first specification encodes an elevator behaviour and is parameterized by the number of floors. Its GR(1) specification has one liveness assumption and a parameterized number of guarantees ($\mathsf{GF} \to \bigwedge_i \mathsf{GF}$). Lift's LTL version adds an additonal request-response assumption and has the form $\mathsf{GF} \wedge (\mathsf{GF} \to \mathsf{GF}) \to \bigwedge_i \mathsf{GF}$, which requires 5 parity colors. There are 24 GR(1) instances and 21 LTL instances, with the number of Boolean propositions ranging from 7 to 34. The AMBA specification describes the behaviour of an industrial on-chip bus arbiter serving a parameterized number of clients. Its GR(1) version has the shape $\mathsf{GF} \to \bigwedge_i \mathsf{GF}$; our new LTL modification replaces one safety guarantee φ by $\mathsf{FG}\varphi$, which allows the system to violate it during some initial phase, and we add an assumption of the form $\mathsf{GF} \to \mathsf{GF}$. Overall, the AMBA's LTL specification has the form $\mathsf{GF} \wedge (\mathsf{GF} \to \mathsf{GF}) \to \mathsf{FG} \wedge \bigwedge_i \mathsf{GF}$, and requires 7 parity colors. There are 14 GR(1) instances and 7 LTL instances; the number of Boolean propositions is 22 for the specification serving two clients, and 77 for the 15-client version.

Robot on a grid. This benchmark describes the standard scenario from robotics domain: a robot moves on a grid, there are walls, doors, pickup and delivery locations, and a moving obstacle. When requested, the robot has to pickup a package and deliver it to the target location, while avoiding collisions with the walls and the obstacle and passing through the doors only when they are open. The GR(1) specification has parameterized number of assumptions and guarantees: $\bigwedge_i \mathsf{GF} \to \bigwedge_i \mathsf{GF}$. The LTL version introduces preferential paths: the robot has to eventually always use it assuming that the moving obstacle only moves along her preferred path. This yields the shape $\mathsf{FG} \wedge \bigwedge_i \mathsf{GF} \to \mathsf{FG} \wedge \bigwedge_i \mathsf{GF}$ (5 colors). There are 16 maps of size 8×16 with varying number of delivery-pickup locations and doors. The number of Boolean propositions ranges from 24 to 53.

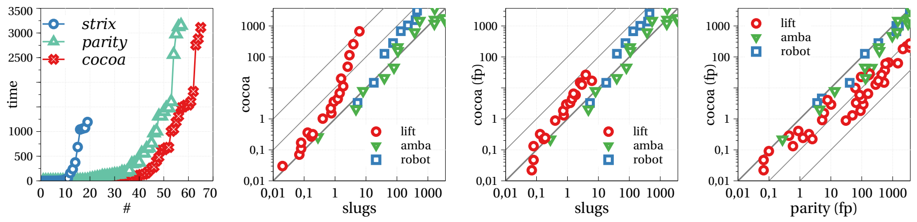

Fig. 4. From left to right: (G1) Cactus plot comparing our approaches with LTL synthesizer `strix` [26]; (G2a) Comparing COCOA-based approach with GR(1) synthesizer `slugs` [17]; (G2b) The same but excluding LTL-to-parity translation time; (G3) Comparing COCOA and parity approaches (excluding LTL-to-parity translation time).

G1: Comparing with LTL synthesizer. Figure 4 shows a cactus plot. On these problems, the LTL synthesizer `strix` is slower than specialized solvers. The reason is the sheer number of states in benchmark game arenas: e.g., benchmark *amba15* uses 77 Boolean propositions, yielding the naive estimate of game arena size in 2^{77} states. Solver `strix` tries to construct an explicit-state automaton describing this game arena and the LTL property, which is a bottleneck. In contrast, symbolic solvers like `slugs` or `reboot` represent game arenas symbolically using BDDs, and `reboot` constructs explicit automata only for LTL properties.

G2: Comparing with GR(1) synthesizer. The second diagram in Figure 4 compares the COCOA approach with `slugs` on the GR(1) benchmarks. The diagram shows the total solving time, including the time `reboot` spends calling SPOT for translating GR(1) liveness formula to parity automaton. On Lift examples, most of the time is spent in this translation when the number of floors exceeds 15: for instance, on benchmark *lift20* `reboot` spent 650 out of total 670 seconds in translation. If we count only the time spent in fixpoint evaluation – and that is a more appropriate measure since GR(1) liveness formulas have a fixed structure – the performances are comparable, see the third diagram.

G3: COCOA vs. parity. The last diagram in Figure 4 compares COCOA and parity approaches on all the benchmarks, and shows that the COCOA approach is significantly faster than the parity one. We note that on these examples, the number of states in the optimized COCOA product was equal to or less than the number of states in the parity automaton. At the same time, the number of fixpoint iterations performed by the COCOA approach was always significantly smaller than for the parity one. Intuitively, this is due to the structure of COCOA fixpoint equation that propagates information faster than the parity one.

Remarks. We did not compare with other symbolic approaches for solving parity or Rabin games [15,14,6]: although they use symbolic algorithms, as input these tools require games in explicit form or their game encoding separates positions into those of player-1 and player-2; both significantly affects the performance.

While all our benchmarks were realizable, the prototype tool was systematically compared against other approaches on both realizable and unrealizable random specifications using fuzz testing.

References

1. Reactive synthesis competition SyntComp 2023: Results. http://www.syntcomp.org/syntcomp-2023-results, accessed: 15-09-2023
2. Abu Radi, B., Kupferman, O.: Minimization and canonization of GFG transition-based automata. Log. Methods Comput. Sci. **18**(3) (2022)
3. Alur, R., Torre, S.L.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Log. **5**(1), 1–25 (2004)
4. Amram, G., Maoz, S., Pistiner, O.: GR(1)*: GR(1) specifications extended with existential guarantees. In: Third World Congress on Formal Methods (FM). pp. 83–100 (2019)
5. Arnold, A., Niwiński, D.: Rudiments of mu-calculus. Elsevier (2001)
6. Banerjee, T., Majumdar, R., Mallik, K., Schmuck, A.K., Soudjani, S.: Fast symbolic algorithms for omega-regular games under strong transition fairness. TheoretiCS **2** (2023)
7. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph Games and Reactive Synthesis, pp. 921–962. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_27, https://doi.org/10.1007/978-3-319-10575-8_27
8. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012)
9. Boker, U., Lehtinen, K.: Good for Games Automata: From Nondeterminism to Alternation. In: Fokkink, W., van Glabbeek, R. (eds.) 30th International Conference on Concurrency Theory (CONCUR 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 140, pp. 19:1–19:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.19, http://drops.dagstuhl.de/opus/volltexte/2019/10921
10. Bradfield, J.C., Walukiewicz, I.: The mu-calculus and model checking. In: Handbook of Model Checking, pp. 871–919 (2018)
11. Bruse, F., Falk, M., Lange, M.: The fixpoint-iteration algorithm for parity games. In: Fifth International Symposium on Games, Automata, Logics and Formal Verification (GandALF). pp. 116–130 (2014)
12. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986)
13. Church, A.: Logic, arithmetic, and automata. In: International Congress of Mathematicians (Stockholm, 1962), pp. 23–35. Institute Mittag-Leffler, Djursholm (1963)
14. Di Stasio, A., Murano, A., Vardi, M.Y.: Solving parity games: Explicit vs symbolic. In: Implementation and Application of Automata: 23rd International Conference, CIAA 2018, Charlottetown, PE, Canada, July 30–August 2, 2018, Proceedings 23. pp. 159–172. Springer (2018)
15. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 291–308. Springer International Publishing, Cham (2018)
16. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and \omega -automata manipulation. In: 14th International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 122–129 (2016)
17. Ehlers, R.: Generalized Rabin(1) synthesis with applications to robust system synthesis. In: Third International NASA Formal Methods Symposium (NFM). pp. 101–115 (2011)

18. Ehlers, R., Raman, V.: Slugs: Extensible GR(1) synthesis. In: 28th International Conference on Computer Aided Verification. pp. 333–339 (2016)
19. Ehlers, R., Schewe, S.: Natural colors of infinite words. In: 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS) (2022), presentation available at https://www.youtube.com/watch?v=RSd25TiELUo
20. Filiot, E., Jin, N., Raskin, J.: An antichain algorithm for LTL realizability. In: 21st International Conference on Computer Aided Verification (CAV). pp. 263–277 (2009)
21. Filiot, E., Jin, N., Raskin, J.F.: Antichains and compositional algorithms for ltl synthesis. Formal Methods in System Design **39**, 261–296 (2011)
22. Finkbeiner, B., Schewe, S.: Bounded synthesis. Int. J. Softw. Tools Technol. Transf. **15**(5-6), 519–539 (2013)
23. Godhal, Y., Chatterjee, K., Henzinger, T.A.: Synthesis of AMBA AHB from formal specification: a case study. Int. J. Softw. Tools Technol. Transf. **15**(5-6), 585–601 (2013)
24. Gritzner, D., Greenyer, J.: Synthesizing executable PLC code for robots from scenario-based GR(1) specifications. In: Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops. pp. 247–262 (2017)
25. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: International Conference on Computer Aided Verification. pp. 31–44. Springer (2006)
26. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. pp. 578–586. Springer (2018)
27. Piterman, N., Pnueli, A.: Temporal Logic and Fair Discrete Systems, pp. 27–73. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_2, https://doi.org/10.1007/978-3-319-10575-8_2
28. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. In: Verification, Model Checking, and Abstract Interpretation: 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006. Proceedings 7. pp. 364–380. Springer (2006)
29. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (FOCS). pp. 46–57 (1977)
30. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: 16th International Colloquium on Automata, Languages and Programming (ICALP). pp. 652–671 (1989)
31. Sohail, S., Somenzi, F.: Safety first: a two-stage algorithm for the synthesis of reactive systems. International Journal on Software Tools for Technology Transfer **15**, 433–454 (2013)
32. Somenzi, F.: CUDD: CU Decision Diagram package release 3.0.0 (2016)
33. Walukiewicz, I.: Monadic second-order logic on tree-like structures. Theoretical computer science **275**(1-2), 311–346 (2002)
34. Wong, K.W., Kress-Gazit, H.: From high-level task specification to robot operating system (ROS) implementation. In: First IEEE International Conference on Robotic Computing, IRC 2017, Taichung, Taiwan, April 10-12, 2017. pp. 188–195 (2017)
35. Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., Murray, R.M.: Tulip: a software toolbox for receding horizon temporal logic planning. In: 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 313–314 (2011)

36. Zudaire, S.A., Nahabedian, L., Uchitel, S.: Assured mission adaptation of UAVs. ACM Trans. Auton. Adapt. Syst. **16**(3–4) (jul 2022)