Formal Methods in Computer-Aided Design 2020

Reactive Synthesis from Extended Bounded Response LTL Specifications

Alessandro Cimatti*, Luca Geatti*†, Nicola Gigante†, Angelo Montanari† and Stefano Tonetta*
* Fondazione Bruno Kessler, Trento, Italy,
Email: [cimatti,lgeatti,tonettas]@fbk.eu
† University of Udine, Udine, Italy,
Email: [name.surname]@uniud.it

***Abstract*—Reactive synthesis is a key technique for the design of correct-by-construction systems and has been thoroughly investigated in the last decades. It consists in the synthesis of a controller that reacts to environment's inputs satisfying a given temporal logic specification. Common approaches are based on the explicit construction of automata and on their determinization, which limit their scalability.**

In this paper, we introduce a new fragment of Linear Temporal Logic, called Extended Bounded Response LTL ($\mathsf{LTL}_{\mathsf{EBR}}$), that allows one to combine bounded and universal unbounded temporal operators (thus covering a large set of practical cases), and we show that reactive synthesis from $\mathsf{LTL}_{\mathsf{EBR}}$ specifications can be reduced to solving a safety game over a deterministic symbolic automaton built directly from the specification. We prove the correctness of the proposed approach and we successfully evaluate it on various benchmarks.

I. Introduction

Since the dawn of computer science, synthesizing correct-by-construction systems starting from a specification is an important and difficult task. A practical algorithm to solve this task would be a big improvement in declarative programming, since it would allow the programmer to write only the specification of the program, freeing her from possible design or implementation errors, that, in many cases, are due to an imperative style of programming. In the context of formal verification and model-based design, the possibility of synthesizing a controller able to comply with the specification for all possible behaviors of the environment would be of great importance as well: all the effort would be directed to improve the quality of the specification for the controller.

Reactive synthesis was first proposed by Church [7] and solved by Büchi and Landweber [5] for S1S specifications with an algorithm of non-elementary complexity. For Linear Temporal Logic (LTL) specifications, the problem has been shown to be 2EXPTIME-complete [21], [22]. In the attempt of making reactive synthesis a practical task, in spite of its very high complexity, research mainly focused on two lines: (i) finding good algorithms for the average case; (ii) restricting the expressiveness of the specification language. Important examples of the first line of research are the contribution by Kupferman and Vardi [15], where the authors devise a procedure to avoid Safra's determinization of Büchi automata (a known bottleneck in all the problems requiring a determinization of a Büchi automaton), and the work by Finkbeiner and Schewe [11], where the problem is reduced to a sequence of smaller problems on safety automata, obtained by bounding the number of visits to a rejecting state of a co-Büchi automaton. A meaningful example of restrictions to the specification language is the definition of the *Generalized Reactivity(1)* logic [20], whose synthesis problem can be solved in $\mathcal{O}(N^3)$ symbolic steps, where N is the size of the arena. Finally, in [25] Zhu et al. consider reactive synthesis from Safety LTL specifications. Although the complexity remains doubly exponential, the proposed restriction allows one to reason on finite words and thus to exploit efficient tools for finite-state automata, like, for instance, MONA [12].

In this paper, we propose a new fragment of LTL, called *Extended Bounded Response* LTL ($\mathsf{LTL}_{\mathsf{EBR}}$ for short), which supports *bounded* operators [18], such as $\mathsf{G}^{[a,b]}$ and $\mathsf{F}^{[a,b]}$, along with universal unbounded temporal operators like G and $\mathcal{R}$. We show that formulas of $\mathsf{LTL}_{\mathsf{EBR}}$ can be turned into *deterministic symbolic automata* over infinite words, with a translation carried out in a completely symbolic way. Such a result is achieved in two steps: (i) a *pastification* of the subformulas containing only bounded operators by making use of techniques similar to those exploited for MTL [17], [18], and (ii) the construction of *deterministic monitors* for the unbounded temporal operators. These two steps allow the entire procedure to be carried out without ever producing any explicit automaton. Then, we use existing algorithms for safety synthesis to solve the game on the deterministic symbolic automaton. We implemented the proposed solution in a tool, called ebr-ltl-synth, and compared its performance against state-of-the-art synthesizers for full LTL over a set of $\mathsf{LTL}_{\mathsf{EBR}}$ formulas. The outcomes of the experimental evaluation are encouraging. For lack of space, some of the proofs are reported in [8].

II. Preliminaries

Linear Temporal Logic with Past (LTL+P) is a modal logic interpreted over infinite state sequences. Let Σ be a set of propositions. LTL+P formulas are inductively defined as follows:

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{X}\phi \mid \phi_1 \,\mathcal{U}\, \phi_2 \mid \mathsf{Y}\phi \mid \phi_1 \,\mathcal{S}\, \phi_2$$

where $p \in \Sigma$. Temporal operators can be subdivided into the *future operators*, *next* (X) and *until* ($\mathcal{U}$), and *past operators*, *yesterday* (Y) and *since* ($\mathcal{S}$). We define the following common abbreviations (where $\top$ stands for true): (i) $\mathsf{X}^i\phi$ is $\mathsf{X}(\mathsf{X}^{i-1}\phi)$

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_15

if $i > 0$ and $\mathsf{X}^0\phi$ is ϕ; (ii) *release*: $\phi_1 \mathcal{R}\, \phi_2 \equiv \neg(\neg\phi_1 \mathcal{U} \neg\phi_2)$; (iii) *eventually*: $\mathsf{F}\phi_1 \equiv \top \mathcal{U}\, \phi_1$; (iv) *globally*: $\mathsf{G}\phi_1 \equiv \neg\mathsf{F}\neg\phi_1$; (v) *trigger*: $\phi_1 \mathcal{T} \phi_2 \equiv \neg(\neg\phi_1 \mathcal{S} \neg\phi_2)$; (vi) *once*: $\mathsf{O}\phi_1 \equiv \top \mathcal{S} \phi_1$; (vii) *historically*: $\mathsf{H}\phi_1 \equiv \neg\mathsf{O}\neg\phi_1$.

LTL is obtained from LTL+P by allowing only the *next* and the *until* operators. Conversely, *Full Past* LTL ($\mathsf{LTL_{FP}}$) is the fragment of LTL+P that only admits past operators.

LTL can also be enriched with *bounded* temporal operators, such as the *bounded until* ($\phi_1 \mathcal{U}^{[a,b]}\, \phi_2$) and *bounded eventually* ($\mathsf{F}^{[a,b]}\phi_1 \equiv \top\, \mathcal{U}^{[a,b]}\, \phi_1$). *Full Bounded* LTL ($\mathsf{LTL_{FB}}$) is the fragment of LTL that includes only the *next*, *bounded until*, and *bounded eventually* operators.

Let us now give the semantics of the above logics. A *state sequence* is an infinite sequence $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in (2^\Sigma)^\omega$ of sets of propositions $\sigma_i \in 2^\Sigma$, called *states*. Given a sequence σ, a position $i \geq 0$, and a formula ϕ, the satisfaction of ϕ by σ at i, written $\sigma, i \models \phi$, is inductively defined as follows:

$\sigma, i \models p$	iff	$p \in \sigma_i$
$\sigma, i \models \neg\phi$	iff	$\sigma, i \not\models \phi$
$\sigma, i \models \phi_1 \vee \phi_2$	iff	either $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$
$\sigma, i \models \phi_1 \wedge \phi_2$	iff	$\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$
$\sigma, i \models \mathsf{X}\phi$	iff	$\sigma, i+1 \models \phi$
$\sigma, i \models \mathsf{Y}\phi$	iff	$i > 0$ and $\sigma, i-1 \models \phi$
$\sigma, i \models \phi_1 \mathcal{U}\, \phi_2$	iff	there exists $j \geq i$ such that $\sigma, j \models \phi_2$ and $\sigma, k \models \phi_1$ for all $i \leq k < j$
$\sigma, i \models \phi_1 \mathcal{S}\, \phi_2$	iff	there exists $j \leq i$ such that $\sigma, j \models \phi_2$ and $\sigma, k \models \phi_1$ for all $j < k \leq i$
$\sigma, i \models \phi_1 \mathcal{U}^{[a,b]}\, \phi_2$	iff	there exists $j \in [i+a, i+b]$ such that $\sigma, j \models \phi_2$ and $\sigma, k \models \phi_1$ for all $i \leq k < j$

We say that σ satisfies ϕ, written $\sigma \models \phi$, if and only if $\sigma, 0 \models \phi$. We define the *language* $\mathcal{L}(\phi)$ of a temporal formula ϕ as $\mathcal{L}(\phi) = \{\sigma \in (2^\Sigma)^\omega \mid \sigma \models \phi\}$.

Symbolic safety automata and safety games

To begin with, we formally define the problems of realizability and reactive synthesis for temporal formulas.

As for realizability, it is convenient to view it as a two-player game between Controller, whose aim is to satisfy the specification, and Environment, who tries to violate it.

Definition 1 (Strategy): Let $\Sigma = \mathcal{C} \cup \mathcal{U}$ be an alphabet partitioned into the set of *controllable* variables $\mathcal{C}$ and the set of *uncontrollable* ones $\mathcal{U}$, such that $\mathcal{C} \cap \mathcal{U} = \varnothing$. A *strategy for Controller* is a function $g : (2^\mathcal{U})^+ \to 2^\mathcal{C}$ that, given the sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices made by *Environment* so far, determines the current choices $\mathsf{C}_n = g(\mathsf{U})$ of *Controller*.

Given a strategy $g : (2^\mathcal{U})^+ \to 2^\mathcal{C}$ and an infinite sequence of uncontrollable choices $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^\mathcal{U})^\omega$, with some abuse of notation, we denote as $g(\mathsf{U}) = \langle \mathsf{U}_0 \cup g(\langle \mathsf{U}_0 \rangle), \mathsf{U}_1 \cup g(\langle \mathsf{U}_0, \mathsf{U}_1 \rangle), \ldots \rangle$ the state sequence resulting from reacting to U according to g.

Definition 2 (Realizability and Synthesis): Let ϕ be a temporal formula over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. We say that ϕ is *realizable* if and only if there exists a strategy $g : (2^\mathcal{U})^+ \to 2^\mathcal{C}$ such that, for any infinite sequence $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^\mathcal{U})^\omega$, it holds that $g(\mathsf{U}) \models \phi$. If ϕ is realizable, the synthesis problem is the problem of computing such a strategy g.

Temporal logic has an intimate relationship with automata on infinite words [24], where different acceptance conditions give rise to different classes of automata. For instance, the acceptance condition of (non-deterministic) Büchi automata allows them to recognize the class of ω-regular languages [4], including all languages definable by LTL+P formulas.

Here, we focus on a restricted type of acceptance condition, called *safety* condition, and we represent automata in a *symbolic* way, as opposed to their common explicit representation.

Definition 3 (Symbolic Safety Automata): A *symbolic safety automaton* (SSA) is a tuple $\mathcal{A} = (V, I, T, S)$, such that (i) $V = X \cup \Sigma$, where X is a set of *state variables* and Σ is a set of *input variables*, and (ii) $I(X)$, $T(X, \Sigma, X')$, and $S(X)$, with $X' = \{x' \mid x \in X\}$, are Boolean formulae which define the set of initial states, the transition relation, and the set of safe states, respectively.

In symbolic automata, states are identified by the values of state variables, and both initial/final states and the transition relation are represented as Boolean formulas. This allows them to be, in many cases, exponentially more succinct than equivalent explicitly represented automata. In particular, the transition relation $T(X, \Sigma, X')$ is built over state variables, input variables, and a *primed* version of state variables that represent the values of state variables at the next state. As an example, if a variable x has to flip at every transition, the transition relation would contain a clause of the form $x \Leftrightarrow \neg x'$.

Definition 4 (Acceptance of SSA): Let $\mathcal{A}$ be an SSA. A *trace* is a sequence $\tau = \langle \tau_0, \tau_1, \ldots \rangle \in (2^V)^\omega$ of subsets τ_i of V that satisfies the transition relation of $\mathcal{A}$, that is, such that for all $i \geq 0$, $T(X, \Sigma, X')$ is satisfied when τ_i is used to interpret variables from X and Σ, and τ_{i+1} is used to interpret variables from X'. We say that a trace τ is *induced* by a word $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle \in (2^\Sigma)^\omega$ iff $\sigma_i = \tau_i \cap \Sigma$ for all $i \geq 0$. A trace τ is *accepting* (or *safe*) iff τ_i satisfies $S(X)$ for all $i \geq 0$. The *language* of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of all $\sigma \in (2^\Sigma)^\omega$ such that there exists an accepting trace induced by σ in $\mathcal{A}$.

For reactive synthesis, a crucial property of an automaton $\mathcal{A}$ is *determinism*, since in order to check if $\sigma \in \mathcal{L}(\mathcal{A})$ it suffices to check if *the* trace induced by σ in $\mathcal{A}$ is accepting.

Definition 5 (Deterministic SSA): An SSA $\mathcal{A} = (V, I, T, S)$ is *deterministic* if:

1) the formula I has exactly one satisfying assignment;
2) the transition relation is of the form:

$$T(X, \Sigma, X') \coloneqq \bigwedge_{x \in X} (x' \Leftrightarrow \beta_x(X \cup \Sigma))$$

where each $\beta_x(X \cup \Sigma)$ is a Boolean formula over X and Σ.

Note that Def. 5 implies that for each $\sigma \in (2^\Sigma)^\omega$, there exists exactly one trace induced by σ for any given deterministic SSA. The realizability and the synthesis problems can be defined over a deterministic automaton as well; this gives rise to a safety game, which is defined as follows.

Definition 6 (Safety Game): Let $\mathcal{A}$ be a deterministic SSA over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. A safety game is a tuple $G = \langle \mathcal{A}, \mathcal{C}, \mathcal{U} \rangle$, where $\mathcal{C}$ and $\mathcal{U}$ are the sets of controllable and uncontrollable variables, respectively. We say that Controller wins the game if and only if there is a strategy $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ such that for all sequences $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^{\mathsf{U}})^\omega$, *the* trace τ induced by $g(\mathsf{U})$ in $\mathcal{A}$ is *accepting*.

III. Extended Bounded Response LTL

In this section, we define *Extended Bounded Response* LTL, abbreviated $\mathsf{LTL_{EBR}}$. $\mathsf{LTL_{EBR}}$ extends $\mathsf{LTL_{FB}}$ (which only features bounded operators) by admitting Boolean combinations of the universal unbounded temporal operators *release* ($\mathcal{R}$) and *globally* (G).

Definition 7 (The logic $\mathsf{LTL_{EBR}}$*):* Let $a, b \in \mathbb{N}$. An $\mathsf{LTL_{EBR}}$ formula χ is inductively defined as follows:

$$\begin{aligned} \psi &:= p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathcal{U}^{[a,b]}\, \psi_2 && \text{Full Bounded Layer} \\ \phi &:= \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathcal{R}\, \phi && \text{Future Layer} \\ \chi &:= \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 && \text{Boolean Layer} \end{aligned}$$

We refer to Sec. II for the semantics of $\mathsf{LTL_{EBR}}$ operators. In the next sections, we will show how to build, given an $\mathsf{LTL_{EBR}}$ formula ϕ, a deterministic symbolic safety automaton $\mathcal{A}(\phi)$ such that $\mathcal{L}(\mathcal{A}(\phi)) = \mathcal{L}(\phi)$.

A. Examples

We now give some simple examples of requirements that can be expressed in the $\mathsf{LTL_{EBR}}$ logic.

The first one is a typical bounded response requirement: Controller has to answer a grant g at most k time units after the request r of Environment is issued. It can be expressed by the following $\mathsf{LTL_{EBR}}$ formula:

$$\mathsf{G}(r \to \mathsf{F}^{[0,k]} g)$$

Another quite common requirement is *mutual exclusion*. As an example, the case of an arbiter that has to grant a resource to at most one client at once can be captured as follows (for each i, g_i means that the resource has been granted to client i):

$$\mathsf{G}(\bigwedge_{1 \le i < j \le n} \neg(g_i \wedge g_j))$$

When a set of clients with different priorities has to be managed, it is possible to introduce a requirement stating that, whenever two or more clients simultaneously send a request, clients with a higher priority must be granted before those with a lower one ($i < j$ means that the priority of client i is higher than that of client j):

$$\bigwedge_{1 \le i < j \le n} \mathsf{G}((r_i \wedge r_j) \to (\neg g_j) \,\mathcal{U}^{[0,k]}\, g_i)$$

Finally, in many situations it is important to include requirements about the *configuration* of a system model. Consider the case of a thermostat. One may ask that if the `prog` modality is off, then the controller has to communicate the signal `on` to the boiler for an indefinitely long amount of time, while, in case the `prog` modality is on, it has to do that only for a specific interval of time, say $[h_1, h_2]$, after which it has to stop the communication with the boiler. This can be expressed in $\mathsf{LTL_{EBR}}$ by the following formula:

$$(\neg\mathtt{prog} \wedge \mathsf{G}(\mathtt{on})) \vee (\mathtt{prog} \wedge \mathsf{G}^{[h_1,h_2]}(\mathtt{on}) \wedge \mathsf{X}^{h_2}\mathsf{G}(\mathtt{off}))$$

B. Comparison with other temporal logics

Zhu *et al.* [25] studied the synthesis problem for *Safety* LTL, which can be viewed as the *until*-free fragment of LTL in negated normal form (NNF). Every formula ϕ of $\mathsf{LTL_{EBR}}$ can be turned into a Safety LTL one by (i) transforming ϕ in NNF and (ii) expanding each bounded operator in terms of conjunctions or disjunctions. As an example, the $\mathsf{LTL_{EBR}}$ formula $\phi := \mathsf{G}(p \to \mathsf{F}^{[0,5]} q)$ is equivalent to the Safety LTL formula $\phi' := \mathsf{G}(p \to \bigvee_{i=0}^{5} \mathsf{X}^i q)$. However, since constants in $\mathsf{LTL_{EBR}}$ are represented by using a logarithmic encoding, $\mathsf{LTL_{EBR}}$ formulas can be exponentially more succinct than Safety LTL ones. Whether the converse holds as well, *i.e.*, whether any formula of Safety LTL can be translated into an equivalent $\mathsf{LTL_{EBR}}$ one, is still an open question. As an example, $\mathsf{G}(p \vee \mathsf{G}q)$ is a Safety LTL formula but, syntactically, is not an $\mathsf{LTL_{EBR}}$ one.

Maler *et al.* [18] introduced *Metric Temporal Logic with a Bounded-Horizon* (MTL−B for short) as the metric temporal logic with *only* bounded operators interpreted over dense time. They addressed the problem of reactive synthesis from MTL−B specifications by showing that each MTL−B formula can be transformed into a *deterministic* timed automaton. With respect to this fragment, and ignoring the differences in the underlying temporal structures (in our setting, time is discrete), $\mathsf{LTL_{EBR}}$ extends MTL−B with Boolean combinations of unbounded universal temporal operators.

IV. From $\mathsf{LTL_{EBR}}$ to Deterministic Symbolic Safety Automata

This section focuses on the procedure to turn every $\mathsf{LTL_{EBR}}$ formula into a deterministic symbolic safety automaton on infinite words (see Def. 5) that recognizes the same language.

In doing that, we apply a few transformation steps on the formula, summarized in Fig. 1, to simplify its syntactic structure and turn it into a form amenable to direct transformation into a deterministic SSA. We define two syntactic restrictions of $\mathsf{LTL_{EBR}}$ that are the targets of the transformation steps.

Definition 8 ($\mathsf{PastLTL_{EBR}}$*):* An $\mathsf{PastLTL_{EBR}}$ formula χ is inductively defined as follows:

$$\begin{aligned} \psi &:= p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{Y}\psi \mid \psi_1 \,\mathcal{S}\, \psi_2 \\ \phi &:= \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid (\mathsf{X}^i\psi) \,\mathcal{R}\, \phi \\ \chi &:= \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \end{aligned}$$

Definition 9 (Canonical $\mathsf{PastLTL_{EBR}}$*):* The *canonical form* of $\mathsf{PastLTL_{EBR}}$ formulas is inductively defined as follows:

$$\begin{aligned} \psi &:= p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{Y}\psi \mid \psi_1 \,\mathcal{S}\, \psi_2 \\ \phi &:= \psi \mid \mathsf{G}\psi \mid \psi_1 \,\mathcal{R}\, \psi_2 \\ \lambda &:= \phi \mid \mathsf{X}\lambda \\ \chi &:= \lambda \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \end{aligned}$$

$$\begin{tikzcd}
\mathsf{LTL}_{\mathsf{EBR}}\ \phi \arrow[d, "\cdot\ \mathsf{toPastLtlEbr}"] \\
\mathsf{PastLTL}_{\mathsf{EBR}}\ \phi \arrow[d, "\cdot\ \mathsf{canonize}"] \\
\text{Canonical } \mathsf{PastLTL}_{\mathsf{EBR}}\ \phi \arrow[d, "\cdot\ \mathsf{ltl2smv}"] \\
\text{SSA } \mathcal{A}(\phi) \arrow[d, "\cdot\ \mathsf{fsmv2aig}"] \\
\text{AIGER} \arrow[d, "\cdot\ \text{call to a safety synthesizer}"] \\
\text{result (real./unreal.)}
\end{tikzcd}$$

Figure 1. The overall procedure.

Canonical $\mathsf{PastLTL}_{\mathsf{EBR}}$ formulas do not contain nested occurrences of unbounded temporal operators, whose operands can be only full-past formulas, and each of these is prefixed by an arbitrary number of *next* operators.

The transformation of $\mathsf{LTL}_{\mathsf{EBR}}$ formulas into deterministic SSAs consists of three steps: (i) a translation from $\mathsf{LTL}_{\mathsf{EBR}}$ to $\mathsf{PastLTL}_{\mathsf{EBR}}$; (ii) a translation from $\mathsf{PastLTL}_{\mathsf{EBR}}$ to its canonical form; (iii) a transformation of canonical $\mathsf{PastLTL}_{\mathsf{EBR}}$ formulas into deterministic SSAs. Once a deterministic SSA $\mathcal{A}(\phi)$ for the original $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ over $\mathcal{C} \cup \mathcal{U}$ has been obtained, to solve the safety game $\langle \mathcal{A}(\phi), \mathcal{C}, \mathcal{U} \rangle$, *i.e.*, to decide the existence of a strategy for Controller in the automaton, we apply an existing safety synthesis algorithm (see Def. 6).

A. *From* $\mathsf{LTL}_{\mathsf{EBR}}$ *to* $\mathsf{PastLTL}_{\mathsf{EBR}}$

Let ϕ be an $\mathsf{LTL}_{\mathsf{EBR}}$ formula. The first step consists in translating each $\mathsf{LTL}_{\mathsf{FB}}$ subformula of ϕ into an *equivalent* one, which is of the form $X^d\psi$, with $\psi \in \mathsf{LTL}_{\mathsf{FP}}$ and $d \in \mathbb{N}$. We refer to this process as *pastification* [17], [18]. As we will see, since "the past has already happened", full-past formulas can be represented by deterministic monitors.

In order to pastify each $\mathsf{LTL}_{\mathsf{FB}}$ subformula of ϕ, we adapt to $\mathsf{LTL}_{\mathsf{EBR}}$ a technique developed by Maler *et al.* for $\mathsf{MTL}{-}\mathsf{B}$ [17], [18]. Intuitively, for each model of a full-bounded formula ϕ, there exists a furthermost time point d (the *temporal depth* of ϕ) such that the subsequent states cannot be constrained by ϕ in any way. The *pastification* of ϕ is a formula that uses only past operators and that is equivalent to ϕ when interpreted at time point d instead of at the origin.

Definition 10 (Temporal Depth [18]): Let ϕ be an $\mathsf{LTL}_{\mathsf{FB}}$ formula. The *temporal depth* of ϕ, denoted as $D(\phi)$, is inductively defined as follows:

- $D(p) = 0$, for all $p \in \Sigma$
- $D(\neg\phi_1) = D(\phi_1)$
- $D(\phi_1 \wedge \phi_2) = \max\{D(\phi_1), D(\phi_2)\}$
- $D(\mathsf{X}\phi_1) = 1 + D(\phi_1)$
- $D(\phi_1 \,\mathcal{U}^{[a,b]}\, \phi_2) = b + \max\{D(\phi_1), D(\phi_2)\}$

Let M_ϕ (only M if unambiguous) be the greatest constant in ϕ, with $M_\phi = 0$ if ϕ has no constants. It can be observed that $D(\phi) \le M \cdot n$, where $n = |\phi|$.

Definition 11 (Pastification [18]): Let ϕ be an $\mathsf{LTL}_{\mathsf{FB}}$ formula and $d \ge D(\phi)$. The pastification of ϕ is the formula $\Pi(\phi, d)$ inductively defined as follows:

- $\Pi(p, d) = \mathsf{Y}^d p$
- $\Pi(\neg\phi, d) = \neg\Pi(\phi, d)$
- $\Pi(\phi_1 \wedge \phi_2, d) = \Pi(\phi_1, d) \wedge \Pi(\phi_2, d)$
- $\Pi(\mathsf{X}\phi, d) = \Pi(\phi, d-1)$
- $\Pi(\phi_1 \,\mathcal{U}^{[a,b]}\, \phi_2, d) =$ $\bigvee_{t=0}^{b-a}(\mathsf{Y}^t(\Pi(\phi_2, d-b) \wedge \mathsf{H}^{b-t-1}\mathsf{Y}\Pi(\phi_1, d-b)))$

Note that from Def. 11 we can derive that $\Pi(\mathsf{F}^{[a,b]}\phi, d) \equiv \Pi(\top \,\mathcal{U}^{[a,b]}\, \phi, d) \equiv \bigvee_{t=0}^{b-a} \mathsf{Y}^t\Pi(\phi, d-b)$, which can be succinctly written using the *once* operator, hence we can define $\Pi(\mathsf{F}^{[a,b]}\phi, d) = \mathsf{O}^{[0,b-a]}\Pi(\phi, d-b)$.

Proposition 1 (Soundness of pastification): Let φ be a $\mathsf{LTL}_{\mathsf{FB}}$ formula. For all state sequences $\sigma \in (2^\Sigma)^\omega$, all $i \in \mathbb{N}$, and all $d \ge D(\phi)$, it holds that:

$$\sigma, i \models \varphi \;\Leftrightarrow\; \sigma, i \models \mathsf{X}^d\Pi(\varphi, d)$$

From now on, let $\mathsf{pastify}(\phi)$ be the formula $X^{D(\phi)}\Pi(\phi, D(\phi))$. As an example, if $\phi := \mathsf{F}^{[0,k_1]}(q \wedge \mathsf{F}^{[0,k_2]}p)$, then $\mathsf{pastify}(\phi) := X^{k_1+k_2}\mathsf{O}^{[0,k_1]}(\mathsf{Y}^{k_2}q \wedge \mathsf{O}^{[0,k_2]}p)$. We state the following complexity result about pastification.

Proposition 2: Let ϕ be a $\mathsf{LTL}_{\mathsf{FB}}$ formula. Then, $\mathsf{pastify}(\phi)$ is a formula of size $\mathcal{O}(n^2 \cdot M^{\log_2 n+1})$, where $n = |\phi|$ and M is the greatest constant in ϕ.

Note that if ϕ has no constants, that is, $M = 1$, the size of $\mathsf{pastify}(\phi)$ is $\mathcal{O}(n^2)$. Given an $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ, we pastify each of its $\mathsf{LTL}_{\mathsf{FB}}$ subformulas with the $\mathsf{pastify}$ operator: we call this step $\mathsf{toPastLtlEbr}$. Once it has been completed, the resulting formula belongs to $\mathsf{PastLTL}_{\mathsf{EBR}}$.

The $\mathsf{toPastLtlEbr}$ algorithm can be improved by observing that there are $\mathsf{LTL}_{\mathsf{FB}}$ formulas that already belong to $\mathsf{PastLTL}_{\mathsf{EBR}}$. One example is the formula $p \wedge \mathsf{XXX}q$. Obviously, for this kind of formulas there is no need for the algorithm to pastify them. Consider the previous example. Without the proposed trick, the algorithm would have produced the formula $\mathsf{XXX}(\mathsf{YYY}p \wedge q)$, while, by simply noticing that the formula already belongs to $\mathsf{PastLTL}_{\mathsf{EBR}}$, it does not need to pastify anything, returning $p \wedge \mathsf{XXX}q$.

Proposition 3: For each $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ, there is an equivalent $\mathsf{PastLTL}_{\mathsf{EBR}}$ formula ϕ' of size $\mathcal{O}(n^3 \cdot M^{\log_2 n+1})$, where $n = |\phi|$ and M is the greatest constant in ϕ.

Proof: Let ϕ be an $\mathsf{LTL}_{\mathsf{EBR}}$ formula and let $\phi' := \mathsf{toPastLtlEbr}(\phi)$. By Prop. 1, the $\mathsf{toPastLtlEbr}$ algorithm replaces the $\mathsf{LTL}_{\mathsf{FB}}$ subformulas of ϕ with an equivalent formula, hence $\phi \equiv \phi'$. Since in ϕ there are at most $n = |\phi|$ subformulas, then, by Prop. 2, $|\phi'| = n \cdot \mathcal{O}(n^2 \cdot M^{\log_2 n+1})$, that is, $|\phi'| = \mathcal{O}(n^3 \cdot M^{\log_2 n})$. ■

Note that if there are no constants in ϕ, that is, $M = 1$, then, by Prop. 2, $|\mathsf{toPastLtlEbr}(\phi)| = \mathcal{O}(n^3)$.

B. *From* $\mathsf{PastLTL_{EBR}}$ *to Canonical* $\mathsf{PastLTL_{EBR}}$

The second step is the canonization of the $\mathsf{PastLTL_{EBR}}$ formula obtained from the previous step, in order to produce an equivalent formula in canonical form (Def. 9). Canonical $\mathsf{PastLTL_{EBR}}$ formulas are Boolean combinations of formulas of the form $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)$, where ψ_1 and ψ_2 are full past formulas. Compared to general $\mathsf{PastLTL_{EBR}}$ formulas, those in canonical form do not admit neither nested unbounded operators nor *next* operators in front of the left-hand argument of a *release*. The canonization of a $\mathsf{PastLTL_{EBR}}$ formula is obtained by applying a set of rewriting rules.

Definition 12 (Canonization): Given a $\mathsf{PastLTL_{EBR}}$ formula ϕ, $\mathsf{canonize}(\phi)$ is the formula obtained by recursively applying the R_1-R_7 rules to the subformulas of ϕ in a bottom-up fashion followed by the application of the R_{flat} rule:

$$R_1 : \mathsf{X}(\psi_1 \wedge \psi_2) \rightsquigarrow \mathsf{X}\psi_1 \wedge \mathsf{X}\psi_2$$

$$R_2 : \psi \,\mathcal{R}\, (\psi_1 \wedge \psi_2) \rightsquigarrow \psi \,\mathcal{R}\, \psi_1 \wedge \psi \,\mathcal{R}\, \psi_2$$

$$R_3 : (\mathsf{X}^i\psi_1) \,\mathcal{R}\, (\mathsf{X}^j\psi_2) \rightsquigarrow \begin{cases} \mathsf{X}^i(\psi_1 \,\mathcal{R}\, (\mathsf{Y}^{i-j}\psi_2)) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathcal{R}\, \psi_2) & \text{otherwise} \end{cases}$$

$$R_4 : (\mathsf{X}^i\psi_1) \,\mathcal{R}\, (\mathsf{X}^j(\psi_2 \,\mathcal{R}\, \psi_3)) \rightsquigarrow \begin{cases} \mathsf{X}^i(\psi_1 \,\mathcal{R}\, ((\mathsf{Y}^{i-j}\psi_2) \,\mathcal{R}\, (\mathsf{Y}^{i-j}\psi_3))) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathcal{R}\, (\psi_2 \,\mathcal{R}\, \psi_3)) & \text{otherwise} \end{cases}$$

$$R_5 : \mathsf{G}\mathsf{X}^i\mathsf{G}\psi \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi$$

$$R_6 : \mathsf{G}\mathsf{X}^i(\psi_1 \,\mathcal{R}\, \psi_2) \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi_2$$

$$R_7 : (\mathsf{X}^i\psi_1) \,\mathcal{R}\, (\mathsf{X}^j\mathsf{G}\psi_2) \rightsquigarrow \begin{cases} \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2 & \text{if } i > j \\ \mathsf{X}^j\mathsf{G}\psi_2 & \text{otherwise} \end{cases}$$

$$R_{\mathit{flat}} : \mathsf{X}^i(\psi_1 \,\mathcal{R}\, (\psi_2 \,\mathcal{R}\, (\ldots(\psi_{n-1} \,\mathcal{R}\, \psi_n)\ldots))) \rightsquigarrow \mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top)\ldots)) \,\mathcal{R}\, \psi_n) \quad \text{for any } n \geq 3$$

where ψ, ψ_1, ψ_2, and ψ_3 are full-past formulae.

It is worth noticing that, so far, we do not have rules (preserving equivalence) to deal with the following cases: (i) $(\phi_1 \wedge \phi_2) \,\mathcal{R}\, (\phi)$, (ii) $(\mathsf{G}\phi_1) \,\mathcal{R}\, (\phi)$ or (iii) $(\phi_1 \,\mathcal{R}\, \phi_2) \,\mathcal{R}\, (\phi)$. This is why, in Def. 7, we restricted the left-hand argument of each *release* operator to be a full-bounded formula.

Lemma 1 (Soundness of $\mathsf{canonize}(\cdot)$*):* For any $\mathsf{PastLTL_{EBR}}$ formula ϕ, it holds that ϕ and $\mathsf{canonize}(\phi)$ are equivalent and $\mathsf{canonize}(\phi)$ is a Canonical $\mathsf{PastLTL_{EBR}}$ formula.

Proposition 4 (Complexity of $\mathsf{canonize}(\cdot)$*):* For any $\mathsf{PastLTL_{EBR}}$ formula ϕ, $\mathsf{canonize}(\phi)$ can be built in $\mathcal{O}(n)$ time, and the size of $\mathsf{canonize}(\phi)$ is $\mathcal{O}(n)$, where $n = |\phi|$.

C. *From Canonical* $\mathsf{PastLTL_{EBR}}$ *to deterministic SSA*

The particular shape of canonical $\mathsf{PastLTL_{EBR}}$ formulas makes it possible to encode the specification into deterministic SSAs. The key observation is that $\mathsf{LTL_{FP}}$ formulas can be encoded into deterministic automata: since these formulas talk exclusively about the past, their truth can be evaluated at any single step depending only on previous steps, without making any guess about the future ("the past already happened"). But $\mathsf{LTL_{FP}}$ formulae are not the only ones that can be encoded deterministically. Consider, for instance, the formula $\phi \equiv \mathsf{X}p \vee \mathsf{X}q$. At a first glance, it may seem that ϕ needs a non-deterministic automaton to be encoded, which at the first state makes a choice about whether p or q will hold in the next state. Nevertheless, this formula is equivalent to $X(p \vee q)$ and it corresponds to the *deterministic* automaton that, once arrived in its second state by reading any proposition symbol, proceeds to an accepting state by reading either p or q, or goes to a sink (*error*) state otherwise.

$\mathsf{PastLTL_{EBR}}$ in its canonical form combines full past formulas into a broader language that can still be turned into symbolic deterministic automata, extending the above intuition and exploiting the *monitorability* of *universal* temporal operators.

Monitoring is a technique coming from *runtime verification* [16]. Consider the formula $\mathsf{G}\alpha$. By observing a state sequence, at each step we can decide if a *violation* has occurred; indeed, if α is false at the current step, then the value of $\mathsf{G}\alpha$ is certainly false for each of the previous steps. More generally, universal temporal formulas, such as $\mathsf{G}\phi$ and $\phi_1 \,\mathcal{R}\, \phi_2$, are *monitorable*, meaning that a violation of them can be decided on the basis of the observation of a *finite* number of steps. In particular, reporting an error in the next state can be done by considering only the current values. This means that any universal temporal operator can be monitored by adding a Boolean *error variable* with a *deterministic* transition relation.

Therefore, despite not being able to evaluate the truth of a formula such as $\mathsf{G}\alpha$, as it can be done in the case of past operators, we can nevertheless state in the accepting condition that an error state can never be reached. In this way, if the trace is accepting, that is, an error state can never be reached, then we know that there are no violations, *e.g.*, for $\mathsf{G}\alpha$, we have forced α to be true in every state. Otherwise, if the trace is not accepting, that is, an error state is reachable, we know that there is a (finite) violation and that the temporal formula was falsified at some step. We therefore introduce an *error bit* for each $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \,\mathcal{R}\, \psi_2)$ of a canonical $\mathsf{PastLTL_{EBR}}$ formula.

Let ϕ be a canonical $\mathsf{PastLTL_{EBR}}$ formula over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. We define the deterministic SSA $\mathcal{A}(\phi) = (V, I, T, S)$ as follows:

- *Variables.* The set of *state variables* of the automaton is defined as $X = X_P \cup X_F \cup X_C$, where:

$$X_P = \{v_\alpha \mid \alpha \text{ is an } \mathsf{LTL_{FP}} \text{ subformula of } \phi\}$$

$$X_F = \left\{ \mathit{error}_\varphi \;\middle|\; \begin{array}{l} \varphi \text{ is subformula of } \phi \text{ of the form} \\ \mathsf{X}^i\psi,\ \mathsf{X}^i\mathsf{G}\psi, \text{ or } \mathsf{X}^i(\psi_1 \,\mathcal{R}\, \psi_2) \end{array} \right\}$$

$$X_C = \left\{ \mathit{counter}_i \;\middle|\; \begin{array}{l} i \in \{0, \ldots, \log_2 d\} \\ d \text{ max. among all } \mathsf{X}^d\psi \text{ in } \phi. \end{array} \right\}$$

Intuitively, variables in X_P track the truth value of all the full-past subformulas, variables in X_F implement the above-described monitoring mechanism, and variables in

X_C are used to encode a binary counter used to monitor nested *tomorrow* operators. In particular, for n nested *tomorrow* operators, a counter with $\log_2(n)$ bits is needed.
- *Initial state.* All the state variables, including the counter bits, are initially false, that is, $I(X) = \bigwedge_{x \in X} \neg x$.
- *Transition relation.* $T(X, \Sigma, X')$ is the conjunction of the transition *functions* of the binary counter and the monitors of each subformula of ϕ, as will be defined later. Notice that each conjunct is of the form $x' \Leftrightarrow \beta(X \cup \Sigma)$, and thus it is a deterministic transition relation.
- *Safety condition.* $S(X)$ is a Boolean formula obtained from ϕ by replacing each formula $\varphi \in X_F$ by $\neg error_\varphi$, *i.e.*, $S(X) = \phi[\varphi/\neg error_\varphi]$.

We now define the monitors for the binary counter, used to handle nested *tomorrow* operators, any formula $\psi \in \mathsf{LTL_{FP}}$, and any canonical $\mathsf{PastLTL_{EBR}}$ formula of one of the forms $\mathsf{X}^i\psi_1$, $\mathsf{X}^i\mathsf{G}\psi_1$, and $\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)$. We give the definition of the monitors using the SMV language [6], as it provides useful shorthands (like the *switch-case* primitive). Each of the following SMV statement corresponds to the Boolean formula that defines transition functions of our monitors.

The monitor for the counter is defined as follows:

```
next(counter_0) := $\neg$ counter_0
next(counter_i) := (counter_{i-1} $\vee$ counter_i) $\wedge$ $\neg$counter_i
```

If $\psi := \alpha \,\mathcal{S}\, \beta$ or $\mathsf{Y}\alpha$, its monitor is defined as follows:

```
next($v_{\mathsf{Y}\alpha}$) := $v_\alpha \wedge$ counter> 0
DEFINE
  $v_{\alpha\mathcal{S}\beta}$ := $v_\beta \vee (v_\alpha \wedge v_{\mathsf{Y}(\alpha\mathcal{S}\beta)})$
```

If ψ is a propositional atom, a negation, or a disjunction of full-past formulas, we define its monitor as follows:

```
DEFINE
  $v_p$ := $p$
  $v_{\neg\alpha}$ := $\neg v_\alpha$
  $v_{\alpha\vee\beta}$ := $v_\alpha \vee v_\beta$
```

For each formula ϕ of type $\mathsf{X}^i\psi$, where ψ is a full-past formula, we introduce a new error bit $error_\phi$. Its monitor is defined as follows:

```
next($error_{\mathsf{X}^i\psi}$) := case
    $error_{\mathsf{X}^i\psi}$ : TRUE;
    $counter = i \wedge \neg v_\psi$ : TRUE;
    TRUE : FALSE;
  esac
```

If $\phi := \mathsf{X}^i\mathsf{G}\psi$, where ψ is a full-past formula, we introduce a new error bit $error_\phi$, and we define its monitor as follows:

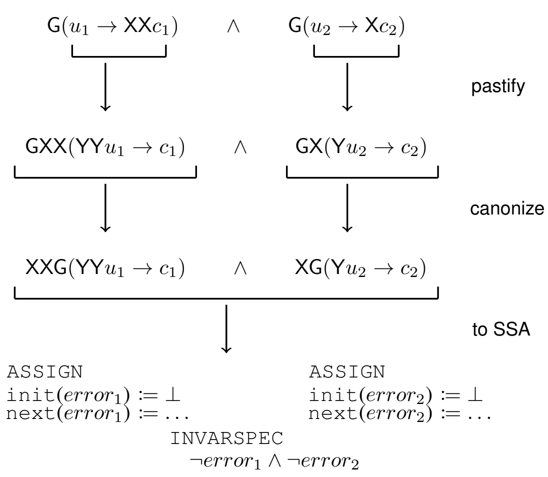
```
next($error_{\mathsf{X}^i\mathsf{G}\psi}$) := case
    $counter < i$ : FALSE;
    $\neg error_{\mathsf{X}^i\mathsf{G}\psi} \wedge v_\psi$ : FALSE;
    TRUE : TRUE;
  esac
```

The same for $\phi := \mathsf{X}^i(\psi_1 \,\mathcal{R}\, \psi_2)$:

```
next($error_{\mathsf{X}^i(\psi_1\mathcal{R}\psi_2)}$) := case
    $counter < i$ : FALSE;
    $\neg error_{\mathsf{X}^i(\psi_1\mathcal{R}\psi_2)} \wedge v^i_{\psi_1^p}$ : FALSE;
    $\neg error_{\mathsf{X}^i(\psi_1\mathcal{R}\psi_2)} \wedge v_{\psi_1} \wedge v_{\psi_2}$ : FALSE;
    $\neg error_{\mathsf{X}^i(\psi_1\mathcal{R}\psi_2)} \wedge v_{\psi_2}$ : FALSE;
    TRUE : TRUE;
  esac

next($v^i_{\psi_1^p}$) := case
    $counter < i$ : FALSE;
    $v_{\psi_1^p}$ : TRUE;
    $v^i_{\psi_1^p}$ : TRUE;
    TRUE : FALSE;
  esac
```

In Fig. 2, we describe the execution of all the steps described so far on a simple formula.



$\mathsf{G}(u_1 \to \mathsf{XX}c_1)$
$\wedge$
$\mathsf{G}(u_2 \to \mathsf{X}c_2)$
pastify
$\mathsf{GXX}(\mathsf{YY}u_1 \to c_1)$
$\wedge$
$\mathsf{GX}(\mathsf{Y}u_2 \to c_2)$
canonize
$\mathsf{XXG}(\mathsf{YY}u_1 \to c_1)$
$\wedge$
$\mathsf{XG}(\mathsf{Y}u_2 \to c_2)$
to SSA
ASSIGN
init($error_1$) := $\bot$
next($error_1$) := ...
ASSIGN
init($error_2$) := $\bot$
next($error_2$) := ...
INVARSPEC
$\neg error_1 \wedge \neg error_2$

Figure 2. The execution of the sequence of steps: a simple example.

Proposition 5: Let ϕ be a canonical $\mathsf{PastLTL_{EBR}}$ formula, with $|\phi| = n$. Then, there exists a deterministic SSA of size $\mathcal{O}(n)$ that accepts the same language.

Theorem 1: Let ϕ be an $\mathsf{LTL_{EBR}}$ formula, with $|\phi| = n$, and let M be the greatest constant in ϕ. Then, there exists a deterministic SSA of size $\mathcal{O}(n^3 \cdot M^{\log_2 n+1})$ that accepts the same language.

Corollary 1: Let ϕ be an $\mathsf{LTL_{EBR}}$ formula with no constants, with $|\phi| = n$. Then, there exists a deterministic SSA of size $\mathcal{O}(n^3)$ that accepts the same language.

V. Solving the Game on the Symbolic Deterministic Automaton

Once we have obtained the deterministic SSA $\mathcal{A}(\phi)$ for an $\mathsf{LTL_{EBR}}$ formula ϕ with the steps described in the previous sections, we can use $\mathcal{A}(\phi)$ as the arena of a two-player game between Controller and Environment in order to solve the realizability (and synthesis) problem for ϕ.

Let us focus on the *safety game* $G = \langle \mathcal{A}(\phi), \mathcal{C}, \mathcal{U} \rangle$ (recall Def. 6). Safety games have been extensively studied, as their reachability objective makes the problem simpler than considering ω-regular objectives, such as, for instance, Büchi and Rabin conditions.

The aim of Controller is to choose an infinite sequence of *controllable* variables in such a way that, no matter what values for the *uncontrollable* variables are chosen by Environment, *the* trace induced by the play in $\mathcal{A}(\phi)$ is *safe*, that is, it visits only states s such that $s \models S(X)$ (see Def. 6). Since in our case

$\mathcal{A}(\phi)$ recognizes exactly the language of ϕ, the play satisfies ϕ, and thus Controller has a winning strategy for ϕ.

Since the organization of the SYNTCOMP [14], many optimized tools have been proposed in the literature to solve safety games. For this reason, we chose to use a safety synthesizer as a black box. The majority of these tools accept as input a symbolic arena described in terms of and-inverter graphs (or AIGER format [1]), so we provide a simple utility to obtain the AIGER representation of *functional* SMV modules, that is, SMV modules with the transition relation expressed only in terms of ASSIGN statements, such as the ones resulting from our encoding. The AIGER model is then given as input to the chosen safety synthesizer, completing the process outlined in Fig. 1.

The next theorem states the complexity of the procedure.

Theorem 2: The realizability problem for $\mathsf{LTL}_{\mathsf{EBR}}$ belongs to 2EXPTIME. If no constant is admitted, it belongs to EXPTIME.

Proof: We first show that the proposed algorithm, as described in Fig. 1, belongs to 2EXPTIME for generic $\mathsf{LTL}_{\mathsf{EBR}}$ formulas. It is easy to see that the time complexity of all the steps matches their space complexity. Therefore, we have an algorithm to turn an $\mathsf{LTL}_{\mathsf{EBR}}$ formula ϕ into an equivalent deterministic SSA $\mathcal{A}(\phi)$ whose time complexity is $\mathcal{O}(n^3 \cdot M^{\log_2 n+1})$, where $n = |\phi|$ and M is the greatest constant in ϕ. Since $\mathcal{A}(\phi)$ is symbolically represented, it can be turned into an explicit automaton $\mathcal{A}'(\phi)$ of size at most exponential in the size of $\mathcal{A}(\phi)$, that is, $|\mathcal{A}'(\phi)| \in \mathcal{O}(2^{n^3 \cdot M^{\log_2 n+1}})$. Finally, the time complexity of reachability games is *linear* in the size of the arena [9], and thus the overall time complexity of the realizability problem for $\mathsf{LTL}_{\mathsf{EBR}}$ is 2EXPTIME. If no constant is admitted, then, by Corollary 1, $|\mathcal{A}'(\phi)| \in \mathcal{O}(2^{n^3})$, and the complexity becomes EXPTIME. ■

Comparison with Safety LTL

It is interesting to briefly compare the proposed procedure for realizability to the one used by the Ssyft tool for Safety LTL specifications [25]. In that tool, the negation of the initial formula is first translated into first-order logic over finite words and then transformed into deterministic automata using the tool MONA [12], which uses the classical subset construction to determinize automata over finite words. Finally, Ssyft uses the classical backward fixpoint iteration to compute the set of winning states over the DFA. It is worth to notice that the way MONA represents automata is *not* fully symbolic: the set of states is explicitly represented, while it uses a BDD for each pair of states in order to represent symbolically the transitions between the two corresponding states. In contrast of subset construction, our solution performs the pastification of full-bounded formulas. Most importantly, our construction of deterministic monitors is carried out in a fully symbolic way.

VI. Experimental Evaluation

We implemented the proposed procedure (see Fig. 1) in a tool called ebr-ltl-synth.[1] The transformation from $\mathsf{LTL}_{\mathsf{EBR}}$ to deterministic SSA together with the translation to AIGER has been implemented inside the nuXmv model checker [6]. As the backend for solving the safety game, we have chosen the SAT-based tool demiurge [2].

We tested our tool on a set of scalable benchmarks divided in four categories (the propositional atoms starting with the letter *c* are controllable, while those starting with the letter *u* are uncontrollable):

1) the first category is generated by the realizable formula:

$$\mathsf{G}(c_0 \wedge \mathsf{X}\mathsf{G}(c_1 \wedge \dots \wedge \mathsf{X}^n\mathsf{G}(c_n \vee u)\dots))$$

2) the second category is generated by the realizable formula:

$$\mathsf{G}((c_0 \vee u_0) \wedge \mathsf{X}\mathsf{G}((c_1 \vee u_1) \wedge \dots \wedge \mathsf{X}^n\mathsf{G}((c_n \vee u_n))\dots))$$

3) the third category is generated by the unrealizable formula:

$$\mathsf{G}(c) \wedge \bigvee_{i=1}^{n} \mathsf{G}(\bigwedge_{j=0}^{i} u_i)$$

4) the fourth category is generated by the unrealizable formula:

$$c \wedge \bigwedge_{i=1}^{n} \mathsf{X}^i(u_i \vee u_{i+1})$$

Each category contains the respective scalable formula for $n \in [1, 200]$, for a total of 800 benchmarks, half of which is realizable and the other half is unrealizable. We set a timeout of 180 seconds for each benchmark. We compared ebr-ltl-synth with ltlsynt [13], Strix [19] and Ssyft [25]. The first two tools solve the realizability and synthesis problems for full LTL and are based on a translation to parity games. ltlsynt uses SPOT [10] for efficient translation and manipulation of automata. Strix implements several optimizations like specification splitting, that enables to split the initial formula in safety, co-safety, Büchi, and co-Büchi subformulas and speeds up the process of solving of the game. On the contrary, Ssyft solves the realizability problem for specifications written in Safety LTL (see Sec. V for a brief description of the Ssyft tool).

For realizability, we tested all the tools in their sequential configurations. ltlsynt has two sequential configurations, which differ on whether the split of actions into Controller's and Environment's ones is performed before or after the determinization. Strix has two sequential modes as well, depending on the kind of search on the arena (depth-first for the first configuration and with a priority queue for the second). Ssyft and ebr-ltl-synth have only one configuration.

Fig. 3 shows the outcomes of the comparison between ebr-ltl-synth and the best configuration of ltlsynt: it can be clearly seen that, for both realizable and unrealizable formulas, ltlsynt presents an exponential blow-up in the solving time that is avoided by ebr-ltl-synth. Fig. 4 compares ebr-ltl-synth with the best configuration of Strix: while for realizable formulas there is an exponential blow up of Strix avoided by ebr-ltl-synth, it is interesting to note that for the unrealizable benchmarks the difference between the solving time of the two tools is linear, mostly showing a 10x improvement in favor of ebr-ltl-synth.

[1]http://users.dimi.uniud.it/~luca.geatti/tools/ebrltlsynth.html

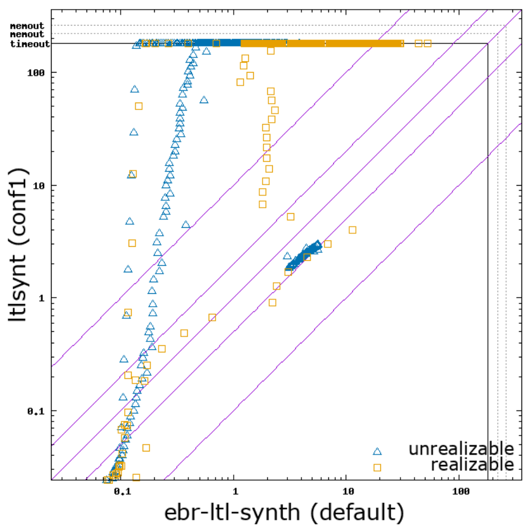


Figure 3. ebr-ltl-synth vs ltlsynt (first conf.) on all scalable benchmarks.

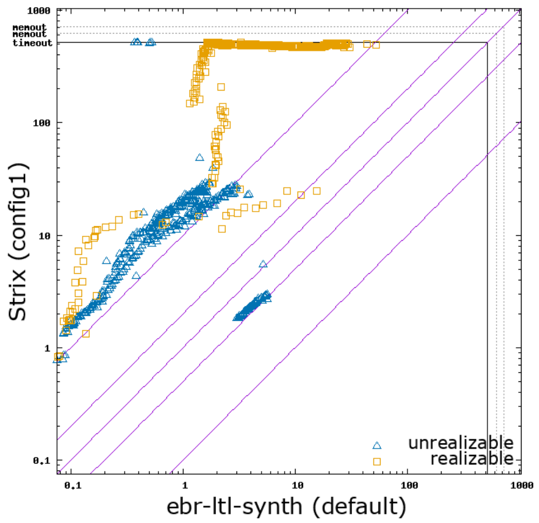


Figure 4. ebr-ltl-synth vs Strix on all scalable benchmarks.

The survival plots for the set of realizable and unrealizable scalable benchmarks are shown in Figs. 5 and 6, respectively.

The outcomes of the comparison between ebr-ltl-synth and Ssyft are shown in Fig. 7. The three lines near the sides of the figure correspond to *timeouts* (the solid black line), *memouts* for unrealizable benchmarks and *memouts* for realizable benchmarks (the dotted lines). It can be noticed that Ssyft reaches a memory out for the vast majority of benchmarks. For instance, on both the realizable categories, Ssyft reaches the first memout with $n = 7$. As for the unrealizable benchmarks, on the third category, Ssyft reaches the first memout with $n = 36$, while for the fourth category with $n = 59$. This is due to MONA, which is not able to build the (explicit) DFA for the

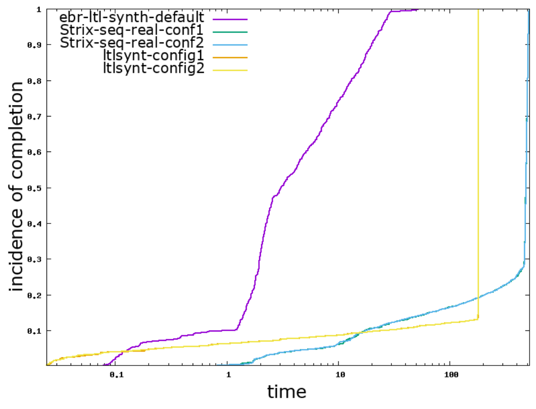


Figure 5. Survival plot for realizable scalable benchmarks.

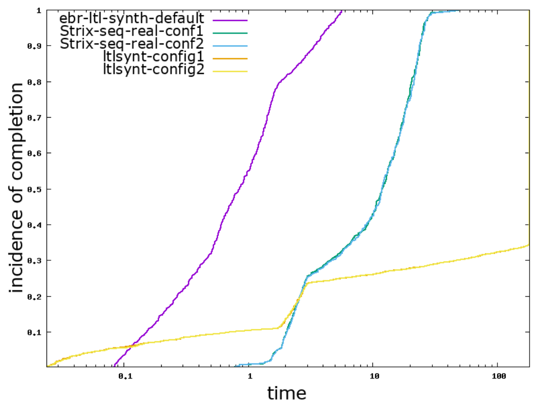


Figure 6. Survival plot for unrealizable scalable benchmarks.

(negation of the) initial specification[2]. This is an important hint about the use of *fully symbolic* techniques for the representation of automata, like the one of ebr-ltl-synth, as in many cases they can avoid an exponential blowup of the automata' state space. The survival plot between ebr-ltl-synth and Ssyft is shown in Fig. 8[3]. The rest of the plots for realizability of scalable benchmarks can be found in [8].

In addition to these scalable formulas, from the benchmarks of SYNTCOMP [14], we filtered the formulas that belong to $\mathsf{LTL}_{\mathsf{EBR}}$: this resulted into a set of 29 formulas. The survival plot showing the comparison with ltlsynt and Strix is shown in Fig. 9, while the comparison with Ssyft is shown in Fig. 10. It is interesting to see that, on the SYNTCOMP benchmarks, the results of ebr-ltl-synth and Ssyft are comparable.

As for the synthesis problem, once a specification is found to be realizable, all the tools except for Ssyft produce a strategy as a witness: this strategy is in the form of an and-inverter graph whose input bits are only the starting uncontrollable variables. Often, such a strategy can be minimized by using logic

[2]We point out that in some cases, like in the fourth category for $n \geq 60$, MONA's memouts are due to its parser.

[3]The reason why we do not have a single survival plot comparing all the four tools is that Ssyft could not have been compiled for the same platform as the others, due to issues with its source code.

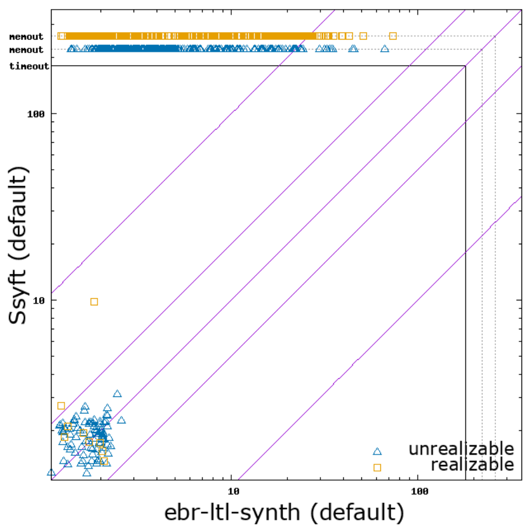


Figure 7. ebr-ltl-synth vs Ssyft on scalable benchmarks.

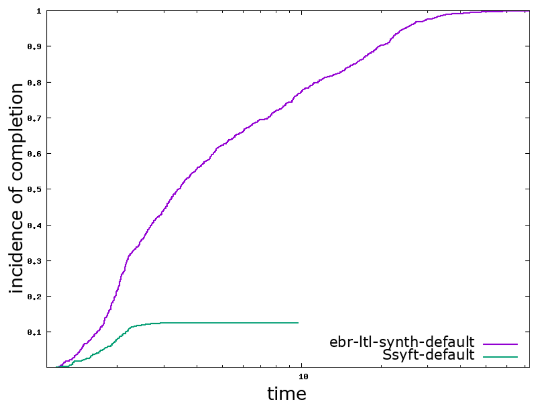


Figure 8. Survival plot for ebr-ltl-synth and Ssyft on scalable benchmarks.

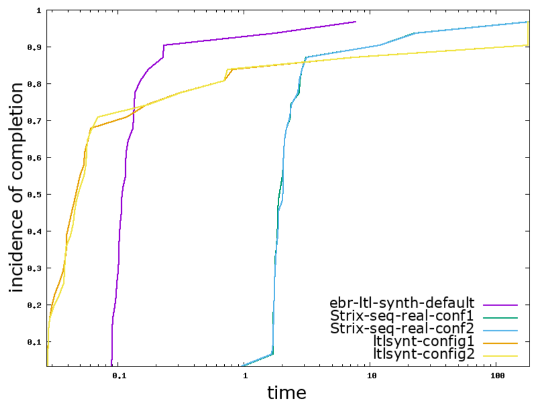


Figure 9. Survival plot for SYNTCOMP benchmarks.

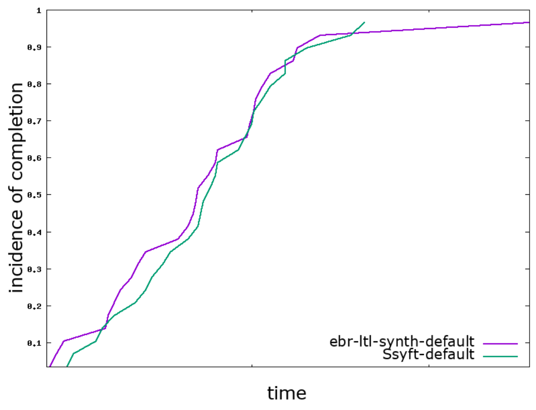


Figure 10. Survival plot for ebr-ltl-synth and Ssyft on SYNTCOMP benchmarks.

synthesis tools (like ABC [3]) as black-box. In the particular case of ebr-ltl-synth, ltlsynt and Strix, they all use a separate logic synthesizer as black box, with different configurations to minimize the strategy. Therefore, we do not compare the size of the resulting strategies, since such a comparison would add nothing about the methods implemented by the tools but would rather compare their backends.

VII. Conclusions

In this paper, we introduce the logic $\mathsf{LTL}_{\mathsf{EBR}}$, a fragment of LTL that combines formulas with only bounded operators and a particular combination of universal unbounded temporal operators. We focus on the realizability and reactive synthesis problems for this logic. The main contribution is a *fully symbolic* translation from any $\mathsf{LTL}_{\mathsf{EBR}}$ formula to a *deterministic* symbolic safety automaton on infinite words. The process applies a pastification step and a set of rules to reach a canonical form for $\mathsf{LTL}_{\mathsf{EBR}}$ formulas. The realizability is then decided by solving a safety game on the arena represented by the automaton. We first showed that realizability for $\mathsf{LTL}_{\mathsf{EBR}}$ belongs to 2EXPTIME, but drops to EXPTIME if no constants are used. Then, we implemented the proposed procedure in a tool, whose experimental evaluation revealed very good performance against tools for realizability and synthesis of full LTL and Safety LTL specifications.

As a future development of this line of work, we believe that the translation from $\mathsf{LTL}_{\mathsf{EBR}}$ to deterministic SSA may provide many benefits in the context of *symbolic model checking* as well, since the search of the state space could benefit from a deterministic representation of the automaton for the formula [23]. On the automata construction side, an interesting development would be to keep the symbolic bounds during pastification and monitor construction, without, for instance, expanding $\mathsf{X}^i\alpha$ into i nested *next* operators. On the expressiveness side, we want to study in which ways *assumptions* can be integrated into $\mathsf{LTL}_{\mathsf{EBR}}$. Last but not least, we aim at checking whether the synthesis problem for more expressive logics, like, for instance, LTL, can be reduced to the synthesis problem for $\mathsf{LTL}_{\mathsf{EBR}}$, for example checking whether it is possible to use $\mathsf{LTL}_{\mathsf{EBR}}$ for solving the safety problems originated from *bounded synthesis* techniques.

Acknowledgments: The authors want to thank all the anonymous reviewers of FMCAD 2020 for the insightful comments on a preliminary version of this paper.

References

[1] Biere, A., Heljanko, K., Wieringa, S.: Aiger 1.9 and beyond. Available at fmv. jku. at/hwmcc11/beyond1. pdf (2011)

[2] Bloem, R., Könighofer, R., Seidl, M.: Sat-based synthesis methods for safety specs. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 1–20. Springer (2014)

[3] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: International Conference on Computer Aided Verification. pp. 24–40. Springer (2010)

[4] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: The collected works of J. Richard Büchi, pp. 425–435. Springer (1990)

[5] Buchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. In: The Collected Works of J. Richard Büchi, pp. 525–541. Springer (1990)

[6] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: International Conference on Computer Aided Verification. pp. 334–342. Springer (2014)

[7] Church, A.: Logic, arithmetic, and automata. In: Proceedings of the international congress of mathematicians. vol. 1962, pp. 23–35 (1962)

[8] Cimatti, A., Geatti, L., Gigante, N., Montanari, A., Tonetta, S.: Reactive synthesis from extended bounded response LTL specifications (2020), https://arxiv.org/abs/2008.05335, Extended Version on *arXiv*

[9] De Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theoretical Computer Science **386**(3), 188–217 (2007)

[10] Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0—a framework for LTL and ω-automata manipulation. In: International Symposium on Automated Technology for Verification and Analysis. pp. 122–129. Springer (2016)

[11] Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer **15**(5-6), 519–539 (2013)

[12] Henriksen, J.G., Jensen, J., Jørgensen, M., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: Mona: Monadic second-order logic in practice. In: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems. pp. 89–110. Springer (1995)

[13] Jacobs, S., Bloem, R.: The 5th reactive synthesis competition-syntcomp 2018

[14] Jacobs, S., Bloem, R., Brenguier, R., Ehlers, R., Hell, T., Könighofer, R., Pérez, G.A., Raskin, J.F., Ryzhyk, L., Sankur, O., et al.: The first reactive synthesis competition (syntcomp 2014). International journal on software tools for technology transfer **19**(3), 367–390 (2017)

[15] Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05). pp. 531–540. IEEE (2005)

[16] Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009)

[17] Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: International Conference on Formal Modeling and Analysis of Timed Systems. pp. 2–16. Springer (2005)

[18] Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: International Conference on Computer Aided Verification. pp. 95–107. Springer (2007)

[19] Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: International Conference on Computer Aided Verification. pp. 578–586. Springer (2018)

[20] Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 364–380. Springer (2006)

[21] Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: International Colloquium on Automata, Languages, and Programming. pp. 652–671. Springer (1989)

[22] Rosner, R.: Modular synthesis of reactive systems. Ph.D. thesis, PhD thesis, Weizmann Institute of Science (1992)

[23] Sebastiani, R., Tonetta, S.: "More Deterministic" vs. "Smaller" Büchi Automata for Efficient LTL Model Checking. In: Geist, D., Tronci, E. (eds.) CHARME. Lecture Notes in Computer Science, vol. 2860, pp. 126–140. Springer (2003). https://doi.org/10.1007/978-3-540-39724-3_12, https://doi.org/10.1007/978-3-540-39724-3_12

[24] Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and computation **115**(1), 1–37 (1994)

[25] Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A symbolic approach to safety LTL synthesis. In: Haifa Verification Conference. pp. 147–162. Springer (2017)