Counter Example Guided Reactive Synthesis for LTL Modulo Theories*

Andoni Rodríguez[1,2], Felipe Gorostiaga[1] and César Sánchez[1]

[1] IMDEA Software Institute, Madrid, Spain
[2] Universidad Politécnica de Madrid, Spain

Abstract. Reactive synthesis is the process of automatically generating a correct system from a given temporal specification. In this paper, we address the problem of reactive synthesis for LTL modulo theories ($\text{LTL}^{\mathcal{T}}$), which extends LTL with literals from a first-order theory *and allows relating the values of data across time*. This logic allows describing complex dynamics both for the system and for the environment—such as a numeric variable increasing monotonically over time. The logic also allows defining relations (and not only assignment) between variables, enabling permissive shielding.

We propose a sound algorithm called Counter-Example Guided Reactive Synthesis modulo theories (CEGRES), whose core is the novel concept of *reactive tautology*, which are valid temporal formulas that preserve the semantics of the specification but make the algorithm conclusive. Although realizability for full $\text{LTL}^{\mathcal{T}}$ is undecidable in general, we prove that CEGRES is terminating for some important theories and for arbitrary theories when specifications do not fetch data across time. We include an empirical evaluation that shows that CEGRES can solve many reactive synthesis problems of practical interest.

1 Introduction

Reactive synthesis [34,33] is the problem of automatically producing a correct controller given a temporal specification, whose Boolean variables (i.e., atomic propositions) are split into those controlled by the environment and those controlled by the system. Realizability is the decision problem of deciding whether such a system exists. Reactive synthesis corresponds to a two-player game between a system and an environment. A specification is realizable if the system player has a winning strategy. Realizability has been widely studied for LTL [32].

At the same time, there is a growing interest in LTL modulo theories ($\text{LTL}^{\mathcal{T}}$), which allows reasoning about values in the domains of first-order theories. For example, [10,43,16] study satisfiability for variants of LTL with data. Also, [15] considers a variant of $\text{LTL}^{\mathcal{T}}$ for finite traces which allows retrieving values of variables at the previous or following instants. However, reactive synthesis is

* This work was funded in part by the DECO Project (PID2022-138072OB-I00) funded by MCIN/AEI/10.13039/501100011033 and by the ESF+.

not studied in [15,17]. Later, [37] showed the decidability of realizability for a fragment of $\text{LTL}^{\mathcal{T}}$ (which we call $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ here) without lookup, later extended in [38,36] into a full synthesis procedure. The method in [37] takes an $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ specification φ, and generates an equi-realizable Boolean LTL abstraction $\varphi^{\mathbb{B}}$. The practical application is however limited because (1) the method exhaustively explores all possible system reactions, which compromises scalability; and (2) the logic restricts the transfer of data values across time. In [37] the controller can remember the valuation of a literal, (e.g. whether x was odd in the previous timestep), but cannot remember the concrete value of x. Other works consider temporal logics that allow numeric information to flow across time, but with different expressive limitations like the temporal fragment [14,21,26] or the lack of operators to compare values at different time instants [7,25,18,42,40,41,26,20].

In this paper we study reactive synthesis for full $\text{LTL}^{\mathcal{T}}$, which essentially consists of LTL temporal operators, literals from the theory $\mathcal{T}$ and the lookup operator $\lhd$ (also called *fetch*). This additional expressivity enables more realistic scenarios, including for example complex dynamics like $\square(|\lhd v - v| \leq 30)$, which means that a current velocity v cannot change in more than 30 units from timestep to timestep. We introduce a novel method called Counter-Example Guided Reactive Synthesis (CEGRES) for temporal logics modulo theory, which is a general algorithm based on refinements (in a CEGAR [8] fashion). CEGRES attempts an abstraction to a simpler reactive synthesis problem, studies the strategy proposed as solution and refines the abstraction by learning from illegal actions of the strategy (whose legality was not preserved in the abstraction). In summary, the contributions of this paper are the following:

(1) A general CEGRES method for reactive synthesis of temporal logics with data.
(2) An instance of the general method for $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ which significantly improves scalability for this decidable fragment. This is used as a building block in the following steps.
(3) An instance of CEGRES for full $\text{LTL}^{\mathcal{T}}$, which is a sound synthesis procedure (yet possibly not terminating because $\text{LTL}^{\mathcal{T}}$ realizability is in general undecidable). This method introduces the novel concept of *reactive tautologies*, which are valid temporal formulas that improve the equi-realizability between the original $\text{LTL}^{\mathcal{T}}$ specification and its abstraction. These tautologies are to realizability modulo theories what inductive invariants are to invariant proving. The refinement phase of CEGRES for general $\text{LTL}^{\mathcal{T}}$ is guaranteed to progress by generating these tautologies.
(4) We show that our method always terminates for theories with bounded domains (very common in embedded systems) and avoids blasting the variables into a Boolean specification.
(5) We showcase the applicability of CEGRES with a large set of benchmarks.

We emphasize our synthesis problem is different, because (in contrast with TSL-MT) it allows choosing different values at different time-steps. To the best of our knowledge, this is the first method to solve this problem.

Related work. Reactive synthesis has been shown to be decidable for $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ (see [37,38,36,39]), but our CEGRES method for $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ is faster because it avoids an upfront exhaustive exploration of the set of reactions. However, $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ does not allow data temporal relations which limits its application in some realistic domains. A more expressive variant of $\text{LTL}^{\mathcal{T}}$ was presented in [15,17] for finite traces, but they address the satisfiability problem and not reactive synthesis.

A close specification language to $\text{LTL}^{\mathcal{T}}$ is Temporal Stream Logic Modulo Theories (TSL-MT), which was first introduced for satisfiability [13,12]. Reactive synthesis for TSL-MT specifications is addressed in [25] using abstraction-refinement, which is often not tractable for the presented benchmarks. The work in [7] presents an abstraction-refinement method introducing *data transformation obligations* (similar in spirit to our reactive tautologies), but obligations are in a `Precondition|State|Postcondition` style that is less general than our reactive tautologies. For instance, their obligation $\square[(a = 0 \wedge \bigcirc(a = \lhd a + 1) \wedge \bigcirc^2(a = \lhd a + 1) \rightarrow \bigcirc^2(a = 2)]$ does not capture enough information about what happens when a increases over three timesteps, or what if addition does not happen two timesteps in a row. In contrast, our tautologies capture a more general relation across time.

The work in [20] provides rules and an invariant notion similar to our tautologies (and to [7,25]), for RP-LTL. RP-LTL is similar to $\text{LTL}^{\mathcal{T}}$ but does not allow to directly refer to previous environment variables and the environment controls all variables in the first instant, making the move of the system in the first timestep irrelevant. Also, [20] is based on symbolic game solving and the implementation and empirical evaluation in [20] is still restricted to TSL-MT. TSL-MT synthesis is different from $\text{LTL}^{\mathcal{T}}$ synthesis because the semantics of TSL-MT only allow to fetch past information via an *update* operation; e.g., $(x = \lhd x)$ is legal in TSL-MT, but $(x < \lhd x)$ or $(x < \lhd x + z)$ are not. This limits the ability to express arbitrary relations in terms of data across time, and precludes the application to the field of shielding modulo theories [35] as post-shields, because a shield that uses equality would force a single fixed move. The reason is that TSL-MT follows a clear design goal: the restriction to updates with equality and restricting the specification to a finite number of possible assignments facilitates extracting controllers as programs. In contrast, $\text{LTL}^{\mathcal{T}}$ is a specification language with more non-determinism.

Another important recent approach is [18,42], which use acceleration-based game solving for expressive reactive synthesis problems. However, the specification allows infinite data domains only in the environment inputs, and not in the system outputs. Also, [18,42] does not reduce to simpler synthesis problems. Instead, they create directly 2-player games and then search strategies in the corresponding arenas. Similarly, [40,41] proposes a fixpoint-based solver for infinite-state safety game-arenas. In addition, [14,21,26] also perform synthesis of modulo theories specifications, but their techniques only apply to safety, assume-guarantee and GR(1) fragments, respectively. In contrast, our method works for any temporal fragment (as long as the underlying solver can handle the fragment).

2 Preliminaries

Let AP be a set of *atomic propositions* and $\Sigma = 2^{AP}$ be the propositional *alphabet*, where we call each element of Σ a *letter*. A *trace* is an infinite sequence $\sigma = a_0 a_1 \cdots$ of letters from Σ, where we use $\sigma(i)$ for a_i. We denote the set of all infinite traces by Σ^ω. A *pointed trace* is a pair (σ, p), where $p \in \mathbb{N}$ is a natural number (called the *pointer*).

LTL and Reactive Synthesis. Linear Temporal Logic (LTL) [32,28] extends propositional logic with time modalities. LTL has the following syntax:

$$\varphi ::= \top \mid a \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi,$$

where a ranges from an atomic set of propositions AP, $\top$ is the true value, $\vee$, $\wedge$ and $\neg$ are the usual Boolean operators, and $\bigcirc$ and $\mathcal{U}$ are the *next* and *until* temporal operators. The semantics of LTL associates traces $\sigma \in \Sigma^\omega$ (where $\Sigma = 2^{AP}$) with formulas as follows (we omit Boolean operators which are standard):

$$\begin{array}{ll}
(\sigma, i) \models a & \text{iff } a \in \sigma(i) \\
(\sigma, i) \models \bigcirc\varphi & \text{iff } (\sigma, i+1) \models \varphi \\
(\sigma, i) \models \varphi \,\mathcal{U}\, \psi & \text{iff for some } j \geq i,\ (\sigma, j) \models \psi \text{ and for all } i \leq k < j, (\sigma, k) \models \varphi
\end{array}$$

We use $\sigma \models \varphi$ for $(\sigma, 0) \models \varphi$.

Reactive synthesis [34,33,11] is the problem of automatically constructing a reactive system from an LTL specification φ. The atomic propositions AP of φ are partitioned into propositions $\overline{e} = \mathit{Vars}_E(\varphi)$ controlled by the environment and $\overline{s} = \mathit{Vars}_S(\varphi)$ controlled by the system (with $\overline{e} \cup \overline{s} = AP$ and $\overline{e} \cap \overline{s} = \emptyset$). Reactive Synthesis corresponds to a turn-based game in an arena where the environment and system players alternate. In each turn, the environment produces values for $\overline{e}$, and the system responds with values for $\overline{s}$. We use $\mathit{val}(\overline{e})$ and $\mathit{val}(\overline{s})$ for valuations. A play is an infinite sequence of turns, which induces a trace σ by joining at each position the valuations that the environment and system players choose. The system player wins a play σ whenever $\sigma \models \varphi$.

We introduce now a slightly non-standard notion of strategy as a Mealy machine, which is a common output from synthesis tools like Strix [30]. This definition is convenient in this paper because it is common for both players—environment and system—and eases the description of the algorithms in Sections 4 and 5. A strategy is a tuple $M : \langle Q, q_0, T \rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state and $T \subseteq Q \times Q \times \mathit{val}(\overline{e}) \times \mathit{val}(\overline{s})$. Given $t = (q, q', e, s)$ we use $t.\mathit{from}$ for the pre-state q, $t.\mathit{to}$ for the post-state q', $t.\mathit{env}$ for the valuation e of environment variables and $t.\mathit{sys}$ for the valuation s of system variables. We use T_q for $\{t \mid t.\mathit{from} = q\}$. We say that a strategy M is *legal for the system* if, for all states q, all environment valuations are considered in some transition, that is for every q and every $\overline{a} \in \mathit{val}(\overline{e})$ there is a $t \in T_q$ with $t.\mathit{env} = \overline{a}$. Similarly, a strategy M is *legal for the environment* if for all states q, all sets of outgoing transitions that agree on the environment move, cover all

possible moves of the system. Formally, for every state q (1) there is a $\overline{a} \in \mathit{val}(\overline{e})$ and a transition $t \in T_q$ with $t.\mathit{env} = \overline{a}$; and (2) for every $\overline{a} \in \mathit{val}(\overline{e})$ such that there is a $t \in T_q$ with $t.\mathit{env} = \overline{a}$, then for every $\overline{c}$ in $\mathit{val}(\overline{s})$ there is a $t' \in T_q$ with $t'.\mathit{env} = \overline{a}$ and $t'.\mathit{sys} = \overline{c}$. Of course, in practice T can be represented explicitly, or symbolically as formulas covering many valuations.

A trace $((\overline{a}_0, \overline{c}_0), (\overline{a}_1, \overline{c}_1), \ldots)$ is played according to a strategy M if there is a sequence $(q_0, q_1, \ldots)$ such that $(q_i, q_{i+1}, \overline{a}_i, \overline{c}_i) \in T$ for all $i \geq 0$. We use $\mathit{Traces}(M)$ for the set of traces played according to M. A strategy M is winning for the system if M is legal for the system and $\mathit{Traces}(M) \models \varphi$. A strategy M is winning for the environment if M is legal for the environment and $\mathit{Traces}(M) \models \neg\varphi$. If there is a winning strategy for the system, then φ is *realizable*.

Theories. A first-order theory $\mathcal{T}$ (see e.g. [4]) is described by a signature that consists of a finite set of functions and predicates, a set of variables, and a domain, which is the sort of its variables. For simplicity in the presentation we assume single-sorted theories but all our results can be extended to multi-sorted theories. For example, the domain of linear integer arithmetic $\mathcal{T}_{\mathbb{Z}}$ is $\mathbb{Z}$ and we denote this by $\mathbb{D}(\mathcal{T}_{\mathbb{Z}}) = \mathbb{Z}$. A predicate α in a theory $\mathcal{T}$ is valid iff $I \models \alpha$ for every interpretation I of $\mathcal{T}$. A literal is a predicate or its negation. We use $\mathit{Terms}_{\mathcal{T}}(V)$ for the terms that can be built from variables in V using the constructors of $\mathcal{T}$. Similarly, $\mathit{Lits}_{\mathcal{T}}(V)$ is the set of literals. We drop $\mathcal{T}$ from $\mathit{Lits}_{\mathcal{T}}$ and $\mathit{Terms}_{\mathcal{T}}$ when clear from the context. We use $\mathit{Vars}(l)$ for the set of variables in the literal l and $\mathit{Vars}(\varphi)$ for the union of the variables that occur in the literals of the formula φ. A valuation for a set of variables $\overline{z}$ is a map from $\overline{z}$ into $\mathbb{D}(\mathcal{T})$. We assume that every theory assigns a meaning to each function f and predicate symbol l such that $[\![f]\!]$ gives a value of the domain and $[\![l]\!]$ gives a truth value given values of the arguments. We also assume a default value d in $\mathbb{D}(\mathcal{T})$.

3 LTL Modulo Theories

Syntax and Semantics. LTL modulo theories extends LTL using first-order theories. Given a set of $\mathcal{T}$ variables V the *alphabet* is a map $\Sigma_{\mathcal{T}} : V \to \mathbb{D}(\mathcal{T})$. Given an alphabet a over a set of vars V and a term t with $\mathit{Vars}(t) \subseteq V$, we use $t(a)$ as the value obtained by substiting every variable in t according to a and evaluating $[\![t]\!]$. Similarly we use $l(a)$ for a predicate l.

LTL modulo Theory ($\mathrm{LTL}^{\mathcal{T}}$) extends LTL in the following way. First, the atomic predicates AP are substituted with literals from the theory. Second, $\mathrm{LTL}^{\mathcal{T}}$ introduces two new term constructors $\lhd x$ and $\rhd x$ (that apply to variables) with the intended meaning of retrieving the value of variable x. We use *fetch* (or look back) to refer to the $\lhd$ operator and *look ahead* for $\rhd$. Then, for a set of variables V, we use $F(V)$ for the set of terms $V \cup \{\lhd x | x \in V\} \cup \{\rhd x | x \in V\}$ and we extend the set of terms to $\mathit{Terms}(F(V))$ and the set of literals to $\mathit{Lits}(F(V))$. Then, the syntax of an $\mathrm{LTL}^{\mathcal{T}}$ formula over variables V is

$$\varphi ::= T \mid l \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi,$$

where $l \in \mathit{Lits}(F(V))$. Given a trace $\sigma \in \Sigma_{\mathcal{T}}(V)^\omega$ we define the valuation of a variable as $[\![x]\!]_{(\sigma,i)} \stackrel{\text{def}}{=} \sigma(i)(x)$, and the valuation $[\![t]\!]_{(\sigma,i)}$ of a term $t \in \mathit{Terms}(F(V))$ (and of a literals $l \in \mathit{Lits}_{\mathcal{T}}(F(V))$) at position i as recursively:

$$\begin{aligned}
[\![\lhd x]\!]_{(\sigma,i)} &\stackrel{\text{def}}{=} \sigma(i-1)(x) \text{ (or } d \text{ if } i = 0) \\
[\![\rhd x]\!]_{(\sigma,i)} &\stackrel{\text{def}}{=} \sigma(i+1)(x) \\
[\![t(s_1,\ldots,s_n)]\!]_{(\sigma,i)} &\stackrel{\text{def}}{=} [\![t]\!]([\![s_1]\!]_{(\sigma,i)},\ldots,[\![s_1]\!]_{(\sigma,i)}) \\
[\![l(s_1,\ldots,s_n)]\!]_{(\sigma,i)} &\stackrel{\text{def}}{=} [\![l]\!]([\![s_1]\!]_{(\sigma,i)},\ldots,[\![s_1]\!]_{(\sigma,i)})
\end{aligned}$$

Then, the semantics of $\mathrm{LTL}^{\mathcal{T}}$ associates traces $\sigma \in \Sigma_{\mathcal{T}}(\mathit{Vars}(\varphi))^\omega$ with formulas φ as in LTL but considering the base case: $(\sigma,i) \models l$ whenever $[\![l]\!]_{(\sigma,i)}$ holds. The logic defined here is essentially equivalent (but for infinite traces) to LTL modulo theories in [15]. The previous syntax can be extended to allow nested $\lhd$ and $\rhd$ operators, which can be eliminated linearly into equivalent formulae.

Eliminating $\lhd\lhd x$. Consider an arbitrary formula φ and a term $\lhd \lhd x$ in φ. Assume that y is assigned to the system. Let φ' be the formula $(\bigcirc\Box(y_{new} = \lhd y) \wedge \varphi[\lhd y \leftarrow y_{new}])$. Let σ be a trace for φ and let us extend σ into σ' by adding the assignment to the new varaible y_{new}, $\sigma'(i+1)(y_{new}) = \sigma(i)(y)$ and $\sigma'(0)(y_{new}) = d$. It is easy to see that $\sigma' \models \bigcirc\Box(y_{new} = \lhd y)$, that $\sigma' \models \varphi$ if and only if $\sigma \models \varphi$, and that $\sigma' \models \varphi$ if and only if $\sigma' \models \varphi'$. The same argument can be followed for a variable x assigned to the environment creating φ' as $(\bigcirc\Box(x_{new} = \lhd x) \rightarrow \varphi[\lhd x \leftarrow x_{new}])$.

Similarly, $\rhd$ can also be eliminated by adding a $\bigcirc$ operator and shifting the variables in the formula introducing $\lhd$ if necessary.

Eliminating $\rhd$. Consider a fomula φ that contains a literal l with a term t that has $\rhd x$ as a sub-term. We create a literal l' from l by replacing every term $\rhd y$ with x, every term $\lhd y$ with $\lhd\lhd y$, and every variable z not preceded with $\rhd$ or $\lhd$ with $\lhd z$. It is easy to see following the definitions of semantics and valuations that for every (σ,i) a $[\![l]\!]_{(\sigma,i)} = [\![l']\!]_{(\sigma,i+1)}$. Then, we replace l with $\bigcirc l'$ in φ and obtain a new equivalent formula φ'.

Based on the previous elimination methods, in the rest of the paper we consider $\mathrm{LTL}^{\mathcal{T}}$ with only $\lhd$ in front of variables. The fragment of $\mathrm{LTL}^{\mathcal{T}}$ that does not use the $\lhd$ operator is denoted by $\mathrm{LTL}^{\mathcal{T}}_{\not\lhd}$, which is the fragment used in [37]. We sometimes refer to this fragment as fetch-less $\mathrm{LTL}^{\mathcal{T}}$, which is exactly the logic obtained by replacing in LTL atomic propositions with literals from $\mathcal{T}$.

Fetch within bounds. Given an $\mathrm{LTL}^{\mathcal{T}}$ formula, we say that the formula φ *fetches within bounds* whenever every evaluation of a term required to evaluate $(\sigma,i) \models \varphi$ for every σ and i does not require the default value d in the rule for evaluating $[\![\lhd x]\!]_{(\sigma,i)}$. If every term that contains a fetch variable $\lhd x$ occurs in a literal l that is within the scope of a $\bigcirc$ operator in φ, then φ fetches within bounds. All the formulae in the paper fetch within bounds.

Applying $\lhd$ to terms. Extending LTL$^{\mathcal{T}}$ allowing $\lhd$ to be applied to arbitrary term t does not add expressively, because we can apply recursively the rewriting $\lhd f(s_1, \ldots, s_n) \mapsto f(\lhd s_1, \ldots, \lhd s_n)$ until $\lhd$ is applied only to variables. It is easy to see that this rewriting preserves semantics, since $[\![\lhd f(s_1, \ldots, s_n)]\!]_{(\sigma,i)} = [\![f(\lhd s_1, \ldots, \lhd s_n)]\!]_{(\sigma,i)}$ for every σ and i.

Reactive Synthesis for LTL$^{\mathcal{T}}$. Given an LTL$^{\mathcal{T}}$ formula φ the reactive synthesis problem for LTL$^{\mathcal{T}}$ splits the set of (theory) variables in $\mathit{Vars}(\varphi)$ between environment variables V_E and system variables V_S. In the corresponding game the environment and system players alternate choosing valuations of the variables they control, whose valuations are now theory values. An infinite sequence of moves forms now a trace σ in $\Sigma_{\mathcal{T}}(V_E \cup V_S)^\omega$, which is winning for the system if $\sigma \models \varphi$.

We classify the literals in φ as $L_E = \{l \mid \mathit{Vars}(l) \subseteq V_E\}$ and $L_S = \mathit{Lits}(\varphi) \setminus L_E$. A strategy is again a tuple $M : \langle Q, q_0, T\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state and $T \subseteq Q \times Q \times (2^{L_E}) \times (2^{L_S})$. In this case $t.env$ is a valuations of the environment literals and $t.sys$ is a valuations of the system literals. Given a transition t the $\mathcal{T}$ formulas E_t and S_t characterize the valuations of theory variables that satisfy the valuations of the literals in t:

$$E_t(\overline{x}) : \bigwedge_{l_e \in t.env} l_e(\overline{x}) \wedge \bigwedge_{l_e \notin t.env} \neg l_e(\overline{x})$$

$$S_t(\overline{x}, \overline{y}) : \bigwedge_{l_s \in t.sys} l_s(\overline{x}, \overline{y}) \wedge \bigwedge_{l_s \notin t.sys} \neg l_s(\overline{x}, \overline{y}) \qquad t(\overline{x}, \overline{y}) : E_t(\overline{x}) \wedge S_t(\overline{x}, \overline{y})$$

We call E_t and S_t the *characteristic formulas* of a transition. The notion of legal strategy for the environment and the system are analogous the Boolean cases (for the environment legality all valuations of the variables of the system are considered for each environment move, and for the system legality all environment moves have a response). The following holds:

Theorem 1. *Let φ be an* LTL$^{\mathcal{T}}$ *specification.*
- *If there is winning strategy for the system player then φ is realizable.*
- *If there is winning strategy for the environment player then φ is unrealizable.*

Proof. (Sketch). Let φ be an LTL$^{\mathcal{T}}$ specification. Assume there is a winning strategy M for the system player. Hence, $\mathit{Traces}(M) \models \varphi$. We show that at each step the environment can play every possible move, i.e., for every suffix $(\overline{x}_0, \overline{y}_0) \ldots (\overline{x}_k, \overline{y}_k))$ of a trace $\sigma \in \mathit{Traces}(M)$ and every valuation $\overline{x}_{k+1}$ there is a $\overline{y}_{k+1}$ and trace σ' in $\mathit{Traces}(M)$ whose prefix is $(\overline{x}_0, \overline{y}_0) \ldots (\overline{x}_k, \overline{y}_k)(\overline{x}_{k+1}, \overline{y}_{k+1})$. Since M is legal for the system, for every q and $\overline{x}$ there is a transition t such that (1) $E_t(\overline{x})$ and (2) there is $\overline{y}$ with $S_t(\overline{x}, \overline{y})$. For $i = 0$ to k let us pick one such transition such that (1) and (2) hold for $\overline{x}_i$ and $\overline{y}_i$. Let q be the state reached. For $k + 1$ let us pick a transition t out of q such that $E_t(\overline{x}_{k+1})$ holds and let $\overline{y}_{k+1}$ be such that $S_t(\overline{x}_{k+1}, \overline{y}_{k+1})$ holds. Let q' be the reaching state. Since k is arbitrary the result holds. The result for the environment is analogous. □

Note that, unfortunately, the opposite implications do not hold in general. If they did, one could enumerate the strategies of the system and the environment and traverse these sets in parallel, checking whether the strategies are legal and winning for the corresponding player. One of the two searches is guaranteed to terminate revealing the correct answer to realizability for $\mathrm{LTL}^{\mathcal{T}}$, which is undecidable, contradicting Thm .3. It is also easy to show that the reverse implications cannot for every specification for the system (or for every specification for the environment). Given an $\mathrm{LTL}^{\mathcal{T}}$ specification φ we can construct another $\mathrm{LTL}^{\mathcal{T}}$ specification φ' over the same theory such that φ is realizable if and only if φ' is unrealizable.

The following is shown in [37], which is extended in [38,36] to full synthesis procedures using Skolem function generation or SMT queries to generate concrete theory outputs.

Theorem 2. *Realizability of* $\mathrm{LTL}^{\mathcal{T}}_{\not\lhd}$ *specifications is decidable.*

Example 1. Consider $\mathcal{T} = \mathcal{T}_{\mathbb{Z}}$ and the following specification φ_1 (left):

$$\varphi_1 : \square \begin{pmatrix} (x < 0) \rightarrow \bigcirc(y \geq x) & \wedge \\ (x \geq 0) \rightarrow (y < x) & \end{pmatrix} \qquad \varphi_1^{\mathbb{B}} : \square \begin{pmatrix} l_0 \rightarrow \bigcirc l_2 & \wedge \\ l_1 \rightarrow l_3 & \end{pmatrix}$$

where the variable x belongs to the uncontrollable environment and the variable y to the system. The environment strategy that assigns $x : -1$ at timestep 0 and $x : 1$ at timestep 1 is winning, so φ_1 is unrealizable. Note that the arena of the corresponding game is infinite so a naive explicit representation is not feasible. One approach to realizability [37] is to abstract φ_1 into an equi-realizable LTL specification. A naive abstraction of φ_1 is to replace each literal by a fresh Boolean variable like $\varphi_1^{\mathbb{B}}$ (right), where l_0 and l_1 belong to the environment and l_2 and l_3 belong to the system. Unfortunately, realizability between φ_1 and $\varphi^{\mathbb{B}}$ is not preserved because, in φ_1, literals $(y \geq x)$ and $(y < x)$ cannot happen at the same time, whereas l_2 and l_3 forget this constraint in $\varphi_1^{\mathbb{B}}$. Thus, a sound abstraction method would require to add additional constraints to $\varphi_1^{\mathbb{B}}$ that capture the dependencies between the literals.

If the system controls all variables reactive synthesis becomes satisfiability, which is undecidable for $\mathrm{LTL}^{\mathcal{T}}_{\lhd}$ [15] (and can be trivially extended for $\mathrm{LTL}^{\mathcal{T}}$).

Theorem 3. *Realizability of arbitrary* $\mathrm{LTL}^{\mathcal{T}}$ *specifications is undecidable.*

Example 2. Recall φ_1 from Ex. 1 and consider $\varphi_2 : \big(\bigcirc\square[x < \lhd x] \rightarrow \bigcirc\varphi_1\big)$, which encodes that x must be monotonically decreasing. Note that the strategy for the environment mentioned in Ex. 1 does not violate φ_2 because once the environment plays a value of x below 0, the upcoming valuations of x will need to also be below 0 (because of $\square[x < \lhd x]$), so the strategy of playing some x below 0 at timestep k and some x above 0 in timestep $k+1$ is no longer legal. In fact, φ_2 is realizable. Note that if $\lhd x$ is substituted by a fresh variable z controlled by the environment, that is $\varphi_2' : \big(\bigcirc\square[x < z] \rightarrow \bigcirc\varphi_1\big)$ then φ_2' is again unrealizable. This naive abstraction over-approximates the power of the environment removing the monotonicity constraint. In other words, φ_2 and φ_2' are not equi-realizable.

4 Counter Example Guided Reactive Synthesis

We present now a novel method called *Counter-Example Guided Reactive Synthesis* (CEGRES) modulo theories, shown schematically in Fig. 1. Given a specification φ in a logic L, the CEGRES method first computes an abstraction φ' in a less expressive variant L'. The formulas φ and φ' are not necessarily equirealizable. Then, a synthesis tool for L' is used to get the verdict v' (whether φ' is `real` or `unreal`), and a strategy M' for the winning player. Then, the method examines M' to check whether it corresponds to a consistent strategy for the higher logic L. If M' is consistent in L, then the verdict is conclusive, and we derive a machine M in L based on the machine M' in L', returning v' as verdict and M as answer. If, on the other hand, M' is inconsistent in L then the outcome is not conclusive. The method then finds a counterexample based on the inconsistencies in M and computes additional constraints which are incorporated to φ, reiterating the process with the strengthened formula. Note how in CEGRES we use an existing synthesis tool for a simpler logic to build a synthesis method for a richer logic. Both methods receive a temporal logic formula, with a split of the set of variables between environment and system, and both return a realizability outcome with a finite machine as witness.

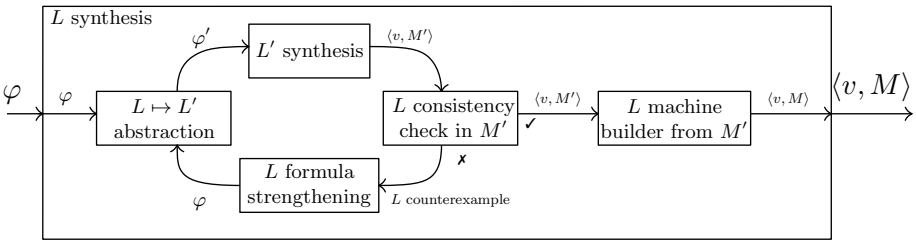

Fig. 1: CEGRES to generate a verdict v (real/unreal) and strategy M from φ.

CEGRES for LTL$^{\mathcal{T}}_{\not\triangleleft}$. We first illustrate the CEGRES method for LTL$^{\mathcal{T}}_{\not\triangleleft}$ specifications, using classical LTL as the internal synthesis tool box. The reactive synthesis problem for LTL$^{\mathcal{T}}_{\not\triangleleft}$ was solved in [37] using an explicit exhaustive exploration method. In contrast, CEGRES is much more lightweight because it captures only the necessary $\mathcal{T}$ information and avoids exploring all the potential reactions. As we will see in Section 6 this is the fastest algorithm up to date for realizability and synthesis for LTL$^{\mathcal{T}}_{\not\triangleleft}$. The abstraction in CEGRES for LTL$^{\mathcal{T}}_{\not\triangleleft}$ combines a naive abstraction $\varphi^{\mathbb{B}}_{-}$ of φ—that simply translates literals to Boolean variables—with a set of assumptions A that constrain the plays of the environment and a set of guarantees G that constrain the legal responses of the system. The resulting formula is $\varphi^{\mathbb{B}}$: $\square A \rightarrow (\square G \wedge \varphi^{\mathbb{B}}_{-})$, where A and G are non-temporal Boolean formulas that constrain the combinations of literal valuations preventing $\mathcal{T}$ inconsistencies.

We instantiate the CEGRES components for $\text{LTL}^{\mathcal{T}}_{\not\triangleleft}$ as follows:

- L is $\text{LTL}^{\mathcal{T}}_{\not\triangleleft}$ (φ is an $\text{LTL}^{\mathcal{T}}_{\not\triangleleft}$ formula) and L' is LTL;
- φ' is the (propositional) LTL formula obtained by replacing literals by fresh Boolean variables (the Boolean variables that abstract L_E are assigned to the environment, and the variables that abstract L_S, to the system);
- the internal solver is an off-the-shelf LTL synthesis tool (e.g., Strix [30]) and M' is a Mealy machine encoding a strategy of the winning player.
- the derivation of M from M' uses either Skolem function generation [36] or SMT queries [38] on $\forall\exists$ queries that are guaranteed to be satisfiable.
- the CEGRES refinement is based on the following lemma, where a counterexample is a tautology in $\mathcal{T}$ that is falsified in a transition of M'.

Lemma 1. *Given a strategy M' for a φ' that is realizable (resp. unrealizable), if for every transition $t \in M'$, the formula: $\exists\overline{x}.E_t(\overline{x}) \wedge \forall\overline{x}.(E_t(\overline{x}) \rightarrow \exists\overline{y}.S_t(\overline{x},\overline{y}))$ is valid, then φ is realizable (resp. unrealizable).*

The proof proceeds by contradiction, assuming that φ is not equi-realizable with respect to φ', taking a loosing play according to M and generating a loosing play played according to M', which is a contradiction.
The CEGRES method for $\text{LTL}^{\mathcal{T}}_{\not\triangleleft}$ is shown in Alg. 1 and it works as follows.

- The abstraction method (line 5) assigns a fresh Boolean variable for each literal, and tags it as controlled by the environment if it only contains variables from the environment, and controlled by the system otherwise. Then, $\varphi^{\mathbb{B}}$ is obtained from φ by replacing each literal by its associated Boolean variable.
- Then, an off-the-shelf synthesis tool receives $\varphi^{\mathbb{B}}$ as input and produces a Mealy machine M as a winning strategy for the system if $\varphi^{\mathbb{B}}$ is realizable, or for the environment if $\varphi^{\mathbb{B}}$ is unrealizable.
- If $\varphi^{\mathbb{B}}$ is unrealizable, the method *check_env* is invoked for each transition t, and line 16 checks whether $\exists\overline{x}.E_t(\overline{x})$ is valid. If the check fails for some

Algorithm 1: CEGRES for $\text{LTL}^{\mathcal{T}}_{\not\triangleleft}$.

```
1  Require: φ
2  A, G ← true
3  done ← false
4  while ¬done do
5      φ^B ← abstract(A → (G ∧ φ))
6      is_real, M ← synth(φ^B)
7      T ← get_trans(M)
8      done ← true
9      if ¬is_real then
10         foreach t:T do check_env(t);
11     else
12         foreach t:T do check_sys(t);
13 return is_real, M
14 check_env(t)
15     E(x̄) ← t.env
16     if ∃x̄.E(x̄) invalid then
17         A ← A ∧ ¬E(x̄)
18         done ← false
19 check_sys(t)
20     E(x̄) ← t.env
21     S(x̄, ȳ) ← t.sys
22     if (∀x̄.E(x̄) → ∃ȳ.S(x̄, ȳ)) invalid then
23         G ← G ∧ (S(x̄, ȳ) → ∃ȳ.S(x̄, ȳ))
24         done ← false
```

transition, the result is inconclusive because the environment may be winning by using illegal moves. In this case line 17 incorporates to A the fact that $E_t(\overline{x})$ is unsatisfiable and restarts the process with the refined formula. This is performed for all illegal environment moves detected.

– If $\varphi^{\mathbb{B}}$ is realizable, *check_sys* is invoked for every transition t, which checks for every $\overline{x}$ that makes $E_t(\overline{x})$ hold, there is a $\overline{y}$ that makes $S_t(\overline{x},\overline{y})$ hold as well (in line 22). If the formula is not valid, then the transition t is illegal: there is an $\overline{x}$ that satisfies $E_t(\overline{x})$ for which there is no $\overline{y}$ that satisfies $S_t(\overline{x},\overline{y})$. That is, $(\exists\overline{x}.E_t(\overline{x}) \wedge \neg(\exists\overline{y}.S_t(\overline{x},\overline{y})))$ is valid. In particular, $(\exists\overline{x}.\neg(\exists\overline{y}.S_t(\overline{x},\overline{y})))$ holds. That is, we found that the system can make the formula $S_t(\overline{x},\overline{y})$ hold only if the environment makes the formula $\exists\overline{y}.S_t(\overline{x},\overline{y})$ hold. We incorporate this information as a strengthening predicate into the specification $S_t(\overline{x},\overline{y}) \rightarrow \exists\overline{y}.S_t(\overline{x},\overline{y})$ to G, and we reiterate the process with the updated formula.

Alg. 1 finishes when there are no inconsistencies in M according to Thm. 1, generating a correct verdict and strategy. Alg. 1 is guaranteed to terminate, generating no more strengthening predicates than "reactions" allowed by the set of literals in the formula (see [37]). In practice this method generates considerably fewer strengthenings (which is the why it dramatically improves scalability). The following theorem establishes the correctness of CEGRES for LTL$^{\mathcal{T}}$.

Theorem 4 (Correctness of Alg. 1). *Alg. 1 terminates and produces an equi-realizable LTL formula $\varphi^{\mathbb{B}}$ along with a strategy for the winning player in φ.*

Proof. (Sketch). Consider an execution of Alg. 1. If the algorithm terminates, the strategy found is legal and winning for the corresponding player, so according to Thm. 1 the result is correct. Therefore we only need to prove termination. The formulas in line 21 are bounded by combinations of L_S so the set is bounded by $2^{|L_S|}$. In turn, the set of predicates $E(\overline{x})$ is bounded by $|L_E|$ plus the new literals that can be generated in the quantifier elimination of line 23 (the elimination of $\overline{y}$ in $\exists\overline{y}.S(\overline{x},\overline{y})$ can generate a new environment literal). Therefore the size of the set of environment literals in combinations from E is also bounded by $|L_E| + 2^{|L_S|}$. This also limits the number of $E(\overline{x})$ formulas in line 16. In every iteration either the main loop terminates or at least a new formula from a finite set is added to A or to G. Hence, Alg. 1 terminates with the correct answer. □

5 Reactive Synthesis for LTL$^{\mathcal{T}}$

We now study CEGRES for LTL$^{\mathcal{T}}$. The main idea is to (1) abstract all fetch terms by fresh variables controlled by the environment (thus over-approximating the power of the environment). Then (2) solve the resulting LTL$^{\mathcal{T}}_{\not\lhd}$ specification and analyze the winning strategy, which is either conclusive, or we learn how to limit the over-approximation power of the environment to avoid similar illegal moves.

Example 3. Consider the LTL$^{\mathcal{T}}$ specification over $V_E : \{x\}$ and $V_S : \{y\}$:

$$\square(x < 10) \rightarrow \bigcirc\square(x < \lhd y).$$

This specification states that, assuming that the environment variable x is always less than 10, then the system can provide a value y that is always greater than the next value of x. This specification is realizable (for example, by simply producing $y : 11$ at all instants). If we replace the expression $\lhd y$ by a fresh variable y' that represents the previous value of y and assign it *to the environment*, we obtain a property in $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ over $V_E : \{x, y'\}$ and $V_S : \{y\}$:

$$\square(x < 10) \rightarrow \bigcirc\square(x < y')$$

which is trivially unrealizable because the environment can play $x : 9, y' : 9$ at all instants. However, if we strengthen the assumption with $\bigcirc(y' < 10) \rightarrow (y < 10)$ (which is trivially an $\text{LTL}^{\mathcal{T}}$ temporal validity, that is, it holds at all instants of all traces), which is equivalent to $\neg(y < 10) \rightarrow \bigcirc\neg(y' < 10)$):

$$\square[(x < 10) \wedge \big(\bigcirc(y' < 10) \rightarrow (y < 10)\big)] \rightarrow \bigcirc\square(x > y')$$

The resulting $\text{LTL}^{\mathcal{T}}$ formula is realizable.

5.1 A General CEGRES Method for $\text{LTL}^{\mathcal{T}}$.

We instantiate the CEGRES components in Fig. 1 as follows:

- the input logic L is $\text{LTL}^{\mathcal{T}}$, so φ is an $\text{LTL}^{\mathcal{T}}$ formula.
- the logic L' after the abstraction is $\text{LTL}^{\mathcal{T}}_{\not\lhd}$ and φ' is the formula obtained by replacing every expression $\lhd v$ by a fresh *environment variable* v'.
- the inner solver is Alg. 1 from Sect. 4.

Before we describe the method in detail we need some auxiliary definitions. We use $\overline{v}' : V_F$ to describe the set of primed variables that have replaced the fetch terms. For example, in the example above, we have $V_E : \{x\}$, $V_S : \{y\}$, and $V_F : \{y'\}$. The variables in V_F belong to the environment.

The next definition captures a constraint that every pair of consecutive steps in a strategy M must satisfy for M to be legal according to the semantics of $\lhd$.

Definition 1. *Given a candidate strategy M and given two consecutive transitions t_{pre} and t (i.e., $t_{pre}.to = t.from$), the following formula $C_{t_{pre},t}$ is the* consistency condition *for (t_{pre}, t).*

$$\begin{array}{l}\forall(\overline{x}_0 : V_E), (\overline{v}'_0 : V_F), (\overline{y}_0 : V_S).t_{pre}(\overline{x}_0 \cup \overline{v}'_0, \overline{y}_0) \rightarrow \\ \quad\left(\begin{array}{l}\exists(\overline{x}_1 : V_E), (\overline{v}'_1 : V_F).\big(\varphi_{pr} \wedge E_t(\overline{x}_1 \cup \overline{v}'_1)\big) \wedge \\ \forall(\overline{x}_1 : V_E), (\overline{v}'_1 : V_F).(\varphi_{pr} \wedge E_t(\overline{x}_1 \cup \overline{v}'_1) \rightarrow (\exists\overline{y}_1 : V_S).t(\overline{x}_1 \cup \overline{v}'_1, \overline{y}_1)\end{array}\right)\end{array}$$

where

$$\varphi_{pr} \stackrel{\text{def}}{=} \bigwedge_{v'_1 \in \overline{v}'_1} (v'_1 = v_0).$$

This formula essentially captures that the values of the variables in a transition coincide with the values of their corresponding fetched variables in the successive transition.

The following theorem guarantees that if a strategy for an abstracted formula satisfies the constraint for all two consecutive transitions, then the strategy is valid for the original $\mathrm{LTL}^{\mathcal{T}}$ formula.

Theorem 5. *Let φ be an $\mathrm{LTL}^{\mathcal{T}}$ formula and φ' the $\mathrm{LTL}^{\mathcal{T}}_{\not\lhd}$ formula obtained by replacing all fetch terms by primed variables. Given a winning strategy M' for φ', if every two consecutive transitions t_{pre} and t satisfy $C_{t_{pre},t}$, then we can obtain a winning strategy M for the same player for φ.*

Proof. (Sketch). It is enough to build a winning strategy M for φ from M'. M contains the same states and transitions as M', but at every step the environment player is additionally given the values of the fetched variables at the previous instant, and its characteristic formula constrained to choose these values at every point (except at the first instant). It holds that the environment characteristic formula with these additional constraints is always satisfiable (due to the check performed before terminating). This strategy M preserves the semantics of the operators, it is a legal strategy and it is a winning strategy for the same player as the winner of M'. □

If $C_{t_{pre},t}$ fails for some consecutive t_{pre} and t, the CEGRES method will strengthen the environment assumptions of φ by learning tautologies using the following concepts.

Definition 2 (environment ability). *Given a strategy M and transition t with environment and characteristic formulas E_t and S_t, the following formula*

$$A_t(\overline{v}) \stackrel{\text{def}}{=} \exists(\overline{x} : V_E).E_t(\overline{x} \cup \overline{v})$$

captures the ability *of the environment of choosing environment variables $\overline{x}$ and satisfy the environment play of t, given the variables $\overline{v}$.*

Given an ability formula A_t extracted from transition t we assign $R_t(\overline{v})$ to be the formula $\bigcirc A_t(\overline{v}') \rightarrow A_t(\overline{v})$. The following lemma justifies that formulas of the form of R_t are tautologies, called *reactive tautologies*.

Lemma 2. *Given a trace σ, an instant i such that, for every $v' : V_F$, $\sigma(i)(v) = \sigma(i+1)(v')$. Let $Q(\overline{v}')$ be a predicate over V_F. Then, $(\sigma, i) \models \bigcirc Q(\overline{v}') \rightarrow Q(\overline{v})$.*

The lemma follows immediately because the value of $Q(v')$ at time $i+1$ is exactly the value of $Q(v)$ at time i.

An example of reactive tautology, shown in Ex. 3 above is $\big(\bigcirc(y' < 10) \rightarrow (y < 10)\big)$, obtained from $y' < 10$. Given a formula $C_{t_{pre},t}$ for a pair of consecutive transitions, adding R_t (the reactive tautology of the second transition) to the environment assumptions of φ prevents the inner synthesis step from searching similarly illegal transitions in the next iteration of the CEGRES method. Unfortunately, this does not guarantee termination because the machine in the new iteration can contain other transitions with new literals and formulas.

Based on Thm. 5 we present a CEGRES algorithm that uses the concept

Alg. 2: CEGRES for Full LTL$^{\mathcal{T}}$.

```
1  Require: φ
2  while true do
3  |  φ^e ← overApprEnv(φ)
4  |  M ← synth(φ^e)
5  |  if φ^e is real then
6  |  |  return real
7  |  else if consTmp(M) then
8  |  |  return unreal
9  |  else
10 |  |  φ ← addTempTaut(M, φ)
```

of reactive tautology in the learning phase, in Alg. 2 (left). Since our method over-approximates the power of the environment, a `real` verdict immediately implies that the original specification φ is realizable with the strategy found. Otherwise, the checker component checks (line 7) whether for every pair of successive transitions t_{pre} and t, the formula $C_{t_{pre};t}$ in Thm 5 holds. If every pair is fine, Thm. 5 implies that the machine obtained is winning for the environment. Otherwise, the reactive tautologies R_t learned are added (line 10) to the environment assumptions of φ for every outgoing transition t of a failing pair.

The check for $C_{t_{pre};t}$ above can be accelerated using the following result.

Lemma 3. *If R_t is in the environment assumption of φ, then every pair of consecutive transitions t_{pre} and t satisfies $C_{t_{pre};t}$ in Thm 5.*

Proof. (Sketch). Follows from simple algebraic transformations □

Example 4. Consider again $\varphi : \square(x > 10) \rightarrow \bigcirc\square(x > \lhd y)$ from Ex. 3. Alg. 2 proceeds as follows:

1. Abstract φ into φ'.
2. The inner LTL$^{\mathcal{T}}$ computes internally the abstraction $\varphi^{\mathbb{B}} : \square l_0 \rightarrow \bigcirc\square(l_1)$, where both l_0 and l_1 belong to the environment.
3. Next, an LTL synthesis tool discovers that $\varphi^{\mathbb{B}}$ is unrealizable and provides a Mealy machine $M_{\mathbb{B}}$ that witnesses the strategy of the environment. $M_{\mathbb{B}}$ has a single transition t, whose corresponding environment play is $E_t = (x \leq y') \wedge (x > 10)$ and the system play is $S_t = true$, which means that the system's choice is irrelevant.
4. CEGRES for LTL$^{\mathcal{T}}_{\not\lhd}$ (Alg. 1) checks that $M_{\mathbb{B}}$ has no inconsistencies in $\mathcal{T}$.
5. Then, the temporal analysis reveals that $M_{\mathbb{B}}$ might not be valid because the check of traversing t twice:

$$\begin{array}{l} \forall x_0, y_0', y_0.((x_0 \leq y_0') \wedge (x_0 > 10)) \rightarrow \\ \Big(\exists x_1, y_1'.\big((y_0 = y_1') \wedge ((x_1 \leq y_1') \wedge (x_1 > 10))\big) \wedge \\ \forall x_1, y_1'.((y_0 = y_1') \wedge (x_1 \leq y_1') \wedge (x_1 > 10)) \rightarrow (\exists y_1).((x_1 \leq y_1') \wedge (x_1 > 10))\Big) \end{array}$$

is not valid. In particular, it is not true that for every y_0 there are x_1 and y_1' such that $(y_0 = y_1') \wedge ((x_1 \leq y_1') \wedge (x_1 > 10))$—it suffices to take any y_0 lower than 11. From the transition t, the algorithm explores the conditions that system-controlled variables must meet to make E_t satisfiable, in particular, it explores the ability $A_t(y') = \exists x.(x \leq y') \wedge (x > 10)$ or, equivalently via quantifier elimination, $A_t(y') = (y' < 11)$.

6. Then, $R_t : \bigcirc(y' < 11) \rightarrow (y < 11)$ is added to φ, obtaining $\varphi' : R_t \rightarrow \varphi$, invoking again the abstraction and LTL synthesis and obtaining a new M', that this time is consistent in $\text{LTL}^{\mathcal{T}}$.
7. Finally, the algorithm constructs a machine M in $\text{LTL}^{\mathcal{T}}$ from M' and returns the realizability verdict along with the witness M[3].

5.2 CEGRES for Bounded Domains

We now study CEGRES for $\text{LTL}^{\mathcal{T}}$ for a very important practical domain: bounded theories and in particular bounded arithmetic. This theory is very important in the specification of embedded critical systems. A typical scenario in embedded systems uses numeric domains (like 32 bit integers) where bit operations are used instead of ideal integers.

In theory, one can blast all variables in the specification, for example for an 8 bit signed variable d, one would introduce 256 Boolean variables $d_{-127} \dots d_{128}$, and include all corresponding precise rules of arithmetic, like $d_0 < d_1$, $d_1 < d_2$, $\dots$, and temporal rules $d_0 \rightarrow \bigcirc \lhd d_0$ and $d_1 = d_0 + 1$, etc. This blasting method generates 256 atomic propositions for each 8-bit numeric variable which does not scale for the example above. Even though one can use a logarithmic encoding and capture arithmetic operations and relations, LTL synthesis does not scale either. Instead, we propose using Alg. 2 directly—which results in a much faster procedure, and it is guaranteed to terminate for bounded domains. Specifications in $\text{LTL}^{\mathcal{T}}$ are much more succinct and close to the human intention than their blasted versions.

We envision the impact of our solution to be analogous to bit-vectors solvers in SMT, in the sense that practitioners can trust precise solutions which can also be much faster (and in our case produce much simpler strategies). For the sake of simplicity we consider $\text{LTL}^{\mathcal{T}}$ for the theory $\mathcal{T} = \mathbb{Z}^k$ of bounded integer arithmetic (for example $\mathbb{Z}^{32}$ for 32 bit signed integer arithmetic).

Example 5. Consider the following scenario of a water tank evolution. At each timestep, the environment provides a *stamina* value d (between 0 and 40) that models the effect on the water tank of inflow and outflow of water, where two valves (input and output) control the income and output of water into the tank. Additionally, there is a non-observable *rain* of extra water that can be added to the tank when the input valve is opened, and also some non-observable *leak* of water that controls the amount of outgoing water when the output valve is opened. We model the effect of the valves as inequalities rather than exact equalities. The safety goal of the controller is to make the water level wl, which the system controls, between 0 and 1000. We express this as the following $\text{LTL}^{\mathcal{T}}$ specification with, for example, domain $\mathbb{Z}^{16}$ (that is, 16-bit integers).

$$\varphi : \square(0 < d < 40) \rightarrow \square \left(\begin{array}{ccc} \bigcirc(wl > \lhd wl + d) & \vee & \bigcirc(wl < \lhd wl - d) \\ & \wedge & \\ & (0 < wl < 1000) & \end{array} \right)$$

[3] The construction of the full machine for φ from the inner M' can be done analogously to [36,38] but the details are not included here due to space limitations.

One simple strategy that our method finds is to let the tank fill until it is close to being full or let it drain until it is nearly empty.

One important question is whether CEGRES for $\text{LTL}^{\mathcal{T}}$ and bounded domains like $\mathbb{Z}^{32}$ terminates. The following theorem provides a positive answer.

Theorem 6. *Alg. 2 terminates for every* $\text{LTL}^{\mathcal{T}}$ *formula with a bounded domain.*

Proof. (Sketch). The idea of the proof is to first observe that all reactive tautologies that can be found are equivalent to a finite collection of ground axioms of arithmetic for bounded domains, and that this collection is finite. Then, at each iteration of the algorithm, either the correct solution is found or a new reactive tautology is found that corresponds to a new set of axiom that the previous tautologies did not cover. In other words, every new tautologies strictly expands the set of axioms covered. Since the largest set of axioms is finite, after a finite number of iterations, either the algorithm has already converged providing the right answer or all axioms are captured. At this point, the algorithm is guaranteed to terminate in the next iteration. □

Even though the bound provided by Thm. 6 can be large, in practice the algorithm runs much faster and produces a much smaller and understandable controller than a blasting strategy. This is because the algorithm can reuse moves in the strategy for many different states, particularly when the situation is far from the boundaries established in the specification. Many times the algorithm takes comparable time and yields equivalent systems when using 8 bit, 16 bit or 32 bit arithmetic (or changing boundaries like 40 and 1000 to 4000 and 60000 above), where a blasting strategy would not scale for large bound, and generate a convoluted solution when it terminates for small domains.

6 Empirical Evaluation

In order to evaluate the applicability of our approach, we carried out an empirical evaluation that intends to address the following research questions:

- **RQ1**: Is CEGRES faster for $\text{LTL}^{\mathcal{T}}_{\not\Diamond}$ than previous methods?
- **RQ2**: How well does CEGRES for full $\text{LTL}^{\mathcal{T}}$ perform when solving TSL-MT style specifications?
- **RQ3**: Is CEGRES for full $\text{LTL}^{\mathcal{T}}$ capable of solving arbitrary $\text{LTL}^{\mathcal{T}}$ specifications?

To address these questions, we implemented a prototype tool for synthesizing $\text{LTL}^{\mathcal{T}}$ specifications called `syntheos`[4]. The tool is written in Python and uses Z3 [31] for the SMT queries necessary in the Boolean abstraction, and Strix [30] to check the realizability of the abstraction. Strategies are Mealy machines in the standard `HOA` format. All our tests were run using a MacBook Air (M1, 2020) machine, with processes limited to 16GB.

[4] Syntheos is available at `https://github.com/imdea-software/syntheos`

#	Name	$\|V\|$	$\|L\|$	Prv.	T_{prv}	T_{CG}
1	Lift 1	1	7	2^{2^7}	31.77	1
2	Lift 2	2	4	2^{2^4}	0.7	1
3	Lift 1-2	3	11	$2^{2^{11}}$	TO	1
4	Lift 3	1	3	2^{2^3}	0.52	1
5	Lift 1-3	4	14	2^{2^3}	TO	1
6	Lift 4	1	2	2^{2^2}	0.09	1
7	Lift All	5	16	$2^{2^{16}}$	TO	1
8	Train 1	1	3	2^{2^3}	0.04	1
9	Train 2	2	1	2^{2^1}	0.04	1
10	Train 3	1	3	2^{2^3}	0.21	1
11	Train 4	1	1	2^{2^1}	0.05	1
12	Train 1-4	4	9	2^{2^1}	TO	1
13	Train 5	4	5	2^{2^5}	112.5	1
14	Train 6	3	5	2^{2^5}	359.3	1
15	Train 7	4	12	$2^{2^{12}}$	6571	1
16	Train 5-7	11	22	$2^{2^{12}}$	TO	1
17	Train All	19	26	$2^{2^{26}}$	TO	2
18	Connect	2	2	2^{2^2}	0.09	1
19	Cooker	3	5	2^{2^5}	2.81	1
20	Usb 1	2	3	2^{2^3}	0.17	1
21	Usb 2	3	5	2^{2^5}	231.9	1
22	Usb All	5	8	2^{2^8}	TO	1
23	Stages 1	8	8	2^{2^8}	18.19	1
24	Stages 2	3	6	2^{2^6}	194.8	1
25	Stages All	11	14	$2^{2^{14}}$	TO	1

Table 1: Comparison of [37] with our CEGRES method (time is in seconds).

RQ1: CEGRES for LTL$^{\mathcal{T}}_{\not\lhd}$. In order to address **RQ1**, we consider the industry-inspired benchmarks from [37] and compare with their results. Tab. 1 shows the name of each benchmark, the number of variables ($|V|$) and literals ($|L|$), the number of strengthenings in eager methods (Prv.), the running time of the different versions of the eager algorithms from [37] with accelerating heuristics (T_{prv}), the time for Alg. 1 (T_{CG}) *TO* means time-out after 12 hours of execution. As expected, CEGRES outperforms [37] in running time, especially for large instances (which cannot be solved with eager methods). Indeed, these experiments show that eager methods are not tractable for realistic-size instances. We hypothesize that CEGRES performs better because it produces a smaller amount of refinements, that is., the number of literals in the final specification is smaller. This means that the original specification suffers less modifications. Comparing specification interpretability before and after the abstraction processes among different methods is a possible research direction (where the preferred method is the one that modifies the original specification the least).

RQ2: CEGRES for the TSL-MT fragment. In order to answer **RQ2**, we first observe that the syntax of LTL$^{\mathcal{T}}$ is strictly subsumes that of TSL-MT. We use a complete recent set of benchmarks from [20], which consists of TSL-MT specifications in integer arithmetic. The columns I and O in Tab. 2 indicate the number of input and output variables in the specification, respectively. The column R? in that table indicates whether the specification is realizable or not. The column CG indicates the time it takes for the CEGRES algorithm to finish. The columns that are not CG correspond to specialized TSL-MT solvers: MO, WI are monitored and non-monitored versions for `tslmt2rpg` [20], RA corresponds to `raboniel` [25] and TE corresponds to `temos` [7]. We ran all the experiments in the same machine. As we can see, in most cases, CG either timeouts (TO) after

Name	I	O	R?	CG	MO	WI	RA	TE
Box Lim.	2	2	R	TO	7	1	1	MO
Box	2	2	R	394	33	3	1	TO
Diagonal	2	1	R	MO	43	1	5	MO
Evasion	4	2	R	MO	82	4	2	TO
Follow	4	2	R	TO	TO	18	TO	TO
Solitary	2	0	R	278	8	1	1	ER
Sqr-5x5	2	2	R	MO	203	10	43	TO
E.Smp 3	1	0	R	111	27	2	1	MO
E.Smp 4	1	0	R	333	47	2	1	MO
E.Smp 5	1	0	R	1124	74	4	4	TO
E.Smp 8	1	0	R	TO	211	8	23	TO
E.Smp10	1	0	R	TO	356	11	98	TO
E.Sgnl 3	2	1	R	MO	MO	MO	17	MO
E.Sgnl 4	2	1	R	MO	MO	MO	111	MO
E.Sgnl 5	2	1	R	TO	MO	MO	735	MO
G-real	3	1	R	MO	330	MO	3	TO
G-ur-1	2	1	U	MO	27	TO	TO	–
G-ur-2	2	1	U	1	28	TO	ER	–
G-ur-3	1	0	U	1	46	TO	ER	–
F-real	3	1	R	4	66	TO	ER	ER
F-unreal	2	1	U	MO	104	TO	TO	–
FGcntr1	1	0	U	1	34	TO	ER	–
FGcntr2	2	1	U	5	137	TO	1	–
GF-real	1	1	R	MO	7	TO	ER	ER
GF-ur	1	0	U	28	18	TO	TO	–
GF-contr	1	1	U	TO	8	TO	ER	–

Name	I	O	R?	CG	MO	WI	RA	TE
thrmF	2	1	R	MO	77	TO	TO	MO
thrmFur	2	1	U	MO	142	TO	TO	–
thrmGF	2	1	R	MO	TO	TO	TO	MO
thrmGFur	2	1	U	MO	136	TO	TO	–
ordvisit	2	1	R	MO	488	TO	TO	TO
ptrl-alr	2	2	R	TO	TO	TO	TO	MO
ptrl	2	0	R	MO	277	TO	ER	TO
charging	3	1	R	6	287	TO	TO	TO
charge-ur	3	1	U	MO	39	TO	TO	–
rtarget	3	1	R	MO	398	TO	TO	MO
rtrgt-ur	3	1	U	MO	314	TO	TO	–
unordvisit	2	0	R	TO	253	TO	ER	TO
helipad	3	6	R	MO	102	MO	TO	MO
delivery	5	2	R	38	88	MO	TO	MO
tasks	3	0	R	191	TO	TO	ER	MO
tasks-ur	3	0	U	186	334	TO	ER	–
buffer-st	3	1	R	47	TO	TO	45	MO
heli-cont	2	0	U	1	182	15	TO	–
ord-ch.	2	0	R	TO	TO	TO	ER	TO
precise	2	0	R	12	TO	ER	ER	–
st.-GF-64	2	0	R	1	TO	TO	ER	MO
unord-ch	3	0	R	TO	TO	TO	TO	TO
unsat	2	1	U	MO	133	TO	ER	–
vacuous	2	1	R	MO	24	TO	ER	ER
dschrg-GF	2	1	R	MO	19	TO	TO	ER
fut-work	2	1	R	1	–	–	–	–

Table 2: Comparison between TSL-MT methods and our approach for all the benchmarks by [20] (measured in s).

20 minutes or solves the benchmark, but in a worse time than some TSL-MT competitors. This is expected because we are comparing CEGRES—which is a general LTL$^{\mathcal{T}}$ solver—against tools that are specific for TSL-MT. However, we find some instances in which CG finishes but no TSL-MT solver can.

It is very important to note that the most difficult task for CG is to find the appropriate reactive tautologies for each benchmark. Once the tautologies have been found, the proof finishes in one iteration (see **RQ3**), so checking proofs once the appropriate tautologies are provided is fast. Proofs are sometimes simplified by adding facts of the specification like $\square(x > \lhd x)$ or validities like $(\square(x > \lhd x)) \rightarrow \diamondsuit(x > 1000)$. Therefore, a line of future work is to explore the addition of simple arithmetic and temporal facts to specifications; for instance, generation of tautologies via the invariant generation of [20].

RQ3: CEGRES for General LTL$^{\mathcal{T}}$. Since the syntax of TSL-MT is oriented to extracting programs more easily (as the system is restricted to a finite number of known assignments), the previous benchmarks are not able to use all the expressive power of LTL$^{\mathcal{T}}$. An example is `fut-work` from Tab. 2, which is a

Name	I	O	R?	CG	TSL	Name	Auto	Oracle	TSL
Future-Work	1	0	U	1	–	AD-Real	1	1	–
E. Signal-3	2	1	R	6	–	AD-Unreal	1	–	–
E. Signal-4	2	1	R	7	–	TC-Real	3	1	–
E. Signal-5	2	1	R	9	–	TC-Unreal	1	–	–
Ordered-vis.	2	1	R	22	–	SL-Real	84	2	–
Buff-storage	3	1	R	63	–	SL-Unreal	1	–	–
Ordered-vis-ch.	2	0	R	7	–	MS-Real	3	1	–
Prec.-reachab.	2	0	R	6	–	MS-Unreal	1	–	–
Stor.-GF	2	0	R	2	–	PM-Real	16	1	–
Unordered-vs	3	0	R	26	–	PM-Unreal	1	–	–

Table 3: Showcasing performance of CEGRES with full LTL$^{\mathcal{T}}$ specifications and limitation of TSL-MT expressivity (measured in s).

benchmark from [20] that cannot be solved (and is not clear how to express) in previous tools because it includes a relation across time: $x = 0 \wedge \bigcirc\Box(x > \lhd x) \wedge \Diamond(x < 0)$. In practice, these relations are useful because they allow us to express rich dynamics of both players (as in Ex. 2).

Therefore, we evaluate now the merits of our tool with specifications in general LTL$^{\mathcal{T}}$, with no restriction in $\mathcal{T}$, temporal fragment or operators across time. Since there are no benchmarks to compare with, we randomly took some examples of Tab. 2, but modified them in order to include relations and randomly removed some subformulae. We also designed 5 additional benchmarks that illustrate the importance of expressing environment dynamics. In all of them, like in Ex. 2, an LTL$^{\mathcal{T}}_{\not\lhd}$ version that does not consider the environment dynamics properly using $\lhd$ results in an unrealizable verdict, whereas the full LTL$^{\mathcal{T}}$ version considers such dynamics and is realizable. Most importantly, the *oracle* version shows that CEGRES terminates in a single iteration of the outer loop (the one solving full LTL$^{\mathcal{T}}$) when key reactive tautologies are added manually in advance (as anticipated in **RQ2**). Tab. 3 shows the results, where we can see that TSL-MT tools do not express any of these benchmarks, while the reported times in CEGRES were very similar to Tab. 2. Also, note that all of these benchmarks are solved by Alg. 2 iteration due to the use of CEGRES for LTL$^{\mathcal{T}}_{\not\lhd}$ (Alg. 1).

RQ4: CEGRES for Bounded Domains. Additionally, we ask ourselves the following question: can, in some case, LTL$^{\mathcal{T}}$ be a better specification language than LTL bit-blasting and does CEGRES scales better? Ex. 5 shows that LTL$^{\mathcal{T}}$ can be more succinct than LTL, because it can symbolically represents numerous states. Now, we consider again Ex. 5 (WT-Signal) and a version with no environment (WT-Simple) both with different scalability parameters: see Tab. 4 and the figure at its side (where brown and blue correspond to LTL and black and red to LTL$^{\mathcal{T}}$). The results show that performance is not compromised when the parameters take larger values. Moreover, for Ex. 5 the blasting approach does not scale to bounds higher than 20, whereas in LTL$^{\mathcal{T}}$ this bound is not relevant

#	WL-Simple		WL-Signal	
	LTL	$\text{LTL}^{\mathcal{T}}$	LTL	$\text{LTL}^{\mathcal{T}}$
1	3	5	12	18
10	2738	1222	TO	1926
100	TO	32	TO	44
1000	TO	16	TO	17
10000	TO	13	TO	15
100000	TO	13	TO	13

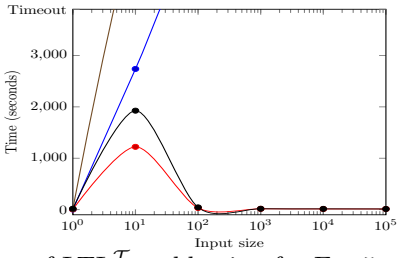

Table 4: Scalability comparison of $\text{LTL}^{\mathcal{T}}$ vs blasting for Ex. 5.

for the example, because the technique directly explores fine-grain solutions close to the boundaries and abstracts the rest. Surprisingly, in Ex. 5, the larger the bounds, the sooner our method is able to check realizability, but this should not be treated as a general result (in this example, this is because large bounds allow a safety boundary to be wider, whereas narrower bounds force the unrealizability search to be more nuances). This is a preliminary set of experiments, not part of the prototype and future work is needed to further assess applicability of $\text{LTL}^{\mathcal{T}}$ to more complex embedded system specifications.

7 Conclusions

In this paper we studied reactive synthesis and realizability for full $\text{LTL}^{\mathcal{T}}$ which includes a fetching operator that allows specifying expressive data temporal relations. We introduced CEGRES, a counter-example guided reactive synthesis method that uses a novel concept of reactive tautologies. CEGRES is guaranteed to terminate for $\text{LTL}^{\mathcal{T}}_{\not\exists}$ and to always progress for general $\text{LTL}^{\mathcal{T}}$. We showed that this algorithm is useful for specifications that require rich dynamics and we also showed that it is competitive against less expressive competing tools.

This paper opens the door for exciting future work, as tautologies are to realizability modulo theory what inductive invariants are for invariant proving. We will investigate how to discover and reuse reactive tautologies, for which we envision that TSL-MT invariant generation methods like [25,7,20] and CEGRES will mutually benefit. As for scalability, we will explore how to improve our CEGAR approach via reductions to simpler arena solving based methods internally [20], LTL methods like mode decomposition [5] or with neurosymbolic techniques like [23,24]. As for theory, finding decidable fragments [17] of realizability for $\text{LTL}^{\mathcal{T}}$ (other than $\text{LTL}^{\mathcal{T}}_{\not\exists}$ or bounded domains) is ongoing work, as well as unifying the contributions with classic reactivity approaches [27] and symbolic model checking [6,29]. Also, we will investigate how to include very recent performant tools for infinite-state reactive synthesis [2,19] under the umbrella of $\text{LTL}^{\mathcal{T}}$ specifications. Finally, we are studying applications to safety shield [3,1] synthesis beyond Booleans [44,35,9,22], where rich dynamics of the environment play an important role.

References

1. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. arXiv **abs/1708.08611** (2017). https://doi.org/10.48550/ARXIV.1708.08611
2. Azzopardi, S., Piterman, N., Stefano, L.D., Schneider, G.: Symbolic infinite-state LTL synthesis (2024). https://doi.org/10.48550/arXiv.2307.09776
3. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: Runtime enforcement for reactive systems. In: Proc. of the 21st International Conference in Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15). LNCS, vol. 9035, pp. 533–548. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_51
4. Bradley, A.R., Manna, Z.: The Calculus of Computation. Springer-Verlag (2007)
5. Brizzio, M., Gorostiaga, F., Sanchez, C., Degiovanni, R.: Mode-based reactive synthesis. In: Proc of the 17th NASA Formal Methods International Symposium (NFM'25). LNCS (2025). https://doi.org/10.1007/978-3-031-60698-4_1
6. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10ˆ20 states and beyond. In: Proc. of the 5th Annual Symposium on Logic in Computer Science (LICS '90). pp. 428–439. IEEE Computer Society (1990). https://doi.org/10.1109/LICS.1990.113767
7. Choi, W., Finkbeiner, B., Piskac, R., Santolucito, M.: Can reactive synthesis and syntax-guided synthesis be friends? In: Proc. of the 43rd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation (PLDI'22). pp. 229–243. ACM (2022). https://doi.org/10.1145/3519939.3523429
8. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. of the 12th In'l Conf. on Computer Aided Verification (CAV'2000). LNCS, vol. 1855, pp. 154–169. Springer (2000)
9. Corsi, D., Amir, G., Rodríguez, A., Katz, G., Sánchez, C., Fox, R.: Verification-guided shielding for deep reinforcement learning. RLJ **4**, 1759–1780 (2024)
10. D'Antoni, L., Veanes, M.: The power of symbolic automata and transducers. In: Proc. of the 29th Int'l Conf. in Computer Aided Verification (CAV 2017), Part I. LNCS, vol. 10426, pp. 47–67. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_3
11. Finkbeiner, B.: Synthesis of reactive systems. In: Dependable Software Systems Engineering, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-72
12. Finkbeiner, B., Heim, P., Passing, N.: Temporal stream logic modulo theories. In: Proc. of the 25th Int'l Conf. on Foundations of Software Science and Computation Structures (FOSSACS'22). LNCS, vol. 13242, pp. 325–346. Springer (2022). https://doi.org/10.1007/978-3-030-99253-8_17
13. Finkbeiner, B., Klein, F., Piskac, R., Santolucito, M.: Temporal stream logic: Synthesis beyond the Bools. In: Proc. of the 31st Int'l Conf. on Computer Aided Verification (CAV'19), Part I. LNCS, vol. 11561, pp. 609–629. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_35
14. Gacek, A., Katis, A., Whalen, M.W., Backes, J., Cofer, D.D.: Towards realizability checking of contracts using theories. In: Proc. of the 7th Int'l Symp. NASA Formal Methods (NFM'15). LNCS, vol. 9058, pp. 173–187. Springer (2015). https://doi.org/10.1007/978-3-319-17524-9_13

15. Geatti, L., Gianola, A., Gigante, N.: Linear temporal logic modulo theories over finite traces. In: Proc. of the 31st Int'l Joint Conf. on Artificial Intelligence, (IJCAI'22). pp. 2641–2647. ijcai.org (2022). https://doi.org/10.24963/ijcai.2022/366
16. Geatti, L., Gianola, A., Gigante, N.: A general automata model for first-order temporal logics. arXive **abs/2405.20057** (2024). https://doi.org/abs/2405.20057
17. Geatti, L., Gianola, A., Gigante, N., Winkler, S.: Decidable fragments of LTL_f modulo theories. In: Proc. of the 26th European Conference on Artificial Intelligence (ECAI'23). Frontiers in Artificial Intelligence and Applications, vol. 372, pp. 811–818. IOS Press (2023). https://doi.org/10.3233/FAIA230348
18. Heim, P., Dimitrova, R.: Solving infinite-state games via acceleration. Proc. ACM Program. Lang. **8**(POPL), 1696–1726 (2024). https://doi.org/10.1145/3632899
19. Heim, P., Dimitrova, R.: Issy: A comprehensive tool for specification and synthesis of infinite-state reactive systems (2025). https://doi.org/10.48550/arXiv.2502.03013
20. Heim, P., Dimitrova, R.: Translation of temporal logic for efficient infinite-state reactive synthesis. Proc. ACM Program. Lang. **9**(POPL) (2025). https://doi.org/10.1145/3704888
21. Katis, A., Fedyukovich, G., Guo, H., Gacek, A., Backes, J., Gurfinkel, A., Whalen, M.W.: Validity-guided synthesis of reactive systems from assume-guarantee contracts. In: Proc. of the 24th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'18), Part II. LNCS, vol. 10806, pp. 176–193. Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_10
22. Kim, K., Corsi, D., Rodríguez, A., Lanier, J., Parellada, B., Baldi, P., Sánchez, C., Fox, R.: Realizable continuous-space shields for safe reinforcement learning. In: Proc. of the 7th Annual Learning for Dynamics & Control Conference (L4DC'25). PMLR (2025), `https://proceedings.mlr.press/v242/zhou24a.html`
23. Kretínský, J., Meggendorfer, T., Prokop, M., Rieder, S.: Guessing winning policies in LTL synthesis by semantic learning. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification (CAV'23). LNCS, vol. 13964, pp. 390–414. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_20
24. Křetínský, J., Meggendorfer, T., Prokop, M., Zarkhah, A.: Semml: Enhancing automata-theoretic ltl synthesis with machine learning. In: Gurfinkel, A., Heule, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 233–253. Springer Nature Switzerland, Cham (2025)
25. Maderbacher, B., Bloem, R.: Reactive synthesis modulo theories using abstraction refinement. In: Proc. of the 22nd Int'l Conf. on Formal Methods in Computer-Aided Design, (FMCAD'22). pp. 315–324. IEEE (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_38
26. Maderbacher, B., Windisch, F., Bloem, R.: Synthesis from infinite-state generalized reactivity(1) specifications. In: Proc. of the 12th Int'l Symp. On Leveraging Applications of Formal Methods, Verification and Validation on Software Engineering Methodologies, (ISoLA 2024), Part IV. LNCS, vol. 15222, pp. 281–301. Springer (2024). https://doi.org/10.1007/978-3-031-75387-9_17
27. Manna, Z., Pnueli, A.: A hierarchy of temporal properties. In: Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC'90). pp. 377–410. ACM (1990). https://doi.org/10.1145/93385.93442
28. Manna, Z., Pnueli, A.: Temporal verification of reactive systems - safety. Springer (1995)
29. McMillan, K.L.: Eager abstraction for symbolic model checking. In: Proc. of the 30th International Conference in Computer Aided Verification (CAV'18), Part I.

LNCS, vol. 10981, pp. 191–208. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_11
30. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Proc. of the 30th Int'l Conf. on Computer Aided Verification (CAV'18) Part I. LNCS, vol. 10981, pp. 578–586. Springer (2018)
31. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). LNCS, vol. 4693, pp. 337–340. Springer (2008)
32. Pnueli, A.: The temporal logic of programs. In: Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS'77). pp. 46–67. IEEE CS Press (1977)
33. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. of the 16th Annual ACM Symp. on Principles of Programming Languages (POPL'89). pp. 179–190. ACM Press (1989)
34. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Proc. of the 16th Int'l Colloqium on Automata, Languages and Programming (ICALP'89). LNCS, vol. 372, pp. 652–671. Springer (1989)
35. Rodríguez, A., Amir, G., Corsi, D., Sánchez, C., Katz, G.: Shield synthesis for LTL modulo theories. In: Proc. of the 39th AAAI Conf. on Artificial Intelligence (AAAI'25). pp. 15134–15142. AAAI Press (2025). https://doi.org/10.1609/AAAI.V39I14.33660
36. Rodríguez, A., Gorostiaga, F., Sánchez, C.: Predictable and performant reactive synthesis modulo theories via functional synthesis. In: Proc. of 22nd the Int'l Symposium on Automated Technology for Verification and Analysis (ATVA'24). Springer (2024)
37. Rodríguez, A., Sánchez, C.: Boolean abstractions for realizabilty modulo theories. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification (CAV'23). LNCS, vol. 13966. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37709-9_15
38. Rodríguez, A., Sánchez, C.: Adaptive Reactive Synthesis for LTL and LTLf Modulo Theories. In: Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024). pp. 10679–10686. AAAI Press (2024). https://doi.org/10.1609/AAAI.V38I9.28939
39. Rodríguez, A., Sánchez, C.: Realizability modulo theories. Journal of Logical and Algebraic Methods in Programming **140**, 100971 (2024). https://doi.org/https://doi.org/10.1016/j.jlamp.2024.100971
40. Samuel, S., D'Souza, D., Komondoor, R.: Gensys: a scalable fixed-point engine for maximal controller synthesis over infinite state spaces. In: Proc. of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21). pp. 1585–1589. ACM (2021). https://doi.org/10.1145/3468264.3473126
41. Samuel, S., D'Souza, D., Komondoor, R.: Symbolic fixpoint algorithms for logical LTL games. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2023). pp. 698–709. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00212
42. Schmuck, A., Heim, P., Dimitrova, R., Nayak, S.P.: Localized attractor computations for infinite-state games. In: Proc. of the 36th Int'l Conf. on Computer Aided Verification (CAV'24), Part III. LNCS, vol. 14683, pp. 135–158. Springer (2024). https://doi.org/10.1007/978-3-031-65633-0_7
43. Veanes, M., Ball, T., Ebner, G., Zhuchko, E.: Symbolic automata: ω-regularity modulo theories. Proc. ACM Program. Lang. **9**(POPL) (2025). https://doi.org/10.1145/3704838

44. Wu, M., Wang, J., Deshmukh, J., Wang, C.: Shield synthesis for real: Enforcing safety in cyber-physical systems. In: Proc. of 19th Formal Methods in Computer Aided Design, (FMCAD'19). pp. 129–137. IEEE (2019). https://doi.org/10.23919/FMCAD.2019.8894264