Predictable and Performant Reactive Synthesis Modulo Theories via Functional Synthesis*

Andoni Rodríguez[1,2], Felipe Gorostiaga[1,3] and César Sánchez[1]

[1] IMDEA Software Institute, Madrid. Spain
[2] Universidad Politécnica de Madrid. Spain
[3] CIFASIS. Argentina

Abstract. Reactive synthesis is the process of generating correct controllers from temporal logic specifications. Classical LTL reactive synthesis handles (propositional) LTL as a specification language. Boolean abstractions allow reducing LTL$^{\mathcal{T}}$ specifications (i.e., LTL with propositions replaced by literals from a theory $\mathcal{T}$), into equi-realizable LTL specifications. In this paper we extend these results into a full *static* synthesis procedure. The synthesized system receives from the environment valuations of variables from a rich theory $\mathcal{T}$ and outputs valuations of system variables from $\mathcal{T}$. We use the abstraction method to synthesize a reactive Boolean controller from the LTL specification, and we combine it with functional synthesis to obtain a static controller for the original LTL$^{\mathcal{T}}$ specification. We also show that our method allows *adaptive responses* in the sense that the controller can optimize its outputs in order to e.g., always provide the smallest safe values. This is the first full static synthesis method for LTL$^{\mathcal{T}}$, which is a deterministic program (hence predictable and efficient).

1 Introduction

Reactive synthesis for Linear Temporal Logic (LTL) specifications [31] has received extensive research attention [33]. A specification φ has its propositions split into those variables controlled by the system and the rest, controlled by the environment. A specification is realizable if there is a strategy for the system that produces valuations of the system variables such that all traces generated by the controller satisfy the specification. Realizability is the decision problem of whether such a strategy for the system exists. Synthesis is the process of generating one such winning strategy. Also, both problems are decidable for LTL [31].

A recent extension of LTL called LTL$^{\mathcal{T}}$ (LTL modulo theories) allows replacing propositions with literals from a first-order theory $\mathcal{T}$. Given an LTL$^{\mathcal{T}}$ specification $\varphi^{\mathcal{T}}$ an equi-realizable LTL $\varphi^{\mathbb{B}}$ formula can be generated, provided

* This work was funded in part by PRODIGY Project (TED2021-132464B-I00) — funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR — by the DECO Project (PID2022-138072OB-I00) — funded by MCIN/AEI/10.13039/501100011033 and by the ESF, as well as by a research grant from Nomadic Labs and the Tezos Foundation.

that the validity of $\mathcal{T}$ formulae of the form $\exists^*\forall^*$ is decidable [35,37]. In LTL$^{\mathcal{T}}$ synthesis the theory variables (for example Natural o Real) in the specification are split into environment-controlled and system-controlled variables, and both kinds can appear in any given literal, whereas in LTL an atomic proposition belongs exclusively to one player.

Note that a controller obtained from an off-the-shelf synthesis procedure for the Booleanized LTL formula $\varphi^{\mathbb{B}}$ cannot be directly used as a controller for $\varphi^{\mathcal{T}}$, because it must handle rich input and output values. Previous similar approaches either (1) focus on the satisfiability problem and not in realizability (e.g., [20,22]); or (2) cannot be adapted to arbitrary $\mathcal{T}$ [27] or (3) do not guarantee termination [28,39,24], which makes these solutions incomplete. Recently, [36] presented a method for synthesis of a fragment of decidable LTL$^{\mathcal{T}}$ specifications, which relies on the use of SMT solvers on-the-fly at every reaction, so it does not produce a standalone controller. This precludes the application to real embedded systems where controllers frequently operate because (1) the SMT solver is not guaranteed to terminate (particularly with limited resources), (2) the solver may not return the same values provided the same formula (affecting *predictability*) and (3) invoking solvers on the fly has an impact on the performance and the reaction time.

In this paper we present a *static* synthesis procedure for LTL$^{\mathcal{T}}$ specifications. Our method proceeds as follows. We first obtain a Boolean controller C for the equi-realizable Boolean specification $\varphi^{\mathbb{B}}$. The controller for $\varphi^{\mathcal{T}}$ uses two additional components: a partitioner, that transforms the environment input u into the corresponding Boolean input to C and a provider that receives the input u and the reaction from C, and produces the reaction v. Our provider, instead of performing SMT calls, is implemented as a a collection of Skolem functions f that generate, given u, an output v, such that the literals in $\varphi^{\mathcal{T}}$ agree with the reaction chosen by C. Since each trace produced by our controller satisfies $\varphi^{\mathcal{T}}$, the composition of the partitioner, the Boolean controller C and the provider is a controller for $\varphi^{\mathcal{T}}$ Therefore, the procedure described here is a static synthesis procedure for specifications in LTL$^{\mathcal{T}}$. The resulting controller is standalone deterministic program, which is predictable and highly performant.

Next, we exploit the fact that we can synthesize Skolem functions that additionally receive a set of constraints to not only generate outputs that satisfy the desired literals, but that also optimize certain criteria from the set of possible correct outputs (e.g., to provide the smallest value among possible values). We call this technique *adaptivity.* All these results are applicable to LTL$^{\mathcal{T}}$ on infinite or on finite traces, using appropriate synthesis tools for the resulting $\varphi^{\mathbb{B}}$.

In summary, the contributions of this paper are: (1) a formalization and soundness proof of the controller architecture for synthesis for LTL$^{\mathcal{T}}$; (2) a derived correct method to synthetise static controllers from LTL$^{\mathcal{T}}$ specifications combining Boolean abstraction, reactive synthesis and functional synthesis; (3) a formalization of the limits and capabilities of using Skolem functions, showing their power to model adaptivity; (4) an extensive empirical evaluation that shows our method predictable and fast. To the best of our knowledge, this is the first full static reactive synthesis approach for LTL$^{\mathcal{T}}$ specifications. Moreover, since our approach leverages off-the-shelf components (reactive synthesis, functional synthesis), it would immediately benefit from advances in those areas and also from discoveries in decidable fragments of LTL$^{\mathcal{T}}$ realizability. The remainder of the paper is structured as follows. Sec. 2 contains preliminary definitions, including the Boolean abstraction method from [35] and a running example that is used in the rest of the paper. Sec. 3 formalizes the controller architecture and proves its correctness. Sec. 4 introduces an adaptive extension of our approach. Sec. 5 contains an empirical evaluation. Sec. 6 shows related work and concludes.

2 Preliminaries

First-order Theories. In this paper we use first-order theories. We describe theories with single domain for simplicity, but this can be easily extended to multiple sorts. A first-order theory $\mathcal{T}$ (see e.g., [7]) is described by a signature Σ, which consists of a finite set of functions and constants, a set of variables and a domain. The domain $\mathbb{D}$ of a theory $\mathcal{T}$ is the sort of its variables. For example, the domain of non-linear real arithmetic $\mathcal{T}_{\mathbb{R}}$ is $\mathbb{R}$ and we denote this by $\mathbb{D}(\mathcal{T}_{\mathbb{R}}) = \mathbb{R}$ or simply by $\mathbb{D}$ if it is clear from the context. A formula φ is valid in $\mathcal{T}$ if, for every interpretation I of $\mathcal{T}$, then $I \vDash \varphi$. A fragment of a theory $\mathcal{T}$ is a syntactically-restricted subset of formulae of $\mathcal{T}$. Given a formula ψ, we use $\psi[\overline{x} \leftarrow \overline{u}]$ for the substitution of variables $\overline{x}$ by terms $\overline{u}$ (typically constants).

Reactive Synthesis. We fix a finite set of atomic propositions AP. Then, $\Sigma = 2^{AP}$ is the alphabet of valuations, and Σ^* and Σ^ω are the set of finite and infinite traces respectively. Given a trace σ we use $\sigma(i)$ for the letter at position i and σ^i for the suffix trace that starts at position i. The syntax of propositional LTL [31,29] is:

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

where $a \in AP$; $\vee$, $\wedge$ and $\neg$ are the usual Boolean disjunction, conjunction and negation; and $\bigcirc$ and $\mathcal{U}$ are the next and until temporal operators. The semantics of LTL associates traces $\sigma \in \Sigma^\omega$ with LTL fomulae as follows:

$$\begin{array}{lll}
\sigma \models \top & & \text{always holds} \\
\sigma \models a & \text{iff} & a \in \sigma(0) \\
\sigma \models \varphi_1 \vee \varphi_2 & \text{iff} & \sigma \models \varphi_1 \text{ or } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \bigcirc\varphi & \text{iff} & \sigma^1 \models \varphi \\
\sigma \models \varphi_1\,\mathcal{U}\,\varphi_2 & \text{iff} & \text{for some } i \geq 0\ \ \sigma^i \models \varphi_2, \text{ and for all } 0 \leq j < i, \sigma^j \models \varphi_1
\end{array}$$

We use common derived operators like $\vee$, $\mathcal{R}$, $\diamondsuit$ and $\square$. Reactive synthesis [33,32,5,17] is the problem of automatically constructing a system based on an LTL specification φ, where the atomic propositions of φ (AP) are divided 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 $\overline{e} \cap \overline{s} = \emptyset$). A reactive specification corresponds to a turn-based game where the environment and system players alternate. In each turn, the environment produces values for $\overline{e}$, and the system responds with values for $\overline{s}$. A valuation is a map from $\overline{e}$ into $\mathbb{B}$ (similarly for $\overline{s}$). We use $\mathit{val}(\overline{e})$ and $\mathit{val}(\overline{s})$ for valuations. A play is an infinite sequence of turns and induces a trace σ by joining at each position the valuations that the environment and system players choose. The system player wins a play if the trace satisfies φ. A strategy for the system is a tuple $\rho : \langle Q, q_0, \delta, o\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the inital state, $\delta : Q \times \mathit{val}(\overline{e}) \to Q$ is the transition function and $o : Q \times \mathit{val}(\overline{e}) \to \mathit{val}(\overline{s})$ is the output function. A play $((\overline{e}_0, \overline{s}_0), (\overline{e}_1, \overline{s}_1), \ldots)$ is played according to ρ if the sequence $((\overline{e}_0, \overline{s}_0, q_0), (\overline{e}_1, \overline{s}_1, q_1), \ldots)$ satisfies that $q_{i+1} = \delta(q_i, \overline{e}_i)$ and $\overline{s}_i = o(q_i, \overline{e}_i)$ for all $i \geq 0$. A strategy ρ is wining for the system if all plays played according to ρ satisfy φ. We will use *strategy* and *controller* interchangeably.

Linear Temporal Logic Modulo Theories. The syntax of LTL$^{\mathcal{T}}$ replaces atoms a by literals l from theory $\mathcal{T}$. We use $\mathit{Vars}(l)$ for the variables in literal l and $\mathit{Vars}(\varphi)$ for the union of the variables that occur in the literals of φ. A valuation for a set of vars $\overline{z}$ is a map from $\overline{z}$ into $\mathbb{D}$. The alphabet of a formula φ is $\Sigma_{\mathcal{T}} : \mathit{Vars}(\varphi) \to \mathbb{D}$. The semantics of LTL$^{\mathcal{T}}$ associate traces $\sigma \in \Sigma_{\mathcal{T}}^{\omega}$ with formulae, where for atomic propositions $\sigma \models l$ holds iff $\sigma(0) \vDash_{\mathcal{T}} l$, that is, if the valuation $\sigma(0)$ makes the literal l true. The rest of the operators are as in LTL.

For realizability and synthesis from LTL$^{\mathcal{T}}$, the variables in $\mathit{Vars}(\varphi)$ are split into those variables controlled by the environment ($\overline{x}$ or $\mathit{Vars}_E(\varphi)$) and those controlled by the system ($\overline{y}$ or $\mathit{Vars}_E(\varphi)$). We use $\varphi(\overline{x}, \overline{y})$ to denote that $\overline{x} \cup \overline{y}$ are the variables occurring in φ (where $\overline{x} \cup \overline{y} = \mathit{Vars}(\varphi)$ and $\overline{x} \cap \overline{y} = \emptyset$). A trace is an infinite sequence of valuations of $\overline{x}$ and $\overline{y}$, which induces an infinite sequence of Boolean values for each of the literals at each position, and ultimately a valuation of φ. For instance, given $\psi = \square(y > x)$ the trace $(\langle x : 4, y : 5\rangle, \langle x : 9, y : 7\rangle, \ldots)$ induces $(\langle l : \mathit{true}\rangle, \langle l : \mathit{false}\rangle, \ldots)$ for the literal $l = (y > x)$. An LTL$^{\mathcal{T}}$ specification corresponds to a game with an infinite arena, where positions can have infinitely many successors. A strategy now for the system is a tuple $\rho^{\mathcal{T}} : \langle Q, q_0, \delta, o\rangle$ where Q and q_0 are as before and $\delta : Q \times \mathit{val}(\overline{x}) \to Q$ is the transition function and $o : Q \times \mathit{val}(\overline{x}) \to \mathit{val}(\overline{y})$ is the output function.

Boolean Abstraction. The Boolean abstraction method [35] transforms an LTL$^{\mathcal{T}}$ specification $\varphi^{\mathcal{T}}$ into an equi-realizable LTL specification $\varphi^{\mathbb{B}}$. The resulting LTL formula can be passed to an off-the-shelf synthesis engine, which generates a controller for realizable specifications. The process of Boolean abstraction involves transforming an input formula $\varphi^{\mathcal{T}}$, which contains literals l_i, into a new specification $\varphi^{\mathbb{B}} = \varphi^{\mathcal{T}}[l_i \leftarrow s_i] \wedge \varphi^{\mathit{extra}}$, where $\overline{s} = \{s_i | \text{for each } l_i\}$ is a set of fresh atomic propositions controlled by the system—such that s_i replaces l_i—and where φ^{extra} is an additional sub-formula that captures the dependencies between the $\overline{s}$ variables[4]. The formula φ^{extra} also includes additional environment variables $\overline{e}$ (controlled by the environment) that encode the power of the environment to leave the system with the power to choose certain valuations of the variables $\overline{s}$. The formula φ^{extra} also constraints the environment in such a way that exactly one of the variables in $\overline{e}$ is true.

A *choice* c is a valuation of $\overline{s}$, $c(s_i) = \mathit{true}$ means that s_i is in the choice. We write $s_i \in c$ as a synonym of $c(s_i) = \mathit{true}$. The characteristic formula $f_c(\overline{x}, \overline{y})$ of a choice c is $f_c = \bigwedge_{s_i \in c} l_i \wedge \bigwedge_{s_i \notin c} \neg l_i$. Note that we often represent choices as valuation $v_{\overline{s}}$ of the Boolean variables $\overline{s}$ (which map each variable in $\overline{s}$ to *true* or *false*). We use $\mathcal{C}$ for the set of choices (that is, the set of sets of $\overline{s}$). A *reaction* $r \subset \mathcal{C}$ is a set of choices, which characterizes the possible responses of the system as the result of a move by the environment. The *characteristic formula* $f_r(\overline{x})$ of a reaction r is:

$$(\bigwedge_{c \in r} \exists \overline{y}. f_c) \wedge (\bigwedge_{c \notin r} \forall \overline{y} \neg f_c)$$

A reaction r is valid whenever $\exists \overline{x}. f_r(\overline{x})$ is valid.

Intuitively, f_r states that for some valuations of the variables $\overline{x}$ controlled by the environment, the system can respond with valuations of $\overline{y}$ making the literals in some choice $c \in r$ but cannot respond with valuations making the literals in choices $c \notin r$. The set of valid reactions partitions precisely the moves of the environment in terms of the reaction power left to the system. For each valid reaction r there is a fresh environment variable $e \in \overline{e}$. Hence, the restriction in φ^{extra} that forces the environment to make exactly one variable in $\overline{e}$ true corresponds to the environment choosing a reaction r (when the corresponding e is true).

Boolean abstraction [35] uses the set of valid reactions to produce an equi-realizable $\varphi^{\mathbb{B}}$ from a formula $\varphi^{\mathcal{T}}$, which are in the same temporal fragment.

Example 1 (Running example). Let $\varphi^{\mathcal{T}}(\overline{x}, \overline{y})$ be the following specification (where $\overline{x} = \{x\}$ is controlled by the environment and $\overline{y} = \{y\}$ by the system):

$$\varphi^{\mathcal{T}} = \square\big[\big((x < 2) \rightarrow \bigcirc(y > 1)\big) \wedge \big((x \geq 2) \rightarrow (y \leq x)\big)\big].$$

In theory $\mathcal{T}_{\mathbb{Z}}$ this specification is realizable (consider the strategy to always play $y = 2$). In this theory, the Boolean abstraction first introduces s_0 to abstract $(x < 2)$, s_1 to abstract $(y > 1)$ and s_2 to abstract $(y \leq x)$. Then $\varphi^{\mathbb{B}} = \varphi'' \wedge \square(\varphi^{legal} \rightarrow \varphi^{extra})$ where $\varphi'' = (s_0 \rightarrow \bigcirc s_1) \wedge (\neg s_0 \rightarrow s_2)$ is a direct abstraction of $\varphi^{\mathcal{T}}$. Finally, φ^{extra} captures the depenencies between the abstracted variables:

$$\varphi^{extra} : \left(\begin{array}{l} \big(e_0 \rightarrow \big([\ \ s_0 \wedge s_1 \wedge \neg s_2] \vee [\ \ s_0 \wedge \neg s_1 \wedge s_2]\big) \\ \wedge \big(e_1 \rightarrow \big([\neg s_0 \wedge s_1 \wedge \ \ s_2] \vee [\neg s_0 \wedge \neg s_1 \wedge \ \ s_2] \vee [\neg s_0 \wedge \ \ s_1 \wedge \neg s_2]\big) \end{array} \right)$$

and $\varphi^{legal} : (e_0 \vee e_1) \wedge (e_0 \leftrightarrow \neg e_1)$, where $\overline{e} = \{e_0, e_1\}$ belong to the environment and represent $(x < 2)$ and $(x \geq 2)$, respectively. Thus, φ^{legal} encodes that e_0 and

[4] The Boolean abstraction process can substitute larger sub-formulae than literals (as long as they do not contain temporal operators).

e_1 characterize a partition of the (infinite) input valuations of the environment (and that precisely one of $\overline{e}$ are true in every move). For example, the valuation $v_e = \langle e_0 : \mathit{false}, e_1 : \mathit{true}\rangle$ of $\overline{e}$ corresponds to the choice of the environment where only e_1 is true. Sub-formulae like $(s_0 \wedge s_1 \wedge \neg s_2)$ represent the *choices* of the system (in this case, $c = \{s_0, s_1\}$), that is, given a decision of the environment (a valuation of $\overline{e}$ that makes exactly one variable e true), the system can *react* with one of the choices c in the disjunction implied by e. We denote $c_0 = \{s_0, s_1, s_2\}$, $c_1 = \{s_0, s_1\}$, $c_2 = \{s_0, s_2\}$, $c_3 = \{s_0\}$, $c_4 = \{s_1, s_2\}$, $c_5 = \{s_1\}$, $c_6 = \{s_2\}$ and $c_7 = \emptyset$. Note that e.g., c_1 can be represented as $v_{\overline{s}} = \langle s_0 : \mathit{true}, s_1 : \mathit{true}, s_2 : \mathit{false}\rangle$.

3 Static Reactive Synthesis Modulo Theories

The Boolean abstraction method [35] reduces an LTL$^{\mathcal{T}}$ formula $\varphi^{\mathcal{T}}$ into an equi-realizable LTL specification $\varphi^{\mathbb{B}}$, but it does not present a synthesis procedure for $\varphi^{\mathcal{T}}$. We solve this problem here by providing a full static synthesis method for LTL$^{\mathcal{T}}$. Our procedure builds a controller for realizable $\varphi^{\mathcal{T}}$ specifications that handles inputs and outputs from a rich domain $\mathbb{D}(\mathcal{T})$, using as a building block the synthetized Boolean controller for $\varphi^{\mathbb{B}}$ and other two sub-components.

3.1 Formal Architecture

We call our approach static LTL$^{\mathcal{T}}$ synthesis (see Fig. 1). Our method starts from $\varphi^{\mathcal{T}}(\overline{x}, \overline{y})$ and statically generates a Boolean controller $\rho^{\mathbb{B}}$ for $\varphi^{\mathbb{B}}$ and combines it with a partitioner and a provider (generated from the abstraction process of $\varphi^{\mathcal{T}}$ to $\varphi^{\mathbb{B}}$) handle the inputs and outputs from $\mathbb{D}$. At run-time, at each instant the resulting controller follows these steps:
(1) a valuation $v_{\overline{x}} \in val(\overline{x})$ is provided by the environment;
(2) the partitioner discretizes $v_{\overline{x}}$ generating a Boolean valuation $v_{\overline{e}} \in val(\overline{e})$ of input variables for $\rho^{\mathbb{B}}$.
(3) $\rho^{\mathbb{B}}$ responds with a valuation $v_{\overline{s}} \in val(\overline{s})$ of the variables $\overline{s}$ that $\rho^{\mathbb{B}}$ controls.

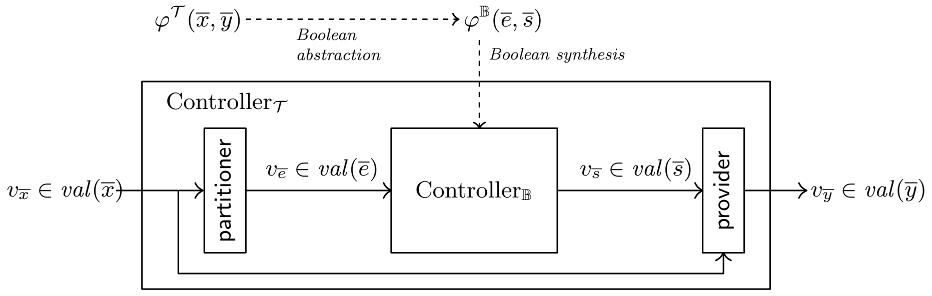
$\varphi^{\mathcal{T}}(\overline{x}, \overline{y})$
Boolean abstraction
$\varphi^{\mathbb{B}}(\overline{e}, \overline{s})$
Boolean synthesis
Controller$_{\mathcal{T}}$
$v_{\overline{x}} \in val(\overline{x})$
partitioner
$v_{\overline{e}} \in val(\overline{e})$
Controller$_{\mathbb{B}}$
$v_{\overline{s}} \in val(\overline{s})$
provider
$v_{\overline{y}} \in val(\overline{y})$

Fig. 1: A controller architecture for reactive synthesis of LTL$^{\mathcal{T}}$ specifications.

(4) the provider produces a valuation $v_{\overline{y}} \in val(\overline{y})$ of the output variables that together with $v_{\overline{x}}$ guarantee that the literals from $\varphi^{\mathcal{T}}$ will be evaluated as indicated by the choice c that corresponds to $v_{\overline{s}}$ indicated by $\rho^{\mathbb{B}}$. This step corresponds to finding a model of $\exists\overline{y}.f_c([\overline{x} \leftarrow v_{\overline{x}}], \overline{y})$.

For step (4) one approach is to invoke an SMT solver on the fly to generate models (proper values of $v_{\overline{y}}$), which is guaranteed to be satisfiable (by the soundness of the Boolean abstraction method). However, most uses of controllers cannot use SMT solvers dynamically. Moreover, note that the formula to be solved has quantifier alternations (within f_c) which is currently challenging for state-of-the art SMT solving technology for many theories. In this paper we present an alternative: a method that produces a totally static controller, using Skolem functions associated to each (e, c) pair. These Skolem functions are models of the formula

$$\forall\overline{x}.\exists\overline{y}.f_r(\overline{x}) \to f_c(\overline{x}, \overline{y})$$

Recall that $f_r(\overline{x})$ is the formula that characterizes the environment valuations for which r captures the possible responses after receiving $\overline{x}$, according to reasoning in the theory $\mathcal{T}$.

Partitioner. At each timestep, the partitioner receives a valuation $v_{\overline{x}} \in val(\overline{x})$ of the environment variables and finds the input variable e_k to be fed to the Boolean controller. The partitioner must find the entry (e, r) in the table of valid reactions for which $f_r(v_{\overline{x}})$ is valid and return e. For instance, recall partitions e_0 and e_1 from Ex. 1, then an input trace $(\langle x : 4\rangle, \langle x : 4\rangle, \langle x : 1\rangle, \langle x : 0\rangle, \langle x : 2\rangle, \ldots)$ will be partitioned into $(\langle e : e_1\rangle, \langle e : e_1\rangle, \langle e : e_0\rangle, \langle e : e_0\rangle, \langle e : e_1\rangle, \ldots)$ (for simplicity, here we show the only Boolean variable e_i that is true). The following defines a legal partitioner.

Definition 1. *Let $\varphi^{\mathcal{T}}(\overline{x}, \overline{y})$ be an LTL$^{\mathcal{T}}$ specification and $\varphi^{\mathbb{B}}(\overline{e}, \overline{s})$ its Boolean abstraction. A* partitioner *is a function $\alpha : val(\overline{x}) \to \overline{e}$ such that if (e, r) is a valid reaction and $f_r[\overline{x} \leftarrow v_{\overline{x}}]$ is valid, then $\alpha(v_{\overline{x}}) = e$.*

Note that, by the soundness of the Boolean abstraction method, there is one and only one such candidate e for every input $v_{\overline{x}}$. Note that $\alpha(v_{\overline{x}}) = e$ induces a valuation $v_{\overline{e}}$ of the variables $\overline{e}$ by $v_{\overline{x}}(e) = true$ and $v_{\overline{x}}(e') = false$ for all other $e' \neq e$. Alg. 1 shows a brute force method to find variable e.

Alg. 1: A brute force partitioner α.

```
Input: $v_{\overline{x}} \in val(\overline{x})$
forall $(e, r) \in VR(\varphi)$ do
    if $f_r[\overline{x} \leftarrow v_{\overline{x}}]$ is valid then
        return $e$
```

Controller. The Boolean controller receives the discrete environment input $v_{\overline{e}}$ and produces a discrete output $v_{\overline{s}}$ that represents the selected choices according to a winning strategy for $\varphi^{\mathbb{B}}$. This controller $\rho^{\mathbb{B}}$ can obtained using off-the-shelf reactive synthesis tools. This controller $\rho^{\mathbb{B}}$ produces a valuation $v_{\overline{s}} \in val(\overline{s})$ at every step, guaranteeing that the trace produced satisfies $\varphi^{\mathbb{B}}$.

Consider an instant where only e is true in the input $v_{\overline{e}}$, and let r be the valid reaction corresponding to e; then if $v_{\overline{s}}$ is the output produced by $\rho^{\mathbb{B}}$ from $v_{\overline{e}}$ the choice $c : \{s_i | v_{\overline{s}}(s_i) = \mathit{true}\}$ belongs to $c \in r$. This is forced by the φ^{extra} constraint in the construction of $\varphi^{\mathbb{B}}$ from $\varphi^{\mathcal{T}}$ in the Boolean abstraction method. To better illustrate this, recall Ex. 1 and among the possible winning strategies that the system has, consider the following. If $v_{\overline{e}}$ is $\langle e_0 : \mathit{true}, e_1 : \mathit{false}\rangle$, then the output choice $v_{\overline{s}}$ is $\langle s_0 : \mathit{true}, s_1 : \mathit{true}, s_2 : \mathit{false}\rangle$ (i.e., c_1). On the other hand, if $v_{\overline{e}}(e_1)$ is $\langle e_0 : \mathit{false}, e_1 : \mathit{true}\rangle$ then output choice $v_{\overline{s}}$ is $\langle s_0 : \mathit{false}, s_1 : \mathit{true}, s_2 : \mathit{true}\rangle$ (i.e., c_4).

Provider. The discrete behavior of the Boolean controller requires an additional component that produces a valuation $v_{\overline{y}} \in \mathit{val}(\overline{y})$ of the system variables over $\overline{y}$ satisfying $\varphi^{\mathcal{T}}$. The provider receives the choice and the input $v_{\overline{x}} \in \mathit{val}(\overline{x})$, and substitutes $v_{\overline{x}}$ for $\overline{x}$ in f_c: $f_c([\overline{x} \leftarrow v_{\overline{x}}], \overline{y})$. The goal of the provider is to find a proper valuation for $\overline{y}$.

Definition 2 (Provider). *A provider is a function $\beta : \mathit{val}(\overline{x}) \times \mathit{val}(\overline{s}) \to \mathit{val}(\overline{y})$ such that for every $v_{\overline{x}} \in \mathit{val}(\overline{x})$ and choice $c \in \mathit{val}(\overline{s})$, the following holds*

$$f_c(\overline{x} \leftarrow v_{\overline{x}}, \overline{y} \leftarrow \beta(v_{\overline{x}}, c)).$$

We will show below that if $v_{\overline{x}}$ is an input to a partitioner, r is the valid corresponding reaction, and c is one of the winning choices of the controller (that is, $c \in r$), then the following formula is valid.

$$[\exists \overline{y}. f_c(\overline{y}, \overline{x} \leftarrow v_{\overline{x}})]$$

This formula can be discharged into a solver with capabilities to produce a model $v_{\overline{y}}$ (e.g., an SMT solver like Z3 [13]), which is exactly the **dynamic** approach presented at [36].

Example 2. Consider again Ex. 1 and input trace $(\langle x : 4\rangle, \langle x : 4\rangle, \langle x : 1\rangle, \langle x : 0\rangle, \langle x : 2\rangle, \ldots)$. This trace is mapped into the following discrete input trace $(\langle c : c_4\rangle, \langle c : c_4\rangle, \langle c : c_1\rangle, \langle c : c_1\rangle, \langle c : c_4\rangle, \ldots)$. Recall that s_0 abstracts $(x < 2)$, s_1 abstracts $(y > 1)$ and s_2 abstracts $(y \leq x)$. Then, the output trace must be a sequence $v_{\overline{y}}$ of values of y such that the following holds:

$$\begin{array}{l}
([\neg(4 < 2) \wedge (y > 1) \wedge (y \leq 4)], \\
[\neg(4 < 2) \wedge (y > 1) \wedge (y \leq 4)], \\
[(1 < 2) \wedge (y > 1) \wedge \neg(y \leq 1)], \\
[(0 < 2) \wedge (y > 1) \wedge \neg(y \leq 0)], \\
[\neg(2 < 2) \wedge (y > 1) \wedge (y \leq 2)], \ldots)
\end{array}$$

One such possible sequence is $(\langle y : 2\rangle, \langle y : 2\rangle, \langle y : 2\rangle, \langle y : 2\rangle, \langle y : 2\rangle, \ldots)$. Note how $\overline{x}$ is replaced in each timestep by concrete input $v_{\overline{x}}$. However, many different values $v_{\overline{y}}$ exist that satisfy the output trace (e.g. $(\langle y : 2\rangle, \langle y : 3\rangle, \langle y : 3\rangle, \langle y : 4\rangle, \langle y : 2\rangle, \ldots)$.)

Correctness. The Boolean system strategy $\rho^{\mathbb{B}} : \langle Q, q_0, \delta, o\rangle$ for $\varphi^{\mathbb{B}}$ produces, at every timestep, a valuation of $\overline{s}$ from a valuation of $\overline{e}$. We now define a strategy $\rho^{\mathcal{T}}$ of the system in $\varphi^{\mathcal{T}}$ and prove that all moves played according to $\rho^{\mathcal{T}}$ are winning for the system; i.e., all produced traces satisfy $\varphi^{\mathcal{T}}$. Intuitively, $\rho^{\mathcal{T}}$ composes the partitioner, which translates inputs to the Boolean controller, collects the move chosen by the Boolean controller and then uses the provider to generate an output.

Definition 3 (Combined Strategy). *Given a* partitioner α*, a controller* $\rho^{\mathbb{B}}$ *for* $\varphi^{\mathbb{B}}$ *and a* provider β*, the strategy* $\rho^{\mathcal{T}} : \langle Q', q'_0, \delta', o'\rangle$ *for* $\varphi^{\mathcal{T}}$ *is:*
- $Q' = Q$ *and* $q'_0 = q_0$*,*
- $\delta'(q, v_{\overline{x}}) = \delta(q, v_{\overline{e}})$ *where* $v_{\overline{e}} = \alpha(v_{\overline{x}})$*,*
- $o'(q, v_{\overline{x}}) = \beta(v_{\overline{x}}, c)$ *where* $c = o(q, v_{\overline{x}})$*.*

We use $C(\alpha, \rho^{\mathbb{B}}, \beta)$ for the combined strategy of α, $\rho^{\mathbb{B}}$ and β. Now we are ready to state the main theorem.

Theorem 1 (Correctness of Synthesis Modulo Theories). *Let* $\varphi^{\mathcal{T}}$ *be a realizable specification,* $\varphi^{\mathbb{B}}$ *its Boolean abstraction,* α *a* partitioner *and* β *a* provider*. Let* $\rho^{\mathbb{B}}$ *be a winning strategy for* $\varphi^{\mathbb{B}}$*, and let* $\rho^{\mathcal{T}} = C(\alpha, \rho^{\mathbb{B}}, \beta)$ *be the combined strategy. Then* $\rho^{\mathcal{T}}$ *is winning for* $\varphi^{\mathcal{T}}$*.*

Proof. (Sketch). Let $\rho^{\mathbb{B}} : \langle Q, q_0, \delta, o\rangle$ and $\rho^{\mathcal{T}} : \langle Q, q_0, \delta', o'\rangle$ be the strategies. Let $\pi = ((\overline{x}_0, \overline{y}_0, q_0), (\overline{x}_1, \overline{y}_1, q_1), \ldots)$ be an infinite sequence played according to $\rho^{\mathcal{T}}$, that is $\overline{y}_i = o'(q_i, \overline{x}_i)$ and $q_{i+1} = \delta'(q_i, \overline{x}_i)$. Consider the sequence $((\overline{e}_0, \overline{s}_0, q_0),$ $(\overline{e}_1, \overline{s}_1, q_1), \ldots)$ such that $\overline{e}_i = \alpha(\overline{x}_i)$, $\overline{s}_i = o(q_i, \overline{e}_i)$ and $q_{i+1} = \delta(q_i, \overline{e}_i)$. Note that this a play of $\varphi^{\mathbb{B}}$ played according to $\rho^{\mathbb{B}}$ so it satisfies $\varphi^{\mathbb{B}}$. In particular, it satisfies φ^{extra}. Moreover, for every time instant i, $\overline{y}_i = \beta(\overline{x}_i, \overline{s}_i)$ by construction. It follows that, for every i, every literal l_i in $\varphi^{\mathcal{T}}$ and the corresponding s_i in $\varphi^{\mathbb{B}}$ have the same valuation. By structural induction, all corresponding sub-formulae of $\varphi^{\mathbb{B}}$ and $\varphi^{\mathcal{T}}$ have the same valuation at every position. Therefore, $\pi \models \varphi^{\mathcal{T}}$. □

3.2 Standalone Synthesis Modulo Theories

Static Provider. As stated above, a provider produces, at every step, a model of a (satisfiable) formula (where some of the elements in the formula are the inputs received at that specific step). This can be implemented using an SMT solver at every step. In this paper we propose an alternative approach where we produce at static time a provider via the functional synthesis of a Skolem function. The controller then invokes the function produced instead of using dynamic queries to an SMT solver. Given an arbitrary relation $R(x, y)$ a Skolem function is a function $\mathtt{h}$ that witnesses the validity of $\forall x.\exists y.R(x, y)$ by guaranteeing that $\forall x..R(x, \mathtt{h}(x))$ is valid. Recall that a correct provider is a function $\beta : val(\overline{x}) \times val(\overline{s}) \rightarrow val(\overline{y})$ which is a witness of the validity of the following formula:

$$\forall \overline{x}.\exists \overline{y}.f_r(\overline{x}) \rightarrow f_c(\overline{x}, \overline{y})$$

for a given reaction r and choice $c \in r$. A Skolem function for c is a function $\mathtt{h}_c : val(\overline{x}) \rightarrow val(\overline{y})$ such that the following is valid:

$$\forall \overline{x}. f_r(\overline{x}) \rightarrow f_c(\overline{x}, h_c(\overline{x}))$$

For instance, consider a specification where the environment controls an integer variable x and the system controls an integer variable y in the specification $\varphi^{\mathcal{T}} = \square(y > x)$. A Skolem function $\mathtt{h}(x) = x + 1$ serves as a witness (providing values for y) of the validity of $\forall x. \exists y. (\top \rightarrow (y > x))$ and can be used to provide correct integer values for y. For many theories, Skolem functions for β can be statically computed, which means that we can generate statically a provider for these theories, and in turn, a full static controller for $\varphi^{\mathcal{T}}$. In this paper, we used the `AEval` funtional synthesis tool [16], which generates witnessing Skolem functions for (possibly many) existentially-quantified variables; i.e., `AEval` will output a function for every existentially quantified variable: e.g., $\forall x \exists y, z.(y > x) \wedge (z > y)$ results in $\mathtt{h}(x) = x + 1$ for y and $\mathtt{h}(x) = x + 2$ for z.

Example 3. Consider the strategy Ex. 2 where the input e_1 : *true* is mapped to c_4 and e_0 : *true* is mapped to c_1. The first case requires to synthetize the Skolem function for:

$$\forall x. \exists y. (x \geq 2) \rightarrow [(x \geq 2) \wedge (y > 1) \wedge (y \leq x)])$$

whereas the second case requires to handle:

$$\forall x. \exists y. (x < 2) \rightarrow [(x < 2) \wedge (y > 1) \wedge (y > x)]),$$

The corresponding invocations to `AEval` produce the following:

$$\mathtt{h}_{(e_1,c_4)} = \begin{cases} 2 & \text{if } (x \geq 2) \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \mathtt{h}_{(e_0,c_1)} = \begin{cases} 0 & \text{if } (x \geq 2) \\ x+1 & \text{elif } (1 < x) \\ 2 & \text{otherwise} \end{cases}$$

where we can see that, when the function $\mathtt{h}_{(e_1,c_4)}$ is called, (when e_1 holds) only the if branch will hold and will always return 2. Similarly, $\mathtt{h}_{(e_0,c_1)}$ is called when e_0 holds, so only the else branch will hold since in $\mathbb{D}(\mathcal{T}_{\mathbb{Z}})$ it never happens that $(x > 1)$ and $(x < 2)$ at the same time. Hence, invocation will always return 2.

Predictability. Ex. 2 shows that there may exist many different valuations $v_{\overline{y}}$ such that $v_{\overline{y}}$ matches the Boolean output trace with values in $\mathcal{T}$. The fact that there are many possible outputs that satisfy the same literals for a given input opens the opportunity to synthesize a controller for $\varphi^{\mathcal{T}}$ by adding additional constraints to optimize certain criteria; e.g., return the greatest value for $\overline{y}$ possible (we latter study this *adaptivity* in Sec. 4).

However, since there are many possible outputs that satisfy the same literals for a given input, using SMT solvers on-the-fly for providing such outputs does not guarantee that for the same input to the solver, the same output will be produced. In practise, different solvers (or even the same solver) can internally perform different calculations and construct different models of the same formula even for the same invocation. This means that the dynamic solver-based approach of [36] does not guarantee a provider as a function (as we present here) but instead it can be non-deterministic: different invocations with the same input to satisfy the same literals can produce different outputs. In other words, the program that implements β can be non-deterministic. Instead, in our approach with Skolem functions, β is a mathematical function from $\mathbb{B}$ to $\mathbb{D}(\mathcal{T})$ and thus guarantee **predictability** in the following sense.

Theorem 2 (Predictability). *Let $\varphi^{\mathcal{T}}$ be a realizable specification, $\varphi^{\mathbb{B}}$ its Boolean abstraction, α a* partitioner *and β a static* provider*. Let $\rho^{\mathbb{B}}$ be a winning strategy for $\varphi^{\mathbb{B}}$, and $\rho^{\mathcal{T}} = C(\alpha, \rho^{\mathbb{B}}, \beta)$ be the composed strategy Then, given two input traces $\pi_{\overline{x}}$ and $\pi'_{\overline{x}}$ such that $\pi_{\overline{x}} = \pi'_{\overline{x}}$, $\rho^{\mathcal{T}}$ will produce two output traces $\pi_{\overline{y}}$ and $\pi'_{\overline{y}}$ such that $\pi_{\overline{y}} = \pi'_{\overline{y}}$.*

The theorem follows immediately by β being a mathematical function. In the next section we extend β to provide different outputs for the same input by explicitly adding arguments to this function, but first we show the complete running example.

3.3 Complete running example

In $\varphi^{\mathcal{T}}$ of Ex. 1 a valid (positional) strategy of the system is to always play $y : 2$. In this appendix we show that a controller synthetised using our technique will, precisely, respond in this manner infinitely many often. To do so, we rely on the trace of Ex. 2 and Skolem functions of Ex. 3.

First, we Booleanize $\varphi^{\mathcal{T}}$ using [35] and get $\varphi^{\mathbb{B}}$ (also, recall from Ex. 1 that we use the notation c_i to indicate choice i; e.g., $c_0 = \{s_0, s_1, s_2\}$, $c_1 = \{s_0, s_1\}$, ..., $c_6 = \{s_2\}$, $c_7 = \emptyset$.). Then, we get a controller $C_{\mathbb{B}}$ from $\varphi^{\mathbb{B}}$. We note that many strategies satisfy $\varphi^{\mathbb{B}}$, but $C_{\mathbb{B}}$ by Strix is as follows: $C_{\mathbb{B}}(e_1) = c_4$ and $C_{\mathbb{B}}(e_0) = c_1$. Also, note that this particular strategy is memoryless, but there are diverse strategies that use memory. We now show how the static $\mathcal{T}$-controller computes Skolem functions on demand.

Step 1: Environment forces instant response. Let $x : 4$, which holds $(x \geq 2)$ and forces constraint $(y \leq x)$. We are in partition e_1, which implies choices $\{c_4, c_5, c_6\}$. Now, $C_{\mathbb{B}}(e_1) = c_4$, so the $\mathcal{T}$-controller looks whether the pair (e_1, c_1) appeared before. Since it did not, it computes $\mathtt{h}_{(e_1,c_4)}$ (see left-hand function at Ex. 3). Thus, $\mathtt{h}_{(e_1,c_4)}(2) = 2$ is the output v_y in the first timestep. Note that a $\mathcal{T}$-controller with a different underlying $C_{\mathbb{B}}$ could also consider c_6 in the current play.

Step 2: Environment repeats the strategy. Again, $x : 4$ and again we are in partition e_1. Now, $C_{\mathbb{B}}(e_1) = c_4$, so the $\mathcal{T}$-controller looks whether the pair (e_1, c_4) appeared before. Since it does, it just calls pre-computed $\mathtt{h}_{(e_1,c_4)}$. Thus, $\mathtt{h}_{(e_1,c_4)}(2) = 2$ is the output v_y in the second timestep.

Step 3: Environment changes its mind. Let $x : 1$, which holds $(x < 2)$ and forces constraint $\bigcirc(y > 1)$, whereas no constraint is further for the current timestep. We are in partition e_0, which implies choices $\{c_1, c_2\}$. Now, $C_{\mathbb{B}}(e_0) = c_1$, so the $\mathcal{T}$-controller looks whether the pair (e_0, c_1) appeared before. Since it did not, it computes $\mathtt{h}_{(e_0,c_1)}$ (see right-hand function at Ex. 3). Thus, $\mathtt{h}_{(e_0,c_1)}(2) = 2$ is the output v_y in the third timestep. Note that a $\mathcal{T}$-controller with a different underlying $C_{\mathbb{B}}$ could also consider c_2 in the current play.

Step 4: Environment prepares its trap. Let $x : 0$, which holds $(x < 2)$ and forces constraint $\bigcirc(y > 1)$, and take into account that the system has constraint $(y > 1)$ forced by the previous timestep. We are again in partition e_0, which implies, again, choices $\{c_1, c_2\}$. Now, $C_{\mathbb{B}}(e_0) = c_1$, so the $\mathcal{T}$-controller looks whether the pair (e_0, c_1) appeared before. Since it does, it just calls pre-computed $\mathtt{h}_{(e_0,c_1)}$. Thus, $\mathtt{h}_{(e_0,c_1)}(2) = 2$ is the output v_y in the fourth timestep. Note that this time there is no correct $C_{\mathbb{B}}$ that could also consider c_2 in the current play.

Step 5: Environment strikes back! Let $x : 2$, which holds $(x \geq 2)$ and forces constraint $(y \leq x)$. Also, note that the system has constraint $(y > 1)$ from previous timestep. We are in partition e_1, which implies choices $\{c_4, c_5, c_6\}$. Now, the same as in step 2 happens: $C_{\mathbb{B}}(e_1) = c_4$, so the $\mathcal{T}$-controller looks whether the pair (e_1, c_4) appeared before. Since it does, it just calls pre-computed $\mathtt{h}_{(e_1,c_4)}$. Thus, $\mathtt{h}_{(e_1,c_4)}(2) = 2$ is the output v_y in the fifth timestep.

Note that this is the dangerous situation, where the system can only output $y : 2$; in other words, it happens again that there is no correct $C_{\mathbb{B}}$ that could also consider another choice (in this case, c_4 and c_5) in the current play. We can derive all the possible behaviours from these steps. Also, note that another possibility is to pre-compute all the Skolem functions, but it is less efficient.

4 Adaptive Synthesis Modulo Theories

4.1 Enhancing Controllers

The static partitioner presented in the previous section always generates the same output, for a given choice (valuation of literals) and input. However, it is often possible that many different values can be chosen to satisfy the same choice. From the point of view of the Boolean controller, any value is indistinguishable, but from the point of view of the real-world controller the difference may be significant. For example, in the theory of linear natural arithmetic $\mathcal{T} = \mathcal{T}_{\mathbb{N}}$, given $x = 3$ and the literal $(y > x)$, a Skolem function $\mathtt{h}(x) = x + 1$ would generate $y = 4$, but $y = 5$ or $y = 6$ are also admissible. We call *adaptivity* to the ability of a controller to produce different values depending on external criteria, while still guaranteeing the correctness of the controller (in the sense that values chosen guarantee the specification). We introduce in this section a *static adaptive* provider that exploits this observation. Recall that the Skolem functions in Sec. 3 are synthetised as follows.

Definition 4 (Basic Provider Formula). *A basic provider formula is a formula of the form* $\forall \overline{x}.\exists \overline{y}.\psi(\overline{x}, \overline{y})$*, where* $\psi = f_{r_k(\overline{x})} \rightarrow f_c(\overline{x}, \overline{y})$ *is the characteristic formula for reaction* r_k *and choice* c*.*

We now introduce additional constraints to ψ that—in the case that the resulting formulae are valid—allow generating functions that guarantee further properties. Given a formula $\psi(\overline{x}, \overline{y})$ and a set of variables $\overline{z}$ (different than $\overline{x}$ and $\overline{y}$) adaptive formulae also enforce an additional constraint ψ^+.

Definition 5 (Adaptive Provider Formula). *Let* $\psi(\overline{x}, \overline{y})$ *be the characteristic formula for a given reaction* r_k *and choice* c*. An* adaptive constraint *is a formula* $\psi^+(\overline{x}, \overline{z}, \overline{y})$ *whose only free variables are* $\overline{x}$*,* $\overline{y}$ *and* $\overline{z}$*. An* adaptive provider formula *is of the form*

$$\forall \overline{x}, \overline{z}.\exists \overline{y}.[\psi(\overline{x}, \overline{y}) \wedge \psi^+(\overline{x}, \overline{z}, \overline{y})],$$

where ψ^+ *is an adaptive constraint.*

Note that, in particular, ψ^+ can use quantification. For example, in an arithmetic theory, the additional constraint $\psi^+ : \forall w.\psi(x, w) \rightarrow (|y-z| \leq |w-z|)$ states that all output alternatives w are farther to z than the y to be computed. This formula is constraining the y that must be computed. The following result guarantees the correctness of using adaptive provider formulae to craft a provider.

Lemma 1. *Let* $\forall \overline{x}, \overline{z}.\exists \overline{y}.(\psi \wedge \psi^+)$ *be a (valid) adaptive provider formula and let* f *be a Skolem function for it. Let* $v_{\overline{x}} \in val(\overline{x})$ *and* $v_{\overline{z}} \in val(\overline{z})$ *be arbitrary values. Then* $\psi(\overline{x} \leftarrow v_{\overline{x}}, \overline{y} \leftarrow f(v_{\overline{x}}, v_{\overline{z}}))$ *is true.*

Lemma 1 shows that synthesizing a Skolem function for an adaptive formula can be easily transformed into an Skolem function for the original characteristic formula, so providers that use adaptive formulae are sound with the original specification.

Example 4. Consider a basic provider formula $\psi : \forall x.\exists y.(y > x)$ in $\mathcal{T}_{\mathbb{N}}$ and the Skolem function $\mathtt{h}(x) = x + 1$ generated by `AEval`. Consider now the constraint $\psi^+ = (y \geq z) \wedge (y \geq 100)$. The adaptive formula $\forall x.\forall z.\exists y.(y > x) \wedge (y \geq z) \wedge (y \geq 100)$ is valid and one possible Skolem function is $\mathtt{h}(x) = \max(x+1, z, 100)$. However, if one considers the constraint $\psi_2^+ = (y < 100)$, the resulting provider formula is not valid and there is no Skolem function. Then, the engineer would have to provide a different constraint or use the basic provider formula.

Note that considering different constraints will produce different Skolem functions without the need of re-synthesizing a different controller, we only need to switch externally between functions.

An *adaptive provider description* is a set $\Gamma = \{\ldots \psi^+_{(r_x,c)} \ldots\}$ of constraint that contains one constraint per pair (r_k, c)—for which r_k is a valid reaction

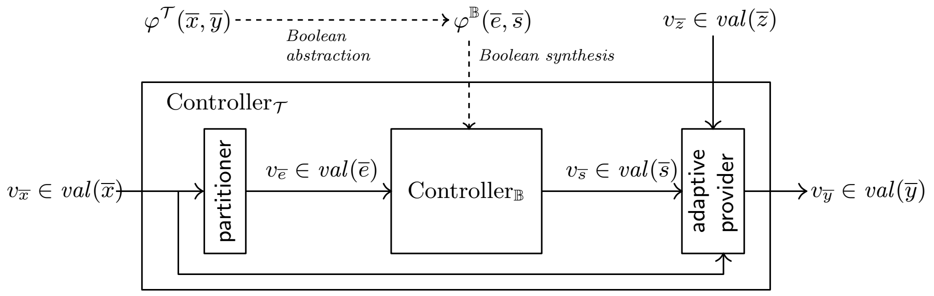
$\varphi^{\mathcal{T}}(\overline{x},\overline{y})$
Boolean abstraction
$\varphi^{\mathbb{B}}(\overline{e},\overline{s})$
Boolean synthesis
$v_{\overline{z}} \in val(\overline{z})$
Controller$_{\mathcal{T}}$
$v_{\overline{x}} \in val(\overline{x})$
partitioner
$v_{\overline{e}} \in val(\overline{e})$
Controller$_{\mathbb{B}}$
$v_{\overline{s}} \in val(\overline{s})$
adaptive provider
$v_{\overline{y}} \in val(\overline{y})$

Fig. 2: Adaptive architecture, which uses adaptive providers and $v_{\overline{z}}$.

and c a choice of r_k—, such that for every (r_k, c), the adaptive provider formula $\forall \overline{x}, \overline{z}. \exists \overline{y}. \psi \wedge \psi^{+}_{(r_x,c)}$ is valid. For example, $\psi^{+}_{(r_k,c)} = true$ for every (r_k, c) corresponds to the basic provider.

Definition 6 (Adaptive Provider). *Let Γ be an adaptive provider description. An adaptive* provider *is a function $\beta_\Gamma : val(\overline{x}) \times val(\overline{z}) \times (\overline{s}) \rightarrow val(\overline{y})$ such that for every $v_{\overline{x}} \in val(\overline{x})$, $v_{\overline{z}} \in val(\overline{z})$ and a choice $c \in val(\overline{s})$ the following holds:*

$$f_{(r_k,c)}(\overline{x} \leftarrow v_{\overline{x}}, \overline{y} \leftarrow \beta(v_{\overline{x}}, v_{\overline{z}}, c))$$

Note that given an adaptive provider description, an adaptive provider always exists, and is given by any Skolem function for each pair (r_k, c).

Definition 7 (Combined Adaptive Strategy). *Let $\varphi^{\mathcal{T}}$ be an $\text{LTL}^{\mathcal{T}}$ specification and Γ be an adaptive provider description. Given a* partitioner *α for $\varphi^{\mathcal{T}}$, a controller $\rho^{\mathbb{B}}$ for $\varphi^{\mathbb{B}}$ and an adaptive* provider *β_Γ, the strategy $\rho^{\mathcal{T}}_{\Gamma} : \langle Q', q'_0, \delta', o' \rangle$ for $\varphi^{\mathcal{T}}$ is:*
- *$Q' = Q$ and $q'_0 = q_0$,*
- *$\delta'(q, (v_{\overline{x}} \cup v_{\overline{z}})) = \delta(q, \overline{e})$ where $\overline{e} = \alpha(v_{\overline{x}})$,*
- *$o'(q, (v_{\overline{x}} \cup v_{\overline{z}})) = \beta_\Gamma(v_{\overline{x}}, v_{\overline{z}}, \overline{s})$ where $\overline{s} = o(q, \overline{e})$.*

Note that in the semantics of $\varphi^{\mathcal{T}}$ now the environment chooses the values $v_{\overline{z}}$ of the variables $\overline{z}$ that appear in the constraints in Γ. Also, note that the overall architecture of adaptive controllers (see Fig. 2) is similar to the one presented in Sec. 3, but using adaptive providers and extra input $v_{\overline{z}}$.

The following holds analogously to Thm. 1 in Sec. 3.

Theorem 3 (Correctness of Adaptive Synthesis). *Let $\varphi^{\mathcal{T}}$ be a realizable specification, $\varphi^{\mathbb{B}}$ its Boolean abstraction, α a* partitioner*, Γ an adaptive provider description and β_Γ an adaptive* provider*. Let $\rho^{\mathbb{B}}$ be a winning strategy for $\varphi^{\mathbb{B}}$, and $\rho^{\mathcal{T}}_{\Gamma}$ the strategy obtained as the composition of α, $\rho^{\mathbb{B}}$ and β_Γ described in Def. 7. Then $\rho^{\mathcal{T}}_{\Gamma}$ is winning for $\varphi^{\mathcal{T}}$.*

Proof (Sketch). The proof proceeds by showing that any play played according to $\rho^{\mathcal{T}}_{\Gamma}$ satisfies, at all steps, the same literals as $\rho^{\mathbb{B}}$, independently of the values of $\overline{z}$. It is crucial that, after each step, the controller state that $\rho_{I}^{\mathcal{T}}$ and $\rho^{\mathbb{B}}$ leave is the same, which holds because their δ' is indistinguishable. □

Skolem functions are computed from basic provider formulae that have a shape $\forall^{*}\exists^{*}.\psi$. This shape is preserved in adaptive provider formulae in which the constraint ψ^{+} is quantifier-free. However, as the last example illustrated, the constraint ψ^{+} may include quantifiers, which does not preserve the shape typically amenable for Skolemization.

For instance, to compute the smallest $y \in \mathbb{D}(\mathcal{T}_{\mathbb{Z}})$ in $\forall x.\exists y.(y > x)$, one can use the adaptive provider formula $\forall x.\exists y.[(y > x) \wedge \forall z.(z > x) \rightarrow (z \geq y)]$. We overcome this issue by performing quantifier elimination (QE) for the innermost quantifier and recover the $\forall^{*}\exists^{*}$ shape. In consequence, our resulting method for adaptive provider generation works on any theory $\mathcal{T}$ that:

(1) is decidable for the $\exists^{*}\forall^{*}$ fragment (for the Boolean abstraction);
(2) permits a Skolem function synthesis procedure (for valid $\forall^{*}\exists^{*}$ formulae), for producing static providers; and
(3) accepts QE (which preserves formula equivalence) for the flexibility in defining quentified constraints ψ^{+}.

Example 5. Consider again the strategy of Ex. 2 and the first Skolem function to synthetise at Ex. 5: $\forall x.\exists y.(x \geq 2) \rightarrow \psi$, where $\psi = [(x \geq 2) \wedge (y > 1) \wedge (y \leq x)]$. Then, we add the adaptivity criteria that we want our strategy to return the greatest value **possible**, so the function to synthetise is as follows:

$$\forall x.\exists y.(x \geq 2) \rightarrow (\psi \wedge \forall z.[(x \geq 2) \wedge (z > 1) \wedge (z \leq x) \rightarrow (z \leq y)])$$

We show below the results of `AEval` invocations with the original (left) and the adaptive (right) versions:

$$\mathtt{h}_{(e_1,c_4)}(x) = \begin{cases} 2 & \text{if } (x \geq 2) \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \mathtt{h}^{+}_{(e_1,c_4)}(x) = \begin{cases} x & \text{if } (x \geq 2) \\ 0 & \text{otherwise} \end{cases}$$

where we can see that, $\mathtt{h}_{(e_1,c_4)}$ is a more static function in the sense that it will always return 2, whereas $\mathtt{h}^{+}_{(e_1,c_4)}$ depends on the value of x in order to return exactly x (which is the greatest value possible).

5 Empirical Evaluation

We now report on empirical evaluation to asses the performance of our approach. We used Python 3.8.8 for the implementation of the architecture and Z3 4.12.2 for the SMT queries. We use Strix [30] as the synthesis engine and `aigsim.c` to execute the synthetised controller. For functional synthesis we used the `AEval` solver [16][5] that leverages Z3. Currently, `AEval` expects formulae in linear arithmetic with the $\forall^{*}\exists^{*}.\varphi$ shape, which is suitable for the static provider we want to synthesise. We translate the Skolem functions into `C++` and used `g++` 14.0.0 as a compiler. We ran all experiments on a MacBook Air 12.4 with the M1 processor and 16 GB. of memory.

[5] Publicly available at: `https://github.com/grigoryfedyukovich/aeval`

Wrap-up experiment. We first report our results on $\mathcal{T}$-controller for Ex. 1. Following the idea of Ex. 2, we execute the input trace $\pi = (\langle x | x \geq 2\rangle, \langle x | x \geq 2\rangle, \langle x | x < 2\rangle, \langle x | x < 2\rangle, \langle x | x \geq 2\rangle)$ 100000 times on (1) a **dynamic** provider following [36] and (2) our **static** provider approach. Throughout both experiments, the average time for the partitioner was 28 ms[6] and the average time for the Boolean controller execution was 2.47 μs. However, the average time for the **dynamic** provider was 169 μs, whereas the **static** provider was about 50 times faster: 2.9 μs. We show in Fig. 3 the time needed (in μs) of the **dynamic** provider and the **static** provider in the first 50 events. We can see that (1) the times required in the **dynamic** approach are more unstable and that (2) the **dynamic** approach is two orders of magnitude faster. Fig. 5 and Fig. 4 zoom over Fig. 3.

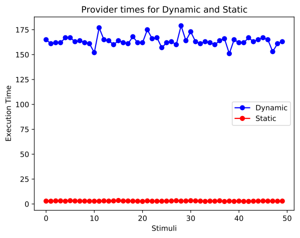

Fig. 3: Comparison.

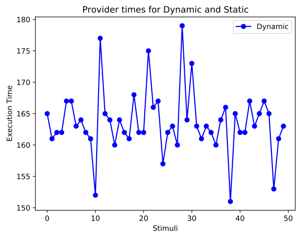

Fig. 4: Zoom in Dynamic.

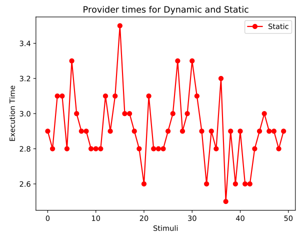

Fig. 5: Zoom in Static.

It is an important detail that, since π only provides ranges of inputs (e.g., a general $x | (x < 2)$ instead of a concrete $x : 1$), the input values may be different in both experiments. Therefore, we executed again the experiments over a same fixed input trace $\pi' = (\langle x : 4\rangle, \langle x : 4\rangle, \langle x : 1\rangle, \langle x : 0\rangle, \langle x : 2\rangle, \ldots)$ in order to do a sanity check. Fig. 6 shows that the execution with π' follows the same tendency as with π. Even though the input numbers are repeated, we still encounter differences in solving times both in Fig. 7 and Fig. 8, which suggests that, for such an amount of constraints to solve, the time to solve is not dominated by the input, but rather by implementation and memory details.

We also checked the predictability of both approaches using π': i.e., how much does the output differ given the same input and position in the game. Recall from Ex. 2 and π' that at timestep $t : 0$ the possible outcomes are $v_{\overline{y}} \in \{2, 3, 4\}$, at $t : 1$ again $v_{\overline{y}} \in \{2, 3, 4\}$, at $t : 2$ $v_{\overline{y}} \in \{2, 3, 4, ...\}$, at $t : 3$ again $v_{\overline{y}} \in \{2, 3, 4, ...\}$ and at $t : 4$ $v_{\overline{y}} \in \{2\}$. Let us denote with k the repetition of this pattern. At time $t : 0 + k$ and $t : 1 + k$ there are three valid outputs, at $t : 2 + k$ and $t : 3 + k$ there are infinitely many valid outputs and at $t : 4 + k$ there is a single valid output. Also, note that $v_{\overline{y}}$ should be a valid output for every input in π' (as captured by both Skolem functions in Ex. 3). In Fig. 9, we show results for 500 timesteps

[6] Note that the time is dominated by the partitioner, shared in both cases, which consists on searching among a finite collection of formulae (valid reactions) to find the correct partition and can be easily optimized.

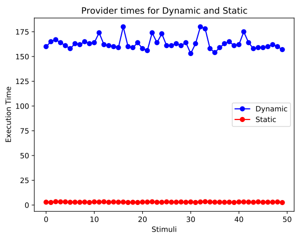

Fig. 6: Comparison.

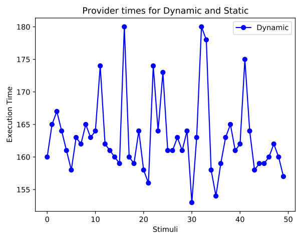

Fig. 7: Zoom in Dynamic.

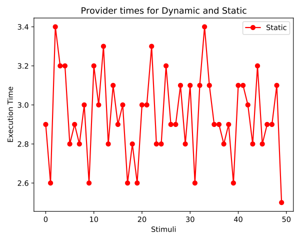

Fig. 8: Zoom in Static.

$(100 \times |\pi'|)$. We can see that the dynamic provider is less predictable (outputs), whereas the static provider always produces the value 2. Note that output values in the dynamic provider are always different in every experiment, but the general shape remains similar. Also, note that one might consider that predictability in the dynamic approach is also remarkably stable, since only for 20 times out of the total of 400 the provider produces a value different than 2 (17 times value 3, twice the value 4 and once value 5).

However, this stability difference increases when adaptivity is considered. Concretely, we will use *return the greatest* (illustrated in Ex. 5) for $t : 0 + k$, $t : 1 + k$ and $t : 0 + 4$, and *return the smallest* for $t : 2 + k$ and $t : 3 + k$, which means that the *ideal* output trace with respect to these criteria is the pattern $(\langle y : 4\rangle, \langle y : 4\rangle, \langle y : 2\rangle, \langle y : 2\rangle, \langle y : 2\rangle)$. As expected, the static provider always returns the ideal pattern, whereas the pattern in the dynamic case is more unstable. For example, in Fig. 10, we show results for 50 stimuli in π', where three times the output was not within the ideal pattern. We acknowledge that, in the dynamic approach, as the timeout restrictions for the underlying SMT solver gets more strict (e.g., in fast embedded contexts), less adaptive constraints will be solved and thus the output will tend to diverge more from the ideal pattern.

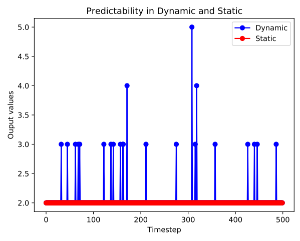

Fig. 9: Comparison.

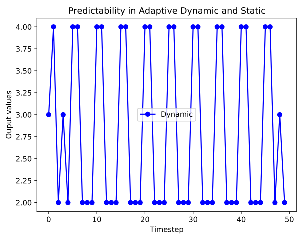

Fig. 10: Dyn. Adaptive.

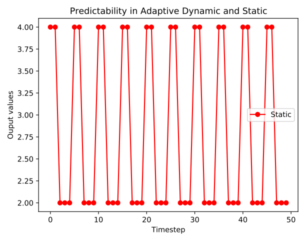

Fig. 11: Static Adaptive.

Benchmark results. In order to perform a wide empirical evaluation, we used benchmarks from [36] to validate the hypothesis whether that our static approach is faster and more predictable. Tab. 1 shows the main group of experiments. *Sz.* refers to the size of the $\varphi^{\mathcal{T}}$ specification both in variables (*vr.*) and literals (*lt.*). *Prep.* refers to the pre-processing time (i.e., the time needed to compute the Boolean abstraction and synthetize a Boolean controller). This and the computation of the partitioner are performed at compile time, while in our experiments provider is computed dynamically for each new $(v_{\overline{e}}, v_{\overline{s}})$ pair discovered. Note that the time necessary to compute Skolem functions was negligible (around 2 seconds) as expected, since the number of constraints we used does not stress `AEval` [7]. The following two groups of columns show results of the execution of 1000 (1K) and 10000 (10K) timesteps of input-output simulations. For each group we do not show the average time that the partitioner takes to respond with a discrete $v_{\overline{e}}$ from $v_{\overline{x}}$ and the average time that the Boolean controller takes to respond with a Boolean c_i, since it is the same for the **dynamic** and the **static** approaches. Instead, we report the time that the provider takes to respond with a valuation associated to c_i in **dynamic** (*Dyn.*) and **static** (*St.*). Note that *Tr.* is also the benchmark that takes the most time on average for *Pr.*, since its functions contain more operations, but it is still efficient enough for the targeted applications. Overall, we can see that the static approach is 50 to 60 times faster.

In addition, we used adaptivity with different criteria. We selected Skolem functions of each benchmark and created adaptive versions that return: (1) minimum and maximum valuation possible (*m/m*) knowing there was at least one of such bounds to stop the search and (2) valuation closest to a p point (*pc.*) that is randomly generated. Then, groups (10K (*m/m*)) and (10K (*pc.*)) measure again average times of the dynamic provider and adaptive provider. Note that it is not clear how adaptivity impacts provider time, since sometimes the amount of adaptive constraints dominates this time, whereas in other cases the complexity of the criteria (e.g., underlying quantified structure) seems to be more relevant.

Moreover, we show approximated predictability (*Pre.*) results of the dynamic approach as a percentage that measures how many of the outputs diverged from the ideal pattern (e.g., we previously mentioned that in Fig. 9 this number was $\sim 5\%$: 20 divergences out of 400 stimuli). In Tab. 1 we can see that predictability is between 2 and 15 percent (average about 5%), whereas in the static varsion it is 0% (not shown in the table). More importantly, it seems that adding adaptivity tends to result in less predictable outputs in the dynamic provider. These results support our hypothesis that our static controller is more efficient and predictable.

We also tested whether $\mathcal{T}$ affected the performance of the controller, via use cases *Syn. (2,3)* to *Syn. (2,6)* of [35] interpreted over linear integer arithmetic and linear real arithmetic (the theories accepted by `AEval`). We show the results in Fig. 12, where we compare 1000 simulations in the basic case (*bs.*), and the adaptive minimal/maximal (*m/m.*) and randomly generated point (*pc.*) cases.

[7] Indeed, an eventual input with a larger amount of constraints (i.e., literals) may or may not yield a bottleneck for the underlying Boolean abstraction procedure, but not for the approach we present in this paper.

Bn. (nm.)	Sz. (vr, lt)	*Prep.* (s.)	$1K$ Dyn.	$1K$ St.	$10K$ Dyn.	$10K$ St.	$10K$ Pre.	$10K$ $(m/m.)$ Dyn.	$10K$ $(m/m.)$ St.	$10K$ $(m/m.)$ Pre.	$10K$ $(pc.)$ Dyn.	$10K$ $(pc.)$ St.	$10K$ $(pc.)$ Pre.
Li.	(5, 16)	33.73	240	5.74	232	4.54	~ 4	216	4.01	~ 4	204	3.52	~ 7
Tr.	(19, 36)	9219.11	272	5.05	262	5.03	~ 9	294	5.08	~ 14	270	4.92	~ 15
Con.	(2, 2)	0.09	104	2.08	104	1.89	~ 2	107	1.94	~ 2	129	2.18	~ 2
Coo.	(3, 5)	2.60	171	2.94	168	2.84	~ 3	168	3.32	~ 4	173	2.81	~ 3
Usb	(5, 8)	346.29	302	6.04	304	4.82	~ 5	329	5.46	~ 7	313	6.00	~ 6
Sta.	(11, 14)	182.1	295	4.91	291	5.19	~ 6	299	5.24	~ 9	298	4.73	~ 11

Table 1: Results in [36] (see Dyn.) versus ours (see St.), where times are measured in μ. Recall from [35] that if the literals in φ are split into clusters that do not share variables, the Boolean abstraction process can handle each cluster independently and composed afterwards.

Note that run-time difference is negligible and so was in the abstraction phase. Also, note that Skolem function synthesis was slightly harder in integers.

6 Related Work and Conclusions

Related Work. LTL modulo theories has been previously studied (e.g., [23,14]), but allowing temporal operators within predicates, again leading to undecidability. Also, infinite-state synthesis has been recently studied at [8,15,19,39,3,24] but with similar restrictions. At [26,27] authors perform reactive synthesis based on a fixpoint of $\forall^*\exists^*$ formulae (for which they use `AEval`), but expressivity is limited to safety and does not guarantee termination. The work in [41] also relies on abstraction but needs guidance and again expressivity is limited. Reactive synthesis of Temporal Stream Logic (TSL) modulo theories [18] is studied in [9,28], which extends LTL with complex data that can be related accross time. Again, note that general synthesis is undecidable by relating values across time. Moreover, TSL is already undecidable for safety, the theory of equality and Presburger arithmetic. Thus, all the specifications considered for empirical evaluation in Sec. 5 are not within the considered decidable fragments.

All approaches above adapt one specific technique and implement it in a monolithic way, whereas [35,37] generates LTL specification that existing tools

Lits	Linear I. Arithmetic 1K (bs.)	Linear I. Arithmetic 1K (m/m)	Linear I. Arithmetic 1K (pc.)	Linear R. Arithmetic 1K (bs.)	Linear R. Arithmetic 1K (m/m)	Linear R. Arithmetic 1K (pc.)
3	3.41	2.32	2.00	3.45	2.14	2.02
4	3.50	2.05	2.07	198	3.60	2.09
5	3.63	2.64	2.63	3.83	2.65	2.67
6	3.78	3.30	3.38	3.85	3.39	3.45

Fig. 12: Comparison of $\mathcal{T}_{\mathbb{Z}}$ and $\mathcal{T}_{\mathbb{R}}$ for *Syn (2,3)* to *Syn (2,6)*.

can process with any of their internal algorithms (bounded synthesis, for example) so we will automatically benefit from further optimizations in these techniques. Moreover, Boolean abstraction preserves the temporal fragments like safety and GR(1) so specialized solvers can be used. Throughout the paper, we have already extensively compared the work [36] with ours and we showed that our approach uses Skolem functions instead of SMT queries on-the-fly, which makes it faster, more predictable and a pure controller that can be used in embedded contexts. It is worth noting that [36] and our approach can be understood as computing *minterms* to produce Symbolic automata and transducers [11,12] from reactive specifications (and using antichain-based optimization, as suggested by [40]). Also, note that any advance in abstraction method (e.g., [4]) has an immediate positive impact in our work.

Last, [25] presents a similar idea to our Skolem function synthesis: instead of solving a quantified formula every time one wants to compute an output, they synthesize a term that computes the output from the input. However, the paper is framed in the program synthesis problem and uses syntax-guided synthesis [2], whereas previous reactive synthesis papers have suggested functional synthesis as a recommended software engineering practise (e.g., [38]).

Conclusion. The main contribution of this paper is the synthesis procedure for $\text{LTL}^{\mathcal{T}}$, using internally a Boolean controller and static Skolem function synthesis, which is more performant and predictable than previous approaches. Our method also allows producing *adaptive responses* that optimize the behaviour of the controller with respect to different criteria. We showed empirically that our approach is fast for many targeted applications and analyzed the cost and predictability of our Skolem functions component compared to [36]. As far as we know, this is the first decidable full reactive synthesis approach (with or with adaptivity) for $\text{LTL}^{\mathcal{T}}$ specifications.

Future work includes first to use winning regions instead of concrete controllers to allow even more choices for the Skolem functions, and to develop a further adaptivity theory. Another direction is studying adaptivity over the environment inputs and combining this approach with monitoring. Also, we plan to study how to extend $\text{LTL}^{\mathcal{T}}$ with transfer of data accross time preserving decidability, since recent results [22] suggest that the expressivity can be extended with limited transfers in semantic fragments of $\text{LTL}^{\mathcal{T}}$. Moreover, explaining our synthesis approach within more general frameworks like (e.g., [21]) is immediate work to do. Finally, we want to study how to use our approach to construct more predictable and performant shields [1,6] (concretely, shields modulo theories [34,10]) to enforce safety in critical systems.

References

1. Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proc. of the Thirty-Second AAAI*

Conference on Artificial Intelligence, (AAAI-18), pages 2669–2678. AAAI Press, 2018.

2. Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Proc. of the 13th Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD 2013)*, pages 1–8. IEEE, 2013.
3. Shaun Azzopardi, Nir Piterman, Gerardo Schneider, and Luca di Stefano. Ltl synthesis on infinite-state arenas defined by programs, 2023.
4. Shaun Azzopardi, Nir Piterman, Gerardo Schneider, and Luca Di Stefano. LTL synthesis on infinite-state arenas defined by programs. *CoRR*, abs/2307.09776, 2023.
5. Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.
6. Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. 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 2015)*, volume 9035 of *LNCS*, pages 533–548. Springer, 2015.
7. Aaron R. Bradley and Zohar Manna. *The Calculus of Computation.* Springer-Verlag, 2007.
8. Chih-Hong Cheng and Edward A. Lee. Numerical LTL synthesis for cyber-physical systems. *CoRR*, abs/1307.3722, 2013.
9. Wonhyuk Choi, Bernd Finkbeiner, Ruzica Piskac, and Mark Santolucito. Can reactive synthesis and syntax-guided synthesis be friends? In Ranjit Jhala and Isil Dillig, editors, *43rd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation (PLDI 2022)*, pages 229–243. ACM, 2022.
10. Davide Corsi, Guy Amir, Andoni Rodriguez, Cesar Sanchez, Guy Katz, and Roy Fox. Verification-guided shielding for deep reinforcement learning. *CoRR*, abs/2406.06507, 2024.
11. Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*, pages 541–554. ACM, 2014.
12. Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *Proc. of the 29th International Conference in Computer Aided Verification (CAV 2017), Part I*, volume 10426 of *LNCS*, pages 47–67. Springer, 2017.
13. Leonardo de Moura and Nikolaj Bjørner. 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)*, volume 4693 of *LNCS*, pages 337–340. Springer, 2008.
14. Rachel Faran and Orna Kupferman. LTL with arithmetic and its applications in reasoning about hierarchical systems. In *Proc. of the 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, (LPAR-22.)*, volume 57 of *EPiC Series in Computing*, pages 343–362. EasyChair, 2018.
15. Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proc. ACM Program. Lang.*, 2(POPL):61:1–61:30, 2018.
16. Grigory Fedyukovich, Arie Gurfinkel, and Aarti Gupta. Lazy but effective functional synthesis. In *20th International Conference in Verification, Model Checking, and Abstract Interpretation, (VMCAI 2019)*, volume 11388 of *LNCS*, pages 92–113. Springer, 2019.
17. Bernd Finkbeiner. Synthesis of reactive systems. In Javier Esparza, Orna Grumberg, and Salomon Sickert, editors, *Dependable Software Systems Engineering*, vol-

ume 45 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 72–98. IOS Press, 2016.

18. Bernd Finkbeiner, Philippe Heim, and Noemi Passing. Temporal stream logic modulo theories. In *25th Int'l Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2022)*, volume 13242 of *LNCS*, pages 325–346. Springer, 2022.
19. Andrew Gacek, Andreas Katis, Michael W. Whalen, John Backes, and Darren D. Cofer. Towards realizability checking of contracts using theories. In *Proc. of the 7th International Symposium NASA Formal Methods (NFM 2015)*, volume 9058 of *LNCS*, pages 173–187. Springer, 2015.
20. Luca Geatti, Alessandro Gianola, and Nicola Gigante. Linear temporal logic modulo theories over finite traces. In *Proc. of the 31st International Joint Conference on Artificial Intelligence, (IJCAI 2022)*, pages 2641–2647. ijcai.org, 2022.
21. Luca Geatti, Alessandro Gianola, and Nicola Gigante. A general automata model for first-order temporal logics (extended version). *CoRR*, abs/2405.20057, 2024.
22. Luca Geatti, Alessandro Gianola, Nicola Gigante, and Sarah Winkler. Decidable fragments of ltlf modulo theories (extended version). *CoRR*, abs/2307.16840, 2023.
23. Alessandro Gianola and Nicola Gigante. LTL modulo theories over finite traces: modeling, verification, open questions. In *Short Paper Proceedings of the 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA 2022)*, volume 3311 of *CEUR Workshop Proceedings*, pages 13–19. CEUR-WS.org, 2022.
24. Philippe Heim and Rayna Dimitrova. Solving infinite-state games via acceleration. *Proc. ACM Program. Lang.*, 8(POPL):1696–1726, 2024.
25. Qinheping Hu and Loris D'Antoni. Automatic program inversion using symbolic transducers. In *Proc. of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*, pages 376–389. ACM, 2017.
26. Andreas Katis, Grigory Fedyukovich, Andrew Gacek, John D. Backes, Arie Gurfinkel, and Michael W. Whalen. Synthesis from assume-guarantee contracts using skolemized proofs of realizability. *CoRR*, abs/1610.05867, 2016.
27. Andreas Katis, Grigory Fedyukovich, Huajun Guo, Andrew Gacek, John Backes, Arie Gurfinkel, and Michael W. Whalen. Validity-guided synthesis of reactive systems from assume-guarantee contracts. In *Proc. of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS 2018)*, volume 10806 of *LNCS*, pages 176–193. Springer, 2018.
28. Benedikt Maderbacher and Roderick Bloem. Reactive synthesis modulo theories using abstraction refinement. In *22nd Formal Methods in Computer-Aided Design, (FMCAD 2022)*, pages 315–324. IEEE, 2022.
29. Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety.* Springer, 1995.
30. Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 578–586, Cham, 2018. Springer International Publishing.
31. Amir Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS'77)*, pages 46–67. IEEE CS Press, 1977.
32. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proc. of the 16th Annual ACM Symp. on Principles of Programming Languages (POPL'89)*, pages 179–190. ACM Press, 1989.

33. Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of the 16th Int'l Colloqium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.
34. Andoni Rodriguez, Guy Amir, Davide Corsi, Cesar Sanchez, and Guy Katz. Shield synthesis for LTL modulo theories. *CoRR*, abs/2406.04184, 2024.
35. Andoni Rodríguez and César Sánchez. Boolean Abstractions for Realizability Modulo Theories. In *Proc. of the 35th Int'l Conf. on Computer Aided Verification (CAV 2023)*, volume 13966 of *LNCS*, pages 1–24. Springer, 2023.
36. Andoni Rodriguez and César Sánchez. Adaptive reactive synthesis for LTL and LTLf modulo theories. In *Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024)*, pages 10679–10686. AAAI Press, 2024.
37. Andoni Rodriguez and César Sanchez. Realizability modulo theories. *Journal of Logical and Algebraic Methods in Programming*, page 100971, 2024.
38. Stanly Samuel, Deepak D'Souza, and Raghavan Komondoor. 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)*, pages 1585–1589. ACM, 2021.
39. Stanly Samuel, Deepak D'Souza, and Raghavan Komondoor. Symbolic fixpoint algorithms for logical LTL games. In *Proc. of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*, pages 698–709. IEEE, 2023.
40. Margus Veanes, Thomas Ball, Gabriel Ebner, and Olli Saarikivi. Symbolic automata: ω-regularity modulo theories. *CoRR*, abs/2310.02393, 2023.
41. Adam Walker and Leonid Ryzhyk. Predicate abstraction for reactive synthesis. In *Proc. of the 14th Formal Methods in Computer-Aided Design, (FMCAD 2014)*, pages 219–226. IEEE, 2014.