Realizability Modulo Theories

Andoni Rodríguez[a,b], César Sánchez[a]

[a] *IMDEA Software Institute, Madrid, Spain*
[b] *Universidad Politécnica de Madrid, Spain*

Abstract

In this paper we study the problem of realizability of reactive specifications written in $\text{LTL}_{\mathcal{T}}$, which is the extension of LTL where atomic propositions can be literals from a first-order theory, including arithmetic theories. We present a solution based on transforming $\text{LTL}_{\mathcal{T}}$ specifications into purely Boolean specifications by (1) substituting theory literals by Boolean variables, and (2) computing an additional Boolean formula that captures the dependencies between the new variables imposed by the literals. We prove that the resulting specification is realizable if and only if the original specification is realizable. Moreover, the resulting specification can be passed to existing Boolean off-the-shelf synthesis and realizability tools, which can handle only Boolean LTL specifications.

A second contribution is to prove that $\text{LTL}_{\mathcal{T}}$ realizability of theories with a decidable $\exists^*\forall^*$ fragment is decidable for all combinations of LTL temporal modalities. We present a simple version of our method, which relies on SMT solving, and performs a brute-force search to construct the "extra requirement". A third contribution is an algorithm that checks whether a candidate is a correct *Booleanization.* in non-Boolean LTL realizability.

Keywords: `Boolean abstraction, LTL modulo theories, (Reactive) synthesis, Reactive systems, Realizability, Formal methods, Formal specification, SMT solving, First-order theories`

1. Introduction

Reactive synthesis [1, 2] is the problem of automatically producing a system that is guaranteed to model a given temporal specification, typically written in Linear Temporal Logic (LTL) [3], where the atomic propositions are split into those controlled by the environment and those controlled by the system. The system produced using reactive synthesis is guaranteed to respond to the environment satisfying the specification. Realizability is the related decision problem of deciding whether such a system exists. These problems have been widely studied for LTL (e.g., [4, 5]). Realizability corresponds to infinite games where players (the system and the environment) alternatively choose the valuations of the Boolean atomic propositions they control. The winning condition is extracted from the specification and determines which player wins a given play. A system is realizable if and only if the system player has a winning strategy, that is, if there is a way to play such that the specification is satisfied in all traces that result from plays where the system player plays according to the strategy. Note that the objective of the environment player is to violate the specification. Also, note that both the winning conditions of the system and environment can be represented using an arena that contains all posible transitions and positions of the game, where the winning condition of the environment is a reachability goal, i.e., it wants to reach *violation* positions of the arena, whereas the goal of the system is to avoid such positions inifnitely many often.

Many industrial specifications use complex data beyond Boolean atomic propositions, which precludes the direct use of realizability and synthesis methods and tools that are typically restricted to Booleans. These specifications cannot be modelled in (propositional) LTL, but require an extension of LTL where Boolean atomic propositions can be literals from a (multi-sorted) first-order theory $\mathcal{T}$, which means that all the atoms in such literals belong to $\mathcal{T}$ (e.g., in $(4.2 < 5.5)$ atoms belong to $\mathbb{R}$). We call this extension LTL$_{\mathcal{T}}$. In this case, the variables that appear in the specification are of sorts from the theory $\mathcal{T}$ (for example, integers or reals) and are split into those controlled by the environment and those controlled by the system. A trace of values is formed by players choosing in turn values for the variables they control in the domain of $\mathcal{T}$. These values induce sequences of Boolean valuations of the literals (because valuations in $\mathcal{T}$ make some literals hold and some others not), which yields a trace of values satisfying or violating the specification. Realizability for $\text{LTL}_\mathcal{T}$ also corresponds to infinite games, but in this case players chose valuations from the domains of $\mathcal{T}$, which may be infinite for many theories. Therefore, arenas for $\text{LTL}_\mathcal{T}$ games have infinite positions and positions have infinitely many successors (if the domains of the theory are infinite).

In this paper, we present a method to solve the problem of reactive realizability modulo theories, that is, realizability of $\text{LTL}_\mathcal{T}$ specifications. Our method transforms a specification that uses data from a theory $\mathcal{T}$ into an equi-realizable Boolean specification, which means that the original $\mathcal{T}$-specification is realizable if and only if the obtained Boolean specification is realizable (in the same spirit a formula ϕ is equi-satisfiable to another formula ϕ' whenever ϕ is satisfiable if and only if ϕ' is satisfiable).

The resulting specification can then be discharged (i.e., given as input to) into an off-the-shelf realizability tool.

The main element of our solution is a *Boolean abstraction* method, which allows to transform $\text{LTL}_\mathcal{T}$ specifications into LTL specifications. The method first substitutes all $\mathcal{T}$ literals by fresh Boolean variables controlled by the system, and then extends the specification with an additional sub-formula that constrains the values of these variables. The main idea is that, after the environment selects values for its (data) variables, the system responds with values for the remaining variables (that the system controls). This creates a valuation for all variables and in turn a Boolean value for each literal. The additional formula should capture the set of possible valuations of literals and the precise options of each player to influence each valuation. In the rest of the paper, we will refer to these options as *power*: e.g., *the system has the power to satisfy a given literal* or *the environment has the power to force the system to satisfy a given literal.*

Example 1.1. *Consider the following specification* $\varphi = \square(R_0 \wedge R_1)$, *where* x *belongs to the environment player and* y *to the system player:*

$$R_0 : (x < 2) \rightarrow \bigcirc(y > 1) \qquad\qquad R_1 : (x \geq 2) \rightarrow (y < x),$$

where $\Box$ and $\bigcirc$ are the classic "always" (means a property is an invariant and must hold in all timesteps) and "next" (a property must hold in the upcoming timestep) operators in LTL, respectively. In the game corresponding to this specification, each player has an infinite number of choices at each time step. For example, in the case of $\mathcal{T}_{\mathbb{Z}}$—the theory of integer arithmetic—, the environment player chooses an integer for x and the system responds with an integer for y. This induces a valuation of the literals in the formula, which in turn induces (also considering the valuations of the literals at other time instants) a valuation of the full specification, according to the temporal operators.

However, in this paper we prove that, from the point of view of the valuations of the literals, there are only finitely many *cases and provide a systematic manner to compute these cases. We do this by reducing the specifications to a purely Boolean specification that is equi-realizable. This specification shows the (finite) set of decisions of the environment, and the (finite) set of reactions of the system to each of those decisions. For instance, we later see that, if we interpret this specification in $\mathcal{T}_{\mathbb{Z}}$, the environment only has three decisions that matter: playing any x such that $(x < 2)$, playing $x : 2$ and playing any x such that $(x > 2)$. Note that both the specific finite set of decisions and the realizability depend on the theory in which we interpret the specification.*

$\Box$

Example 1.1 suggests a naive algorithm to capture the power of the environment and the system to determine a combination of the valuations of the literals, by enumerating all these combinations and checking the validity of each potential reaction of the system. We will see that checking whether a given combination is a possible reaction requires solving an $\exists^*\forall^*$ query (which can be delegated to an SMT solver for appropriate theories). We will see that, if a specification contains literals from $\exists^*\forall^*$-decidable theories (which means that validity of formulae with the $\exists^*\forall^*$ prefix can be decided if interpreted in such theories), then the specification is *Booleanizable*. We prove in this paper that our method leads to a correct Boolean abstraction for all LTL$_\mathcal{T}$ specifications, no matter the temporal fragment, but we use safety specifications in the presentation for simplicity.

In summary, our contributions are:

- a proof that realizability is decidable for all LTL$_\mathcal{T}$ specifications for those theories $\mathcal{T}$ with a decidable $\exists^*\forall^*$ fragment;
- a Boolean abstraction method that transforms LTL$_\mathcal{T}$ specifications into equi-realizable LTL specifications; and
- an algorithm that checks whether a candidate for a *Booleanized* version of a specification is correct.

To the best of our knowledge, this is the first method that succeeds in general non-Boolean LTL realizability or synthesis.

The remainder of the paper is structured as follows. Section 2 contains the preliminaries. Section 3 introduces syntax and semantics of LTL$_\mathcal{T}$. Section 4 presents the Boolean abstraction algorithm that produces a Boolean specification $\varphi_\mathbb{B}$ from a given reactive specification $\varphi_\mathcal{T}$. Section 5 proves that $\varphi_\mathbb{B}$ and $\varphi_\mathcal{T}$ are equi-realizable. The core of the transformation from $\varphi_\mathcal{T}$ into $\varphi_\mathbb{B}$ is the search for a key *extra requirement*, which is a computationally slow process in the algorithm shown in Section 4. Thus, in Section 6, we show an algorithm that checks whether a given $\varphi_\mathbb{B}$ candidate is a correct Booleanization. This opens the door to faster algorithms to compute Booleanizations that are later check for correctness. Section 7 contains related work and Section 8 concludes and raises future lines of research.

2. Preliminaries

We fix an alphabet $\Sigma = 2^{\mathsf{AP}}$, where AP is a set of *atomic propositions* and we call each element $a \in \Sigma$ a *letter*. A *trace* is an infinite sequence $\sigma = a_0 a_1 ...$ of letters from Σ. We denote the set of all infinite traces by Σ^ω. We use $\sigma(i)$ for a_i and σ^i for the suffix $a_i a_{i+1} ...$

2.1. First-order theories

We use sorted first-order theories for extending the expressivity of the atomic predicates. For our purposes, a (sorted) first-order theory $\mathcal{T}$ consists of (1) a first-order vocabulary to form terms and predicates, (2) an interpretation of the domains of the sorts, and (3) an automatic reasoning system to decide the validity of sentences in the theory. A first-order theory $\mathcal{T}$ (see e.g., [6]) is given by its signature Γ, which is a set of constant, function, and predicate symbols. A Γ-formula is constructed from constant, function, and predicate symbols of Γ, together with logical connectives and quantifiers. The symbols Γ are symbols without prior meaning that are later interpreted.

A Γ-formula φ is valid in the theory $\mathcal{T}$ (i.e., φ is $\mathcal{T}$-valid), if for every interpretation I that satisfies the axioms of $\mathcal{T}$, then $I \vDash \varphi$. We write $\mathcal{T} \vDash \varphi$ to mean that φ is $\mathcal{T}$-valid. Formally, the theory $\mathcal{T}$ consists of all closed formulae that are $\mathcal{T}$-valid, where *closed* means that all the variables are quantified in the formula A Γ-formula φ is satisfiable in $\mathcal{T}$ (i.e., $\mathcal{T}$-satisfiable), if there is a $\mathcal{T}$-interpretation I that satisfies φ.

A fragment of a theory is a syntactically-restricted subset of formulae of the theory. For example, the quantifier-free fragment of a theory $\mathcal{T}$ is the set of formulae without quantifiers that are valid in $\mathcal{T}$. A theory $\mathcal{T}$ is decidable if $\mathcal{T} \vDash \varphi$ is decidable for every Γ-formula φ. That is, there is an algorithm that always terminates with $\top$ if φ is $\mathcal{T}$-valid or with $\bot$ if φ is $\mathcal{T}$-invalid. A fragment of $\mathcal{T}$ is decidable if $\mathcal{T} \vDash \varphi$ is decidable for every Γ-formula φ in the fragment. In this paper, we consider theories $\mathcal{T}$ whose $\exists^*\forall^*$ fragment is decidable.

A particular class of first-order theories is that of arithmetic theories, which are well studied in mathematics and theoretical computer science, and are of particular relevance in formal methods. Most of these theories are decidable. We describe below some of these theories:

- *Linear Natural Arithmetic* $\mathcal{T}_{\mathbb{N}}$ is the theory of natural numbers with addition but no arbitrary multiplication [7, 8, 9]. The signature is: $\Gamma_{\mathbb{N}} = \{0, 1, \ldots, +, =, >\}$. One example of a literal is $(2x > 4)$.

- *Linear Integer Arithmetic* $\mathcal{T}_{\mathbb{Z}}$ the theory of integer numbers with addition but no arbitrary multiplication [10, 11, 6, 12]. The signature is $\Gamma_{\mathbb{Z}} = \{\ldots, -1, 0, 1, \ldots, , +, -, =, >\}$. A literal is $(2x > -4)$.

- *Linear Rational/Real Arithmetic* $\mathcal{T}_{\mathbb{Q}}$: the theory of real/rational numbers with addition but no arbitrary multiplication [13, 14, 6, 15]. The signature is $\Gamma_{\mathbb{Q}} = \{0, k, +, -, =, >\}$, where $k \in \mathbb{Q}$. For instance, a literal is $(2x > -\frac{1}{3})$.

- *Nonlinear Real Arithmetic* $\mathcal{T}_{\mathbb{R}}$ is the theory of real numbers with both addition and arbitrary multiplication [16, 17, 18, 19]. The signature is $\Gamma_{\mathbb{R}} = \{0, k, +, -, =, >, \cdot\}$. An example of a literal is $(2x^2 > \frac{1}{3})$.

- *Nonlinear Complex Arithmetic* $\mathcal{T}_{\mathbb{C}}$ is the theory of complex numbers with both addition and arbitrary multiplication [20, 21, 22, 23]. The signature is $\Gamma_{\mathbb{C}} = \{0, k, +, -, =, \cdot, i\}$. For instance, a literal is $(2x^2 = -3)$. Note that $\mathbb{C}$ is not totally ordered and lacks inequality operator.

Each of these theories has a different decision procedure and decidability complexity, but they are usually handled by SMT solvers.

Even though our Boolean abstraction technique is applicable to any theory with a decidable $\exists^*\forall^*$ fragment, we show our technique with arithmetic specifications. Concretely, we will consider $\mathcal{T}_{\mathbb{Z}}$ and $\mathcal{T}_{\mathbb{R}}$, for illustrative purposes and since they are the most common to appear in the literature. Note that the concrete theory used influences the realizability of a given formula. For instance, in Example 1.1, φ is not realizable for $\mathcal{T}_{\mathbb{Z}}$, since, if at a given instant t, the environment plays $x : 0$ (and hence $x < 2$ is true), then y must be greater than 1 at timestep $t + 1$. Then, if at $t + 1$ the environment plays $x : 2$, then $(x \geq 2)$ is true but there is no y such that both $(y > 1)$ and $(y < 2)$ at time $t + 1$. On the contrary, for $\mathcal{T}_{\mathbb{R}}$, φ is realizable: consider the system strategy to always play $y : 1.5$.

Example 2.1. *The following is a slight modification of Example 1.1 that alters its realizability (R_1' now has $y \leq x$ instead of $y < x$):*

$$R_0 : (x < 2) \rightarrow \bigcirc(y > 1) \qquad\qquad R_1' : (x \geq 2) \rightarrow (y \leq x)$$

Now, $\varphi' = \square(R_0 \wedge R_1')$ is realizable also for $\mathcal{T}_{\mathbb{Z}}$, as the strategy of the system to always pick $y : 2$ is winning. $\square$

2.2. Linear Temporal Logic (LTL)

Linear temporal logic is a modal temporal logic with modalities referring to linear time. The syntax of LTL is the following:

$$\varphi ::= T \mid a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{R}\, \varphi$$

where $a \in AP$ is an atomic proposition, $\vee$, $\wedge$ and $\neg$ are the usual Boolean disjunction, conjunction and negation, and $\bigcirc$, $\mathcal{U}$ and $\mathcal{R}$ are the next, until and release temporal operators. The semantics of LTL associate traces $\sigma \in \Sigma^\omega$ with formulae as follows:

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

The set of temporal operators $\bigcirc$, $\mathcal{U}$ and $\mathcal{R}$ is not minimal but admits a negation normal form. Common derived operators are $\Diamond\varphi \stackrel{\text{def}}{=} T\mathcal{U}\varphi$ and $\square\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$. For simplicity in the explanation, we restrict ourserlves to formulae that only use $\bigcirc$ (and an outermost $\square$), which are safety properties

A Kripke structure [24, 25] is a 4-tuple $K = \langle S, I, R, L\rangle$, where S is a finite set of states, $I \subseteq S$ is the set of initial states, $R \subseteq S \times S$ is a transition relation and $L : S \rightarrow 2^{AP}$ is a labeling function. The relation between a Kripke structure, LTL, and reactive systems lies in how the Kripke structure provides a formal semantics for interpreting and evaluating LTL formulas over the states of a system. It allows to reason about the temporal behavior and properties of reactive systems and check whether they meet the desired specifications expressed in LTL. By employing model checking techniques, it is possible to automatically verify the satisfaction of LTL properties in a given Kripke structure, providing formal guarantees about the behavior of reactive systems.

2.3. Infinite games

A game is composed of an arena and a winning condition. An arena is a triple $\mathcal{A} = \langle V_0, V_1, \rightarrow\rangle$, where V_0 is a set of 0-positions, V_1 is a set of 1-positions, disjoint from V_0, and $\rightarrow \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$ is the set of edges. We use $V = V_0 \cup V_1$. The games we consider are played by two players: Player 0 and Player 1. Note that there is no restriction on the number of the successors of a position in an arena. Also, a play in $\mathcal{A}$ is an infinite path $\pi = v_0, v_1, ... \in V^\omega$ with $v_i, v_{i+1} \in \rightarrow$ for all i, where $\text{Visit}(\pi) = \{v \in V \mid \pi(i) = v \text{ for some } 0 \leq i < \omega\}$ is the set of visited positions in a π that ranges from 0 to potentially ω.

Let $\mathcal{A}$ be an arena and $\mathit{Win} \subseteq V$, then the pair $(\mathcal{A}, \mathit{Win})$ is called a *game*, where $\mathcal{A}$ is the arena of the game and Win its winning set. The plays of that game are the plays in $\mathcal{A}$. Player 1 wins play π if and only if Player 0 cannot move (i.e., v_i is a dead end); or (2) π is an infinite play and $\text{Visit}(\pi) \subseteq \mathit{Win}$. Player 0 wins π if Player 1 does not win π.

A strategy for a Player i in an arena $\mathcal{A}$ is a function $\rho : V^*V_i \rightarrow V$ such that whenever $\rho(wv) = v'$, then $(v, v') \in\rightarrow$. A play π on an arena $\mathcal{A}$ is consistent with a strategy ρ whenever for all $n \in \mathbb{N}$ with $\pi_n \in V_i$ we have that $\rho(\pi_0 \cdots \pi_n) = \pi_{n+1}$. We denote the set of plays of $\mathcal{A}$ with $\mathit{Plays}(\mathcal{A})$, and the set of plays of $\mathcal{A}$ from position v with $\mathit{Plays}(\mathcal{A}, v)$. Let $\mathcal{G} = (\mathcal{A}, \mathit{Win})$ be a game and ρ be a strategy for Player i. Strategy ρ is a winning strategy from position $v \in \mathcal{V}$

for Player i if and only if every play $\pi \in \mathit{Plays}(\mathcal{A}, v)$ played according to ρ is winning for Player i. Strategies that do not need memory are called memoryless or positional. A strategy ρ for Player i in an arena $\mathcal{A}$ is positional if and only if $\rho(wv) = \rho(v)$ for all $w \in V^*$ and $v \in V_i$.

2.4. Synthesis and realizability

Reactive synthesis [26, 27, 28, 29, 30] is the problem of producing a system from an LTL specification, where the atomic propositions are split into propositions that are controlled by the environment and those that are controlled by the system. In other words, synthesis is the problem of computing a controller/system that chooses the value of the controllable variables in such a way to satisfy the LTL specification, no matter the values of the uncontrollable variables. Realizability is the related decision problem of deciding whether such a system exists.

Realizability of LTL specifications corresponds to infinite games: the work presented at [31] shows that we can construct a game $\mathcal{G}^{\mathbb{B}}$ from a specification $\varphi_{\mathbb{B}}$ and that $\varphi_{\mathbb{B}}$ is realizable if and only if $\mathcal{G}^{\mathbb{B}}$ is winning for the system. The positions in the arena of the game correspond to valuations of the atomic propositions. The arena of the game $\mathcal{A} = \langle V_E, V_S, \rightarrow \rangle$ is such that $V_E = (E \times S)^*$ whereas $V_S = (E \times S)^* \times E$ where E and S are the possible valuations of variables of the environment and the system. Thus, plays generated have the form: $\epsilon, e_0 s_0, e_0 s_0 e_1 s_1, ...$, where $e_0, e_1, ... \in E$ and $s_0, s_1, ... \in S$. A play π is winning for the system if $\pi \models \varphi_{\mathbb{B}}$.

For convenience we use the following alternative definition of strategy as a Mealy machine: $\mathcal{M} = \langle Q, q_0, \Sigma = \{E, S\}, \delta : Q \times E \rightarrow Q, o : Q \times E \rightarrow S \rangle$, where Q is a set of positions, q_0 is the initial position, Σ is the alphabet composed of the possible valuations of the environment E and the possible valuations of the system S, δ is the transition function of the strategy and o is the output function:

- Q is the set of positions of the environment, that is V_E.

- q_0 is the empty history, as ϵ.
- Σ is the same in both definitions.
- $\delta : Q \times E \to Q$ is $\delta(v, e) = v \cdot (e, \rho(ve))$.
- $o : Q \times E \to S$ is $o(v, e) = \rho(ve)$.

It is easy to see that the set of plays played according to ρ and to $\mathcal{M}$ are the same set. Also, as for the alternative definition of strategy using ρ, memoryless strategies allow to remember only a finite amount of information in Q, instead of the full history.

3. LTL$_{\mathcal{T}}$

We consider in this paper the realizability problem for a theory $\mathcal{T}$. In this case, the alphabet is the possible valuations of the variables (from the domains of the theories) and the atomic propositions are instead literals in the corresponding theories. The syntax of linear temporal logic modulo theories (LTL$_{\mathcal{T}}$) is the same as in LTL, except that atoms a are now literals l from theory $\mathcal{T}$. We use $\mathit{Var}(l)$ for the variables in literal l and $\mathit{Var}(\varphi)$ for the union of variables occurring in the literals of φ. The alphabet of a formula φ is now $\Sigma_{\mathcal{T}} : \mathit{Var}(\varphi) \to D$, where D is the domain of the variables, that is, a letter is a valuation of the variables in the formlua. We use $\vDash_{\mathcal{T}}$ as the usual satisfaction relation of literals using using the usual interpretation of symbols in $\mathcal{T}$. The semantics of LTL$_{\mathcal{T}}$ associate traces $\sigma \in \Sigma_{\mathcal{T}}^{\omega}$ with formulae as follows:

$$
\begin{array}{lll}
\sigma \models T & \text{always} & \\
\sigma \models l & \text{iff} & \sigma(0) \vDash_{\mathcal{T}} l \\
\sigma \models \varphi_1 \vee \varphi_2 & \text{iff} & \sigma \models \varphi_1 \text{ or } \sigma \models \varphi_2 \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff} & \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \bigcirc\varphi & \text{iff} & \sigma^1 \models \varphi \\
\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \text{for some } i \geq 0 \;\; \sigma^i \models \varphi_2, \text{ and} \\
 & & \qquad \text{for all } 0 \leq j < i, \sigma^j \models \varphi_1
\end{array}
$$

Note that semantics of $\text{LTL}_{\mathcal{T}}$ is the same as for LTL, except that $\sigma \vDash l$ if and only if $\sigma(0) \vDash_{\mathcal{T}} l$ and that we can have other derived operations like $\mathcal{R}$.

In realizability for $\text{LTL}_{\mathcal{T}}$ specifications, the variables that occur in the literals in a specification φ are split into those variables controlled by the environment ($\overline{x}$) and those controlled by the system ($\overline{y}$). We use $\varphi(\overline{x}, \overline{y})$ to denote that $\overline{x} \cup \overline{y}$ are the variables occurring in φ (where $\overline{x} \cap \overline{y} = \emptyset$) The alphabet $\Sigma_{\mathcal{T}}$ is now a valuation of the variables in $\overline{x} \cup \overline{y}$. A trace is an infinite sequence of valuations of these variables, which induces an infinite sequence of Boolean values of the literals occurring in φ and ultimately a valuation of the formula.

Realizability for $\text{LTL}_{\mathcal{T}}$ now corresponds to a game with an infinite arena and, like in LTL, given a formula $\varphi_{\mathcal{T}}$ in $\text{LTL}_{\mathcal{T}}$ we construct a game $\mathcal{G}^{\mathcal{T}}$ (with an arena being the secuence of valuations chosen and the $\varphi_{\mathcal{T}}$ as winning condition), such that $\varphi_{\mathcal{T}}$ is realizable if and only if $\mathcal{G}^{\mathcal{T}}$ is winning for the system player. Positions of $\mathcal{G}^{\mathcal{T}}$ may have infinitely many successors if the ranges of the variables controlled by the system and the environment are infinite. For instance, in Example 1.1 with $\mathcal{T} = \mathcal{T}_{\mathbb{Z}}$, valuation ranges over infinite values, and literal $(x \geq 2)$ can be satisfied with $x = 2$, $x = 3$, etc.

4. Boolean Abstraction

In this section, we solve the realizability problem modulo theories by transforming the specification into an equi-realizable LTL specification that only contains purely Boolean literals. Given a specification φ with literals l_i, we produce a new specification $\varphi_{\mathbb{B}} = \varphi'' \wedge \square(A_{\mathbb{B}} \rightarrow \varphi^{extra})$, where s_i are fresh Boolean variables, φ^{extra} is a Boolean formula without temporal operators described in this section and $A_{\mathbb{B}}$ is a conjunction of assumptions of the environment that we later study. The additional sub-formula φ^{extra} uses the freshly introduced variables s_i controlled by the system, as well as additional Boolean variables controlled by the environment e_i, which captures the precise combined power of the players to decide the valuations of the literals in the original formula. We call our approach *Booleanization* or *Boolean abstraction*. The approach is

$$\begin{tikzcd}
\varphi_{\mathcal{T}} \arrow[r, "\text{Booleanization}"] \arrow[d, dotted, leftrightarrow, "\models"'] & \varphi_{\mathbb{B}} \arrow[r] \arrow[d, dotted, leftrightarrow, "\models"'] & \boxed{\text{Realizability Tool}} \\
\mathcal{G}_{\mathcal{T}} \arrow[r, dotted, leftrightarrow, "\text{Thm. 1}"] & \mathcal{G}_{\mathbb{B}} &
\end{tikzcd}$$

Figure 1: The tool chain with the correctness argument.

summarized in Fig. 1: given an LTL$_{\mathcal{T}}$ specification $\varphi_{\mathcal{T}}$, it is translated into a Boolean $\varphi_{\mathbb{B}}$ which can be analyzed with off-the-shelf realizability checkers.

The formula $\varphi_{\mathbb{B}}$ obtained after Booleanization captures that after the environment chooses a valuation of the variables it controls (including e_i), the system responds with valuations of its variables (including s_i) inducing a Boolean value for all literals. For each possible choice of the environment, the system has the power to choose a concrete combination of Boolean values for the literals among a collection of possibilities. Since the set of all possible responses is finite, so are the different collections of outcomes to choose from after each environment move. The extra requirement captures precisely the finite collection of choices of the environment and the resulting finite collection of responses for each case.

4.1. Preliminary definitions

In order to describe the construction of the extra requirement, we introduce some preliminary definitions. We will use Example 1.1 as the running example.

Let $Lit(\varphi)$ be the collection of literals that appear in φ (or Lit, if the formula is clear from the context). For simplicity, we assume that all literals belong to the same theory, but each theory can be Booleanized in turn, as each literal belongs to exactly one theory and we assume in this paper that literals from different theories do not share variables. Note that we use $\overline{x}$ as the environment controlled variables occurring in $Lit(\varphi)$ and $\overline{y}$ for the variables controlled by the system.

Example 4.1. *Recall Example 1.1, where we had formula* $\varphi = \Box R_0 \wedge R_1$*, where* $R_0 : (x > 2) \wedge \bigcirc(y > 1)$ *and* $R_1 : (x \leq 2) \wedge (y < x)$*. Then, we first translate the* *literals in φ. Since $(x < 2)$ is equivalent to $\neg(x \geq 2)$, we use a single Boolean variable for both. The substitutions is:*

$$\begin{array}{lll} (x < 2) \leftarrow s_0 & (y > 1) \leftarrow s_1 & (y < x) \leftarrow s_2 \\ (x \geq 2) \leftarrow \neg s_0 & (y \leq 1) \leftarrow \neg s_1 & (y \geq x) \leftarrow \neg s_2 \end{array}$$

After the substitution we obtain $\varphi'' = \Box(R_0^{\mathbb{B}} \wedge R_1^{\mathbb{B}})$ where

$$R_0^{\mathbb{B}} : s_0 \rightarrow \bigcirc s_1 \qquad R_1^{\mathbb{B}} : \neg s_0 \rightarrow s_2$$

Note that φ'' may not be equi-realizable to φ, as we may be giving too much power to the system if s_0, s_1 and s_2 can be chosen independently without restriction. Of course, this is an over-approximation of the decision power of the system that is later on corrected with the computation of an additional formula. In this case φ'' is realizable, for example by always choosing s_1 and s_2 to be true, but as we saw in Example 1.1, φ is not realizable for $LTL_{\mathcal{T}_{\mathbb{Z}}}$. $\Box$

Definition 1 (Choice). *A choice $c \subseteq Lit(\varphi)$ is a subset of the literals of φ.*

The intended meaning of a choice is to capture what literals are true (after a combined move of the environment and the system), while the rest (i.e., $Lit \setminus c$) are false. Once the environment picks values for $\overline{x}$, the system can exercise some choice c by selecting $\overline{y}$ and making the literals in c true and the rest false. However, for some values of $\overline{x}$, some choices may not be possible for the system for any $\overline{y}$. Given a choice c, we use $f_c(\overline{x}, \overline{y})$ to denote the formula:

$$\bigwedge_{l \in c} l \wedge \bigwedge_{l \notin c} \neg l$$

which is a formula in theory $\mathcal{T}$ over variables $\overline{x}$ and $\overline{y}$ that captures logically the set of values of $\overline{x}$ and $\overline{y}$ that realize precisely choice c. We use $\mathcal{C}$ for the set of choices. Note that there are $|\mathcal{C}| = 2^{|Lit|}$ different choices.

As we will see, a given choice c can act as *potential* (meaning that the response is possible) or as *antipotential* (meaning that the response is not possible). Formally, a potential is a formula that depends only on $\overline{x}$ and captures those values of $\overline{x}$ for which the system can respond and make precisely the literals in c true (and the rest of the literals false). The negation of the potential captures precisely those values of $\overline{x}$ for which there are no values of $\overline{y}$ that lead to c.

Definition 2 (Potential and Antipotential). *Given a choice c, a potential is the following formula c^p and a antipotential is the following formula c^a:*

$$c^p(\overline{x}) = \exists \overline{y}. f_c(\overline{x}, \overline{y}) \qquad\qquad c^a(\overline{x}) = \forall \overline{y}. \neg f_c(\overline{x}, \overline{y})$$

Example 4.2. *We illustrate two choices for Example 1.1. Consider choices $c_0 = \{(x < 2), (y > 1), (y < x)\}$ and $c_1 = \{(x < 2), (y > 1)\}$. Choice c_0 corresponds to $f_{c_0} = (x < 2) \wedge (y > 1) \wedge (y < x)$, that is literals $(x < 2)$, $(y > 1)$ and $(y < x)$ are true.*

Choice c_1 corresponds to $f_{c_1} = (x < 2) \wedge (y > 1) \wedge (y \geq x)$, that is literals $(x < 2)$, $(y > 1)$ are true and $(y < x)$ is false. The potential and antipotential formulae of choices c_0 and c_1 from Example 1.1 are:

$$\begin{array}{ll} c_0^p = \exists y.(x < 2) \wedge (y > 1) \wedge (y < x) & c_0^a = \forall y.\neg\big((x < 2) \wedge (y > 1) \wedge (y < x)\big) \\ c_1^p = \exists y.(x < 2) \wedge (y > 1) \wedge (y \geq x) & c_1^a = \forall y.\neg\big((x < 2) \wedge (y > 1) \wedge (y \geq x)\big) \end{array}$$

c_0^p represents that there is a y such that f_{c_0}, while c_0^a states the contrary. Note that potentials and antipotentials have $\overline{x}$ as the only free variables. Also, the rest of the choices are as follows:

$$\begin{array}{ll} c_2^p = \exists y.(x < 2) \wedge (y \leq 1) \wedge (y < x) & c_2^a = \forall y.\neg\big((x < 2) \wedge (y \leq 1) \wedge (y < x)\big) \\ c_3^p = \exists y.(x < 2) \wedge (y \leq 1) \wedge (y \geq x) & c_3^a = \forall y.\neg\big((x < 2) \wedge (y \leq 1) \wedge (y \geq x)\big) \\ c_4^p = \exists y.(x \geq 2) \wedge (y > 1) \wedge (y < x) & c_4^a = \forall y.\neg\big((x \geq 2) \wedge (y > 1) \wedge (y < x)\big) \\ c_5^p = \exists y.(x \geq 2) \wedge (y > 1) \wedge (y \geq x) & c_5^a = \forall y.\neg\big((x \geq 2) \wedge (y > 1) \wedge (y \geq x)\big) \\ c_6^p = \exists y.(x \geq 2) \wedge (y \leq 1) \wedge (y < x) & c_6^a = \forall y.\neg\big((x \geq 2) \wedge (y \leq 1) \wedge (y < x)\big) \\ c_7^p = \exists y.(x \geq 2) \wedge (y \leq 1) \wedge (y \geq x) & c_7^a = \forall y.\neg\big((x \geq 2) \wedge (y \leq 1) \wedge (y \geq x)\big) \end{array}$$

□

Depending on the theory, the meaning and validity of potentials and antipotentials may be different. For instance, consider c_0^p and theories $\mathcal{T}_{\mathbb{Z}}$ and $\mathcal{T}_{\mathbb{R}}$:

- In $\mathcal{T}_{\mathbb{Z}}$: $\exists y.(x < 2) \wedge (y > 1) \wedge (y < x)$ is equivalent to *false*.
- In $\mathcal{T}_{\mathbb{R}}$: $\exists y.(x < 2) \wedge (y > 1) \wedge (y < x)$ is equivalent to $(x < 2)$.

These equivalences can be obtained by Cooper's algorithm [10] for $\mathcal{T}_{\mathbb{Z}}$ and Tarski's method [16] for $\mathcal{T}_{\mathbb{R}}$.

We introduce now a reaction as a description of the specific choices that the system has the power to select.

Definition 3 (Reaction). *Let P and A be a partition of the set of choices $\mathcal{C}$ (that is, $P \subseteq \mathcal{C}$, $A \subseteq \mathcal{C}$, $P \cap A = \emptyset$ and $P \cup A = \mathcal{C}$). This partition (P, A) is called a reaction r. The formula associated to reaction r is F_r:*

$$F_r(\overline{x}) \overset{def}{=} \bigwedge_{c \in P} c^p \wedge \bigwedge_{c \in A} c^a = \bigwedge_{c \in P} \exists \overline{y}. f_c(\overline{x}, \overline{y}) \wedge \bigwedge_{c \in A} \forall \overline{y}. \neg f_c(\overline{x}, \overline{y})$$

There are $2^{2^{|Lit|}}$ different reactions.

A reaction r is called valid whenever there is a move of the environment for which r captures precisely the remaining power of the system. Formally, a reaction is valid if and only if $\exists \overline{x}.\ F_r(\overline{x})$ is a valid formula. We use $\mathcal{R} = \{(P, A) | P, A \subseteq C \textit{ and } P \cap A = \emptyset \textit{ and } P \cup A = C\}$ for the set of reactions and $VR = \{r \in \mathcal{R} | \exists \overline{x}. F_r(\overline{x}) \textit{ is valid}\}$ for the set of valid reactions. It is easy to see that, for any possible valuation of $\overline{x}$ the environment can pick, the system has a specific power to respond (among the finitely many cases), i.e., a set of potential choices. This is formalized and proven in Theorem 2.

Example 4.3. *In Example 1.1, for theory $\mathcal{T}_{\mathbb{Z}}$, we find there are two valid reactions:*

$$\begin{aligned} r_1 &: \exists x. c_0^a \wedge c_1^p \wedge c_2^p \wedge c_3^p \wedge c_4^a \wedge c_5^a \wedge c_6^a \wedge c_7^a \\ r_2 &: \exists x. c_0^a \wedge c_1^a \wedge c_2^a \wedge c_3^a \wedge c_4^a \wedge c_5^p \wedge c_6^p \wedge c_7^a, \end{aligned}$$

where reaction r_1 models the possible responses of the system after the environment picks a value for x such that $(x < 2)$, and r_2 models the responses to $(x \geq 2)$. On the other hand, for $\mathcal{T}_{\mathbb{R}}$, there are three valid reactions:

$$\begin{aligned} r_1 &: \exists x. c_0^a \wedge c_1^p \wedge c_2^p \wedge c_3^p \wedge c_4^a \wedge c_5^a \wedge c_6^a \wedge c_7^a \\ r_2 &: \exists x. c_0^p \wedge c_1^p \wedge c_2^p \wedge c_3^a \wedge c_4^a \wedge c_5^a \wedge c_6^a \wedge c_7^a \\ r_3 &: \exists x. c_0^a \wedge c_1^a \wedge c_2^a \wedge c_3^a \wedge c_4^p \wedge c_5^p \wedge c_6^p \wedge c_7^a \end{aligned}$$

Note that there is one additional valid reaction, since in $\mathcal{T}_{\mathbb{R}}$ there is one more case: $x \in (1, 2]$. Also, note that c_4 cannot be a potential in $\mathcal{T}_{\mathbb{Z}}$ (not even with a collaboration between environment and system), whereas it can in $\mathcal{T}_{\mathbb{R}}$. □

4.2. The Boolean Abstraction Algorithm

Boolean abstraction is a method to compute $\varphi_{\mathbb{B}}$ from $\varphi_{\mathcal{T}}$. In this section we describe the method and in Section 5 we prove correct a basic exhaustive algorithm of this method. This algorithm first computes the extra requirement, by visiting exhaustively the set of reactions and computing a subset of the valid reactions that guarantees to preserve realizability. Finally, after the loop, the algorithm produces a φ^{extra} formula as a conjunction of cases, one per valid reaction $r : (P, A)$ in VR. For creating this formula, we introduce a fresh variable e_r, controlled by the environment for each valid reaction r, to capture that the environment plays values for $\overline{x}$ that correspond to the case where the system is left precisely with the power to choose captured by r. Therefore, there is one additional environment Boolean variable per valid reaction (in practice we can enumerate the number of valid reactions and introduce only a logarithmic number of environment variables). Each e_r represents a specific power of the environment to let the system select a choice from reaction r.

Finally, the extra requirement takes the potentials P for each valid reaction $r : (P, A)$ to encode the potential moves of the system as a disjunction of the literals described by each choice in P. Each of these disjunction contains precisely the combinations of literals that are possible for the concrete case that (P, A) captures.

An algorithm that implements the Boolean abstraction method by exhaustively searching all reactions is shown in Algorithm 1. The building blocks of this algorithm are:

(1) It stops when the remaining set of reactions is empty (line 6).

(2) It traverses the set $\mathcal{R}$ according to some predetermined order (line 7). Note

Algorithm 1: Boolean abstraction

```
1  **Input:** $\varphi_{\mathcal{T}}$
2  $\varphi' \leftarrow \varphi_{\mathcal{T}}[l_i \leftarrow s_i]$
3  $VR \leftarrow \{\}$
4  $\mathcal{C} \leftarrow choices(Lit(\varphi_{\mathcal{T}}))$
5  $\mathcal{R} \leftarrow 2^{\mathcal{C}}$
6  while $\mathcal{R} \neq \emptyset$ do
7      $r \leftarrow pickelem(\mathcal{R})$
8      if $\exists \overline{x}. F_r(\overline{x})$ is valid then
9          $VR \leftarrow VR \cup \{r\}$
10     $\mathcal{R} \leftarrow \mathcal{R} \setminus \{r\}$
11 $\varphi^{extra} \leftarrow getExtra(VR)$
12 $A_{\mathbb{B}} \leftarrow getAssum(VR)$
13 return $\varphi_{\mathbb{B}} = \varphi'' \wedge \Box(A_{\mathbb{B}} \rightarrow \varphi^{extra})$
```

that, even though the order may affect performance, it does not affect soundness since all relevant reactions will be inevitably explored.

(3) To modify the set of valid reactions, if $r : (P, A)$ is valid it adds r to the set VR (line 9). To modify the set of remaining reactions, it removes r from the search (line 10).

Finally, the extra sub-formula φ^{extra} is generated by *getExtra* (line 11) defined as follows:

$$getExtra(VR) = \bigwedge_{(P,A)\in VR} (e_{(P,A)} \rightarrow \bigvee_{c\in P} (\bigwedge_{l_i\in c} s_i \wedge \bigwedge_{l_i\notin c} \neg s_i)),$$

Note that there is an $\exists^*\forall^*$ validity query in the body of the loop (line 6) to check whether the candidate reaction is valid. This is why decidability of the $\exists^*\forall^*$ fragment is crucial because it captures the finite partitioning of the environment moves (which is existentially quantified) for which the system can react in certain ways (i.e., potentials, which are existentially quantified) by picking appropriate valuations but not in others (i.e., antipotentials, which are universally quantified) This exhaustive algorithm iterates over all the reactions, one at a time, checking whether each reaction is valid or not. In case the reaction (characterized by the set of potential choices) is valid, it is added to VR.

The set $\mathcal{D} = \{e_r | r \in VR\}$ contains the Boolean variables for the environment, one for each reaction. We will use the function $dec : VR \rightarrow \mathcal{D}$, which receives a reaction r and returns its decision variable, and its inverse function $dec^{-1} : \mathcal{D} \rightarrow VR$, which takes a decision variable e_r and returns its reaction r. Also, function $pots : VR \rightarrow 2^C$, given $r : (P, A)$ returns its set of potentials P. The general form of the Boolean abstraction encoding (line 12) is as follows:

$$\varphi_{\mathbb{B}} = \varphi'' \wedge \square(A_{\mathbb{B}} \rightarrow \varphi^{extra})$$

where

$$A_{\mathbb{B}} = (\bigwedge_{e_r, e_s \in \mathcal{D}} e_r \leftrightarrow \neg e_s) \wedge (\bigvee_{e_r \in \mathcal{D}} e_r)$$

and

$$\varphi^{extra} = \bigwedge_{e_r \in \mathcal{D}} e_r \rightarrow \bigvee_{r} pots(r) \text{ where } r = dec^{-1}(e_r)$$

Formula $A_{\mathbb{B}}$ simply forces the environment to pick exactly one decision e_r at each point in time, so we can interpret $A_{\mathbb{B}}$ as the assumptions that the environment player has to hold. Note that we calculate formula $A_{\mathbb{B}}$ with $getAssum(VR)$ in Algorithm 1, where $getAssum(VR)$ uses $dec(VR)$ to get each $e_k \in \mathcal{D}$.

Example 4.4. *Consider again the specification in Example 1.1, with $\mathcal{T}_{\mathbb{Z}}$ as theory. Note that the valid reactions are r_1 and r_2, as shown in Example 4.3, where the potentials of r_1 are $\{c_1, c_2, c_3\}$ and the potentials of r_2 are $\{c_5, c_6\}$. Now, the creation of φ^{extra} requires two fresh variables e_0 and e_1 for the environment (they correspond to environment decisions x such that $(x < 2)$ and $(x \geq 2)$,* *respectively), resulting into:*

$$\varphi^{extra}_{\mathcal{T}_\mathbb{Z}} : \left(\begin{array}{ccl} e_0 & \rightarrow & \big((s_0 \wedge s_1 \wedge \neg s_2) \vee (s_0 \wedge \neg s_1 \wedge s_2) \vee (s_0 \wedge \neg s_1 \wedge \neg s_2)\big) \\ & \wedge & \\ e_1 & \rightarrow & \big((\neg s_0 \wedge s_1 \wedge \neg s_2) \vee (\neg s_0 \wedge \neg s_1 \wedge s_2)\big) \end{array} \right)$$

For example $c_2 = \{s_0\}$ is a choice that appears as potential in valid reaction r_1, so it appears as a disjunct of e_0 as $(s_0 \wedge \neg s_1 \wedge \neg s_2)$. $\square$

Note that $A_\mathbb{B}$ is not used with enumerated decisions (i.e., of an *enumerated* type), so the general form with enumerations is $\varphi_\mathbb{B} = \varphi'' \wedge \square\varphi^{extra}$, which is not the encoding we used for Algorithm 1. Indeed we usually consider the e_r to be of enumerated type, so that the number of bits needed to express decisions is logarithmic in the cardinality of the enum type. Thus, the complete Booleanized specification of Example 1.1 using enumerations is $\varphi^\mathbb{B}_{\mathcal{T}_\mathbb{Z}} = \varphi'' \wedge \varphi^{extra}_{\mathcal{T}_\mathbb{Z}}$. Note that it is straigthforward to obtain Boolean counterparts from enum types.

Moreover, we find some simplifications of $\varphi^\mathbb{B}_{\mathcal{T}_\mathbb{Z}}$ in detail in Example 4.5.

Example 4.5. *We revisit Example 4.4. Since there are only two decisions of the environment (namely, e_0 and e_1), we can represent them with a single bit e (i.e., $e \equiv e_0$ and $\neg e \equiv e_1$), and eliminate the mutual exclusion $e_0 \leftrightarrow \neg e_1$. Also, since $e \equiv e_0$ implies s_0 and $\neg e \equiv e_1$ implies $\neg s_0$, we can get rid of s_0. We can continue performing other propositional logic simplifications to get an even smaller specification: e.g., $((s_1 \wedge \neg s_2) \vee (\neg s_1 \wedge s_2) \vee (\neg s_1 \wedge \neg s_2))$ is equivalent to $(\neg s_1 \vee \neg s_2)$. The resulting $\varphi^\mathbb{B}_{\mathcal{T}_\mathbb{Z}}$ is:*

$$\left(\begin{array}{ccl} e & \rightarrow & \bigcirc s_1 \\ & \wedge & \\ \neg e & \rightarrow & s_2 \\ & \wedge & \\ e & \rightarrow & \big(\neg s_1 \vee \neg s_2\big) \\ & \wedge & \\ \neg e & \rightarrow & \big((s_1 \wedge \neg s_2) \vee (\neg s_1 \wedge \neg s_2)\big) \end{array} \right),$$

where the first two implications correspond to φ'' and the last two to $\varphi^{extra}_{\mathcal{T}_\mathbb{Z}}$. □

We will show later that the resulting $\varphi_\mathbb{B}$ is a Boolean equi-realizable version of $\varphi_\mathcal{T}$. The complexity of the exhaustive Booleanization algorithm is doubly exponential in the number of literals, with one $\exists^*\forall^*$ query in the body of the loop to check whether the candidate reaction is valid.

The extra requirement encodes precisely all reactions; that is, collections of choices, for which there is a move of the environment that leaves the system with precisely that power to respond. Therefore, at each position in the realizability game, the system can respond to moves of the system leaving to the corresponding positions in the Boolean game. In turn, this leads to equi-realizability because each move can be simulated in the corresponding game. This argument is formally developed in Section 5. We first prove some intermediate properties.

4.3. Properties of VR and φ^{extra}

A Booleanization $\varphi_\mathbb{B}$ from $\varphi_\mathcal{T}$ contains a φ^{extra} that has a set of environment decisions $e_i \in \mathcal{D}$ and a set of choices $c \in \mathcal{C}$ for each e_i. Note, again, that $\mathcal{R}$ is the set of reactions, whereas VR is the set of valid reactions and $VR \subseteq \mathcal{R}$. Let $\overline{x}$ and $\overline{y}$ be variables from theory $\mathcal{T}$ and $\overline{v}$ and $\overline{w}$ values from $\mathcal{T}$. We now state some important properties of $\mathcal{R}$ and VR.

Lemma **1.** *For every valuation $\overline{v}$ of the variables $\overline{x}$, there is at least one reaction r such that $F_r[\overline{x} \leftarrow \overline{v}]$ is valid. Therefore, the formula $\varphi_\mathcal{R} = \forall \overline{x}. \bigvee_{r_i \in \mathcal{R}} F_{r_i}$ is valid.*

Proof:. Let $\overline{v}$ be an arbitrary valuation of the variables $\overline{x}$ and let $C = \{c \in \mathcal{C} | I[\overline{x} \leftarrow \overline{v}] \vDash \exists \overline{y}.c(\overline{x}, \overline{y}))\}$ where I is an arbitrary interpretation. It follows that $I[\overline{x} \leftarrow \overline{v}] \vDash F_r$, since for every $c \in C$ then $I[\overline{x} \leftarrow \overline{v}] \vDash f_c$ and for any $c \notin C$ then $I[\overline{x} \leftarrow \overline{v}] \nvDash f_c$. Therefore, $F_r[\overline{x} \leftarrow \overline{v}]$ is valid. □

Lemma 2 states that there is always a valid move of the system (the extra requirement is never blocking). For every move of the environment, the system can move according to at least with one of the valid reactions.

Lemma **2.** *$\varphi_{VR} = \forall \overline{x}. \bigvee_{r_i \in VR} F_{r_i}$ is a valid formula.*

Proof:. Take an arbitrary interpretation I and valuation $\overline{v}$ of the $\overline{x}$ variables. By Lemma 1, $\varphi_{\mathcal{R}}$ is valid so $I[\overline{x} \leftarrow \overline{v}] \vDash F_{r_i}$ holds and hence there is a reaction r_i such that $I[\overline{x} \leftarrow \overline{v}] \vDash F_{r_i}$. Thus $\exists \overline{x}.\ F_{r_i}(\overline{x})$ is valid, which means $r_i \in VR$. Consequently, $I[\overline{x} \leftarrow \overline{v}] \vDash \bigvee_{r \in VR} F_{r_i}$. Since I and $\overline{v}$ are arbitrary, $\forall \overline{x}. \bigvee_{r_i \in VR} F_{r_i}$ is valid, as desired. □

We now show that, in the extra requirement, the set of potentials in valid reactions cannot be empty (see Lemma 3). In other words, for every move of the environment the system can always respond with a valid reaction, which contains at least one potential choice. This is stated in Lemma 3.

Lemma **3.** *Let $P \in \mathcal{C}$ be such that $r : (P, \mathcal{C} \setminus P) \in VR$. Then $P \neq \emptyset$.*

Proof:. Let $\overline{v}$ be such that $F_r[\overline{x} \leftarrow \overline{v}]$ is valid. Let $\overline{w}$ be an arbitrary valuation of $\overline{y}$ and let c be a choice and l a literal. Therefore, the following holds:

$$\bigwedge_{l[\overline{x} \leftarrow \overline{v}, \overline{y} \leftarrow \overline{w}] \ is\ true} l \wedge \bigwedge_{l[\overline{x} \leftarrow \overline{v}, \overline{y} \leftarrow \overline{w}] \ is\ false} \neg l$$

It follows that $I[\overline{x} \leftarrow \overline{v}] \models \exists \overline{y}.P$, so $P \in C$. □

Lemma 3 is crucial because it ensures that once Algorithm 1 is executed, for each fresh e_r variable in the extra requirement that corresponds to a decision of the environment, at least one reaction with one or more potentials can be responded by the system.

Also, it is easy to see that Lemma 4 holds.

Lemma **4.** *Let $r_1, r_2 \in VR$ such that $r_1 \neq r_2$. Then $\forall \overline{x}.\ [F_{r_1}(\overline{x}) \rightarrow \neg F_{r_2}(\overline{x})]$*

5. Correctness

The main element of the proof of correctness is that the system player wins the game $\mathcal{G}^{\mathcal{T}}$ for $\varphi_{\mathcal{T}}$ if and only if the system player wins the game $\mathcal{G}^{\mathbb{B}}$ for $\varphi_{\mathbb{B}}$. This is proven by creating a binary relation $\sim$ between the positions of the games showing (1) that the valuations of atomic propositions correspond to equivalent valuations of literals, and (2) that moves in one game can be simulated to corresponding moves in the other game (and vice-versa) leading to states related by $\sim$. This, in turn, induces a bijective map between winning strategies in one game and in the other, concluding that the system player wins the game for $\varphi_{\mathcal{T}}$ if and only if the system player wins the game for $\varphi_{\mathbb{B}}$.

Definition 4 (Simulation $\sim$). *Let $V_{\mathbb{B}}$ be the set of positions of $\mathcal{G}^{\mathbb{B}}$ and $V_{\mathcal{T}}$ the set of positions of $\mathcal{G}^{\mathcal{T}}$. Then, we define a simulation $V_{\mathcal{T}} \sim V_{\mathbb{B}}$, that relates positions v_t of $V_{\mathcal{T}}$ and v_b of $V_{\mathbb{B}}$ as follows:*

(1) each literal l_i and the corresponding variable s_i have the same truth value in v_t and v_b.

(2) the extra requirement is satisfied in v_b.

The following lemma holds from the definition.

Lemma **5.** *Let π be a play in $\mathcal{G}^{\mathbb{B}}$ and π' a trace in $\mathcal{G}^{\mathcal{T}}$ such that for all i, $\pi(i) \sim \pi'(i)$. Then, π is winning for the system if and only if π' is winning for the system.*

Proof:. Since $\pi(i) \vDash l_i$ if and only if $\pi'(i) \vDash s_i$, this means that valuations of atomic propositions and literals are the same at corresponding positions. It follows that every temporal formula has the same valuation in π than in π'. □

We now show that a system move from related positions can be mimicked in the other game by the system player leading to related positions, which is the main result of this paper.

Theorem **1.** *The system player wins $\mathcal{G}^{\mathcal{T}}$ if and only if the system player wins $\mathcal{G}^{\mathbb{B}}$. Therefore, $\varphi_{\mathcal{T}}$ is realizable if and only if $\varphi_{\mathbb{B}}$ is realizable.*

Proof:. During the proof we will use the following functions:

- $getV : VR \rightarrow E_{\mathcal{T}}$ which receives a reaction r and returns a valuation of the environment variables $\overline{v}$ that satisfies F_r. This is guaranteed to exist becuse r is a valid reaction.

- $getVR : E_{\mathcal{T}} \to VR$, which receives a valuation of the environtment variables $\overline{v}$ and returns the only reaction r for which $F_r(\overline{v})$ holds.
- $getC : E_{\mathcal{T}} \times S_{\mathcal{T}} \to C$ (it stands from *get choice*), which receives $\overline{v}$ and system valuation $\overline{u}$ and returns the resulting choice c.

We now prove the two directions separately.

- "$\Rightarrow$". We assume that the system player wins $\mathcal{G}^{\mathcal{T}}$ and show that the system player wins $\mathcal{G}^{\mathbb{B}}$. Let $\mathcal{M}^{\mathcal{T}} = \langle Q^{\mathcal{T}}, q_0^{\mathcal{T}}, \Sigma_{\mathcal{T}} = \{E_{\mathcal{T}}, S_{\mathcal{T}}\}, \delta_{\mathcal{T}}, o_{\mathcal{T}}\rangle$ be a winning strategy for the system in $\mathcal{G}^{\mathcal{T}}$. We construct a winning strategy for the system in $\mathcal{G}^{\mathbb{B}}$ as follows $\mathcal{M}^{\mathbb{B}} = \langle Q^{\mathbb{B}}, q_0^{\mathbb{B}}, \Sigma_{\mathbb{B}} = \{E_{\mathbb{B}}, S_{\mathbb{B}}\}, \delta_{\mathbb{B}}, o_{\mathbb{B}}\rangle$ where
  - $Q^{\mathbb{B}} = Q^{\mathcal{T}}$
  - $q_0^{\mathbb{B}} = q_0^{\mathcal{T}}$
  - $\delta_{\mathbb{B}}(q, d_i) = \delta_{\mathcal{T}}(q, \overline{v})$ where $\overline{v} = getV(dec^{-1}(d_i)))$
  - $o_{\mathbb{B}}(q, d_i) = \{s_i | l_i \in getC(\overline{v}, \overline{u})\}$, where $\overline{v}$ is as before and $\overline{u} = o_{\mathcal{T}}(q, getV(dec^{-1}(d_i)))$.

  We show now that $\mathcal{M}^{\mathbb{B}}$ is winning. Consider an arbitrary play π played according to $\mathcal{M}^{\mathbb{B}}$. Let π' be the play in $\mathcal{G}^{\mathcal{T}}$, played according to $\mathcal{M}^{\mathcal{T}}$ where, at position i, the environment plays $getV(dec^{-1}(d))$ if the environment plays d at position i in π. It is easy to see by induction on i, that forall i, $\pi(i) \sim \pi'(i)$. By Lemma 5, π is winning for the system player because π' is wining for the system player ($\mathcal{M}^{\mathcal{T}}$ is a winning strategy). Therefore, $\mathcal{M}^{\mathbb{B}}$ is a winning strategy.

- "$\Leftarrow$" Analogously, we show that if $\mathcal{G}^{\mathbb{B}}$ is winning, then $\mathcal{G}^{\mathcal{T}}$ is winning. Assume $\mathcal{G}^{\mathbb{B}}$ is winning, then it has a winning strategy $\mathcal{M}^{\mathbb{B}} = \langle Q^{\mathbb{B}}, q_0^{\mathbb{B}}, \Sigma_{\mathbb{B}} = \{E_{\mathbb{B}}, S_{\mathbb{B}}\}, \delta_{\mathbb{B}} : Q^{\mathbb{B}} \times E_{\mathbb{B}} \to Q^{\mathbb{B}}, o_{\mathbb{B}} : Q^{\mathbb{B}} \times E_{\mathbb{B}} \to S_{\mathbb{B}}\rangle$ using which we can construct a winning strategy of the system in $\mathcal{G}^{\mathcal{T}}$, i.e., $\mathcal{M}^{\mathcal{T}} = \langle Q^{\mathcal{T}}, q_0^{\mathcal{T}}, \Sigma_{\mathcal{T}} = \{E_{\mathcal{T}}, S_{\mathcal{T}}\}, \delta_{\mathcal{T}} : Q^{\mathcal{T}} \times E_{\mathcal{T}} \to Q^{\mathcal{T}}, o_{\mathcal{T}} : Q^{\mathcal{T}} \times E_{\mathcal{T}} \to S_{\mathcal{T}}\rangle$ as follows:
  - $Q^{\mathcal{T}} = Q^{\mathbb{B}}$, $q_0^{\mathcal{T}} = q_0^{\mathbb{B}}$.

- $\delta_{\mathcal{T}}(q, \overline{v}) = \delta_{\mathbb{B}}(q, dec(r))$, where $r = getVR(\overline{v})$ and where $r \in VR$ is the only r such that $F_r(\overline{v})$ (this is guaranteed by Lemma 4).
- $o_{\mathcal{T}}(q, \overline{v}) = \overline{u}$, where $f_c(\overline{v})$ with $[\overline{y} \leftarrow \overline{u}]$ holds in at least one $c \in P$, where $(P, A) : r$ in a $r \in VR$, where $r = getVR(\overline{v})$.

We show now that $\mathcal{M}^{\mathcal{T}}$ is winning. Consider an arbitrary play π' played according to $\mathcal{M}^{\mathcal{T}}$. Let π be the play in $\mathcal{G}^{\mathbb{B}}$, played according to $\mathcal{M}^{\mathbb{B}}$ where, at position i, the environment plays $d_i = dec(getVR(\overline{v}))$ if the environment plays $\overline{v}$ at position i in π'. It is easy to see by induction on i, that forall i, $\pi(i) \sim \pi'(i)$. By Lemma 5, π is winning for the system player because π' is wining for the system player ($\mathcal{M}^{\mathbb{B}}$ is a winning strategy). Therefore, $\mathcal{M}^{\mathcal{T}}$ is a winning strategy.

This finished the proof. □

Note that the following theorem follows immediately from of Thm. 1.

Theorem **2.** *Let $\mathcal{T}$ be a theory with a decidable $\exists^*\forall^*$-fragment. Then,* LTL$_{\mathcal{T}}$ *realizability is decidable.*

6. Automatically Checking Feasible Valid Reactions

6.1. Minimal and feasible reactions

We define in this section two sets of reactions that are smaller than VR but still guarantee that the corresponding Boolean formula obtained from the original formula by Boolean abstraction is equi-realizable.

Consider the following two reactions of Example 4.3:

$$r = \{c_0^a, c_1^p, c_2^p, c_3^p, c_4^a, c_5^a, c_6^a, c_7^a\} \text{ and } r' = \{c_0^a, c_1^p, c_2^p, c_3^a, c_4^a, c_5^a, c_6^a, c_7^a\}$$

It is easy to see that r gives strictly more power to the system than r', since r differs in that r contains c_3 as potential but r' contains c_3 as anti-potential. That is, r captures those values of x for which the system can respond (with different values of y) making the literals according to c_1 true or the literals according to c_2 true, but the system cannot choose values of y for the rest of the choices. The reaction r corresponds to values of x for which the system has the additional power to choose values of y to make c_3 true. We denote this by $r' \sqsubseteq r$. Formally, $(P, A) \sqsubseteq (P', A')$ whenever $P \subseteq P'$. Also, in some cases two reactions r_i and r_j are not comparable, since neither contains the same or a strictly larger set of potentials than the other. For example, let $r = \{c_0^p, c_1^a, c_2^a, c_3^a, c_4^a, c_5^a, c_6^a, c_7^a\}$ and $r' = \{c_0^a, c_1^a, c_2^a, c_3^a, c_4^a, c_5^a, c_6^a, c_7^p\}$ are incomparable because $r_i \not\sqsubseteq r_j$ and $r_j \not\sqsubseteq r_i$.

In terms of the game strategy, the environment will never play valid reactions that are not lowest (e.g., if $r' \sqsubseteq r$, then it will always choose r'), since this move will give more power to the system in the form of more potential choices. In other words, if a strategy is winning for the environment and it chooses r at some position, then the modified strategy that chooses r' is also winning, because it allows strictly less moves to the system. Thus, for the φ^{extra} produced in Algorithm 1 it is not necessary to consider the whole set VR because any subset that preserves the minimal reactions, according to $\sqsubseteq$, preserves realizability. We call MVR the set with *minimal valid reactions*, defined as follows:

$$MVR = \{r \in VR \mid \textit{there is no } r' \in VR \textit{ such that } r' \sqsubseteq r\}$$

Given a set of reactions R we say that R is a *feasible valid reaction (FVR)* whenever $MVR \subseteq R \subseteq VR$. That is, a set of reactions R is feasible if all the reactions in R are valid and it contains at least all minimal reactions.

We can easily check whether a set of reactions R is indeed VR. This test can be used to check the correctness of the φ^{extra} of a $\varphi_{\mathbb{B}}$ Booleanized from a $\varphi_{\mathcal{T}}$. The check has two parts: checking that R is *legitimate* and *(strict) covering*, defined as follows.

Definition 5 (Legitimacy). *A set of reactions R is legitimate if and only if for all of $r \in R$, $\exists \overline{x}.\ F_r(\overline{x})$ is valid.*

If R is legitimate, then $R \subseteq VR$.

Definition 6 (Strict covering). *A set of reactions R is a strict covering if and only if the formula $\forall \overline{x}. \bigvee_{r \in R} F_r(\overline{x})$ is valid.*

The set VR itself generated by Algorithm 1 satisfies both legitimacy and strict covering. Covering refers to the fact that VR is a finite covering (hence the disjunction) of the possible plays of the environment; whereas strict cover implies that all possible finite plays of the environment are being covered, regardless of whether there are moves that a clever environment will never play because these moves leave more power to the system than better alternative moves.

Checking legitimacy and covering require a number of queries that is linear in the number of reactions in R. Thus, checking whether a set of reactions is VR is much more efficient than producing VR using Algorithm 1.

6.2. Checking Feasability

We can also easily check whether a set of reactions R is a feasible set. For example, this can be used in practice to check the correctness of a φ^{extra} produced by an algorithm that, unlike Algorithm 1 does not traverse the whole $\mathcal{R}$. Like for VR, R must be legitimate to be a feasible reaction set. However, strict covering is not required for R to be feasible since there are valid reactions that are not minimal.

The notion of *non-strict* covering considers not all the possible moves in the game, but only those that are optimal for the environment. For instance, if $r' \sqsubseteq r$, a feasible reaction set may or may not contain r. Non-strict covering can be evaluated by checking that for R, for all $\overline{x}$, the disjunction of the **potentials** of its reactions holds.

Definition 7. *The potential formula F_r^P of a reaction $r : (P, A)$ is:*

$$F_r^P(\overline{x}) \overset{def}{=} \bigvee_{c \in P} c^p(\overline{x}),$$

Definition 8 (covering). *A set of reactions R is covering if and only if $\varphi_{cov}(R) = \forall \overline{x}.\ \bigvee_{r \in R} F_r^P(\overline{x})$ is valid.*

If $\varphi_{cov}(R)$ is valid, then R contains a subset of valid reactions that makes R a covering in the game theoretic sense that it considers all the clever strategies for the environment. If the environment has a winning strategy in $\mathcal{G}^{\mathcal{T}}$ where it chooses values for $\overline{y}$ that correspond to moves that are not the minimal, then the strategy that chooses a strictly smaller move is also winning. The following Theorem 3 states this formally.

Theorem **3.** *Let $\varphi_{\mathcal{T}}$ be an* LTL$_{\mathcal{T}}$ *specification, R be a feasible reaction set and $\varphi_{\mathbb{B}} = \varphi'' \wedge \Box(A_{\mathbb{B}} \rightarrow getExtra(R))$ (as in Algorithm 1). Then $\varphi_{\mathcal{T}}$ and $\varphi_{\mathbb{B}}$ are equi-realizable.*

Proof:. Consider $\varphi_{\mathbb{B}}$ created from VR and let $R \subset VR$ be a feasible reaction. This means that there is $r \in VR \setminus R$ such that there is an $r' \in R$ with $r' \sqsubseteq r$. Consider a play of $\mathcal{G}^{\mathbb{B}}$ where the environment plays the move corresponding r at some point. If the move is replaced by r', then if the play with r was winning for the environment then the play with r' is also winning for the environment. Therefore, if $\mathcal{G}^{\mathbb{B}}$ is winning for the environment, there is a strategy for the environment where r is never played. It follows that the $\varphi'_{\mathbb{B}}$ that results from R is equi-realizable to the $\varphi_{\mathbb{B}}$ that results from VR. As a corollary, if we have $MVR \subset VR$, the environment will always play $r \in MVR$. $\Box$

In [32] Alg.2 and Alg.3 guarantee non-strict covering (i.e., they provide a FVR), whereas Alg.1 guarantees strict-covering (i.e. it provides VR). Also, no algorithm in [32] guarantees the computation of a MVR.

Moreover, since the equi-realizability between $\varphi_{\mathcal{T}}$ and $\varphi_{\mathbb{B}}$ is guaranteed by the sub-formula φ^{extra} of $\varphi_{\mathbb{B}}$ (which is characterized by its set of MVR), then Definition. 5 and Definition. 8 yield the basis for the verification algorithm suggested at the beginning of 6.2. Concretely, the fact that we can efficiently check covering and legitimacy of reactions allows designing algorithms with candidate sets of reactions. For instance, consider we have some oracle that returns a set of reactions R (this is a **guess** phase). Then, we can **check** whether $R \in$ FVR and stop or find additional valid reactions to complement R. For example, if a specification is refined by adding additional requirements or by changing requirements, a previously computed R for the previous specification is a reasonable candidate for FVR of the modified specification.

To better illustrate this, we describe the following **guess-and-check** algo-

Algorithm 2: Guess-and-check method

```
1 $\mathcal{U} \leftarrow \mathcal{R}$
2 $R \leftarrow \emptyset$
3 repeat
4     Guess $R_{new} \subseteq \mathcal{U}$ with $R_{new} \neq \emptyset$
5     $R \leftarrow R \cup (R_{new} \cap VR)$
6     $\mathcal{U} \leftarrow \mathcal{U} \setminus R_{new}$
7 until ($R$ is $FVR$);
8 return $R$
```

rithm for searching R. In Algorithm 2 we potentially traverse the whole set of subsets of $\mathcal{R}$. In each iteration, we guess an unexplored reaction sub-set R_{new} and add $(R_{new} \cap VR)$ to the candidate set R. If R is a feasible set of reactions, then R is returned. Otherwise, R_{new} is removed from the search space $\mathcal{U}$. Since in each iteration $R_{new} \neq \emptyset$ and $\mathcal{U}$ is finite, then Algorithm 2 is guaranteed to terminate. Note that this algorithm only provides an upper bound that is still double exponential in the number of literals, however studying the tight lower bound of the running time and the practical comparison of the algorithms is out of the scope of this paper.

7. Related work

Real world specifications often use more sophisticated data than Booleans, like integers, reals, or structured data. For finite domains, it is possible to use bit-blasting to obtain an equivalent Boolean specification. For non-finite domains, in recent years multiple approaches have been proposed to perform reactive synthesis with non Boolean inputs and outputs. For example, there have been decidability results for synthesis using register automata [33, 34, 35].

7.1. Richer Temporal Logics

There are many alternatives to LTL for specifying properties of reactive systems, including STL [36] and GTL [37]. We address here LTL with richer theories in the data, including a large class of first-order theories, which have long been used in verification [38].

Constraint LTL (CLTL) [39] extends LTL with the possibility of expressing constraints between variables at bounded distance (of time). A constraint system $\mathcal{D}$ consists of a concrete domain and an interpretation of relations on the domain. In CLTL over $\mathcal{D}$ (i.e., CLTL($\mathcal{D}$)), one can relate variables with relations defined in $\mathcal{D}$. CLTL can specify assignment-like statements by utilizing the equality relation. Like for all constraints allowing for a counting mechanism, CLTL with Presburger constraints, i.e., $CLTL(\mathbb{Z}, =, +)$, is undecidable; overall, the theories considered are a restricted form of $\mathcal{T}_{\mathbb{Z}}$ with only comparisons and the problem requires additional restrictions to overcome undecidability. In comparison, we do not allow predicates to compare variables at different timesteps, but we prove decidability for all theories with an $\exists^*\forall^*$ decidable fragment. Recently, LTL modulo theories is studied in [40] for finite traces and they allow temporal operators within predicates, again leading to undecidability.

7.2. TSL

Temporal Stream Logic (TSL) [41], was introduced as a new temporal logic for reactive synthesis that separates control from data. In the original TSL semantics, all functions and predicates are uninterpreted, but in [42] they describe an extension to TSL modulo theories. However, they consider only satisfiability and not synthesis; thus, in [43] they propose a synthesis algorithm for TSL modulo theories that can be applied to arbitrary decidable theories in which quantifier elimination is possible. Overall, TSL extends LTL with complex data that can be related accross time, making use of a new *update* operator; e.g., $[\![y \hookleftarrow f x]\!]$ indicates that the result of applying function f to variable x is assigned to y. In all these works, realizability is undecidable. Also, in [44] reactive synthesis and SyGuS [45] collaborate in the synthesis process, and generate executable code that implements both reactive and data-level properties, thus it suffers from two sources of undecidability: the undecidability of TSL synthesis [41] and the undecidability of syntax-guided synthesis [46]. In this paper, we cannot relate theory values accross time but we prove that realizability is decidable for all LTL.

When comparing TSL with LTL$_\mathcal{T}$, note that LTL$_\mathcal{T}$ includes all LTL, whereas TSL is undecidable already for the theory of equality and Presburger arithmetic. Indeed, TSL is undecidable for theory of uninterpreted functions, except for three fragments (see Thm. 7 in [42]): it is (1) semi-decidable for the reachability fragment of TSL (i.e., the fragment of TSL that only permits the next operator and the eventually operator as temporal operators), it is (2) decidable for formulae consisting of only logical operators, predicates, updates, next operators, and at most one top-level eventually operator, and it is (3) semi-decidable for formulae with one cell (i.e., controllable outputs).

Thus, note that Example 1.1 is decidable in TSL; however, a very few changes on it make it belong to a non-decidable TSL fragment: we can (1) add a $\mathcal{U}$ operator; or we can (2) add another $\Box$ operator into any literal (note that $\Box$ and *eventually* are dual); or (3) introduce a new controllable variable y'. This way, the modified running examples are not within the considered decidable nor semi-decidable fragments of TSL, whereas it is decidable in LTL$_\mathcal{T}$. Also, note that TSL allows (finite) uninterpreted predicates, whereas we need to have predicates well defined within the semantics of theories of specifications for which we perform Boolean abstraction.

7.3. Infinite (state) games

As for infinite-state games, a witness of recent researches on them is [47] where a causality-based algorithm for solving two-player reachability games represented by logical constraints is presented, based on the notion of *subgoals*.

Other approaches for solving infinite-state games include symbolic BDD-based state-space exploration [48], computing winning regions of both players using proof rules [49] and predicate abstraction [50]. There is also synthesis of infinite-state reactive implementations with random behavior [51].

7.4. Similar expressivity

Variable automata with arithmetic enable the specification of reactive systems with variables over an infinite domain of numeric values and whose operation involves arithmetic manipulation [52] of these values. [53] studies the synthesis problem for such specifications.

As for works closest to ours, [54] proposes numerical LTL synthesis for cyber-physical systems using an interplay between an LTL synthesizer and a non-linear real arithmetic checker. However, [54] (1) only considers specifications where the arithmetic predicates belong to the environment, (2) *overapproximates* the power of the system and (3) not precise realizability. Also, linear arithmetic games are studied in [55], which introduces algorithms for synthesizing winning strategies for non-reactive specifications. Also, [56] considers infinite theories (like us), but it does not guarantee success or termination, whereas our Boolean abstraction is complete. They only consider safety, while our approach considers all LTL. The follow-up [57] has still similar limitations. Similarly, [58] is incomplete, and requires a powerful solver for many quantifier alternations. As for [50], it only considers safety/liveness GR(1) specifications, and is limited to the theory of fixed-size vectors. We only require $\exists^*\forall^*$-satisfiability (for Boolean abstraction) and we consider multiple infinite theories. The usual main difference is that Boolean abstraction generates a (Boolean) LTL specification so that existing tools can be used with any of their internal techniques and algorithms (bounded synthesis, for example) and will automatically benefit from further optimizations. On the contrary, all approaches above adapt one specific technique and implement it in a monolithic way.

8. Conclusions

In this paper, we addressed decidability of reactive realizability of specifications in LTL$_{\mathcal{T}}$, which is the extension of LTL where atomic propositions can be literals from a first-order theory, including arithmetic theories. Our *Boolean abstraction* method (see Section 4) transforms specifications into purely Boolean specifications by (1) substituting theory literals by Boolean variables, and (2) computing an additional Boolean requirement that captures the dependencies between the new variables imposed by the literals. The resulting specification can be passed to existing Boolean off-the-shelf synthesis and realizability tools, and is realizable if and only if the original specification is realizable.

We present a proof that $\text{LTL}_{\mathcal{T}}$ realizability is decidable not only for temporal fragments such as safety, but for all specifications in theories with a decidable $\exists^*\forall^*$ fragment.

We showed an exhaustive Boolean abstraction method Algorithm 1, which relies on SMT solving and that can naturally be extended to be faster.

To the best of our knowledge, this is the first method that succeeds in non-Boolean LTL realizability or synthesis.

8.1. Discussion and future work

In this paper, we showed that, if variables do not relate over time, then $\text{LTL}_{\mathcal{T}}$ realizabilty is decidable (for $\exists^*\forall^*$-decidable theories); however, it is easy to see that if one can relate variables arbitrarily, we can build two counter machines (or TSL), showing undecidability. Thus, we are studying controlled (and useful) ways to pass information between temporal states; i.e., restricted forms of predicates that relate variables accross time. This allows us to enrich TSL so that benchmarks that previously were semi-decidable and hard to solve (e.g., experiment 5 in [43]) become decidable and easily solvable expressed in $\text{LTL}_{\mathcal{T}}$ and translated with Boolean abstraction.

As for other ongoing challenges, we focus on the followings:

- Enhancing the scalibility of the Boolean abstraction algorithm. Some of these optimizations rely on theory-based reasoning, whereas others exploit well-known decision procedures such as Cooper's quantifier elimination method [10].

- Expanding the expressive fragment of Boolean abstraction while maintaining decidability. We are studiying its usage with non-arithmetic theories such as the *Theory of Arrays* [59].
- Extending applicability of Boolean abstraction in different contexts. For instance, Boolean abstraction can be used to pre-check the realizability of (rich) specifications to be used in monitors of runtime verification [60].

Also, we are researching how to move from realizability modulo theories to reactive synthesis modulo theories, on realizable instances. The missing piece is the function that provides the system's values given environment's values that make the corresponding (Boolean) literals true according to the choice. This function is guaranteed to exist —e.g., if the literal is $(y > x)$, then one such function is $f(x) = x + 1$. An alternative is to use dynamical SMT queries to obtain values of the system [61].

References

[1] A. Pnueli, R. Rosner, On the synthesis of an asynchronous reactive module, in: Proc. of the 16th International Colloquium of Automata, Languages and Programming, (ICALP'89), Vol. 372 of LNCS, Springer, 1989, pp. 652–671. doi:10.1007/BFb0035790.

[2] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL'89), ACM Press, 1989, pp. 179–190. doi:10.1145/75277.75293.

[3] A. Pnueli, The temporal logic of programs, in: Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS'77), IEEE CS Press, 1977, pp. 46–67. doi:10.1109/SFCS.1977.32.

[4] S. Jacobs, N. Basset, R. Bloem, R. Brenguier, M. Colange, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, T. Michaud, G. A. Pérez, J. Raskin,

O. Sankur, L. Tentrup, The 4th reactive synthesis competition (SYNT-COMP 2017): Benchmarks, participants & results, in: Proc. of the 6th Workshop on Synthesis (SYNT@CAV 2017), Vol. 260 of EPTCS, 2017, pp. 116–143. `doi:10.4204/EPTCS.260.10`.

[5] B. Finkbeiner, S. Schewe, Bounded synthesis, Int. J. Softw. Tools Technol. Transf. 15 (5-6) (2013) 519–539. `doi:10.1007/s10009-012-0228-z`.

[6] A. R. Bradley, Z. Manna, The calculus of computation - decision procedures with applications to verification, Springer, 2007. `doi:10.1007/978-3-540-74113-8`.

[7] M. Davis, A computer program for Presburger's algorithm, in: Summaries of talks presented at the Summer Institute of Symbolic Logic in 1957 Cornell University, Vol. 2, 1957, pp. 215–223.

[8] R. Stansifer, Presburger's article on integer arithmetic: Remarks and translation, Tech. Rep. CORNELLCS:TR84-639, Cornell University Computer Science Department. (1984).

[9] D. C. Oppen, A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger Arithmetic, Journal of Computer and System Sciences 16 (3) (1978) 323–332. `doi:10.1016/0022-0000(78)90021-1`.

[10] D. W. Cooper, Theorem proving in arithmetic without multiplication, Machine Intelligence (1972) 91–100.

[11] J. Harrison, Introduction to Logic and Automated Theorem Proving, Cambridge, 2007.

[12] M. J. Fischer, M. O. Rabin, Super-exponential complexity of Presburger Arithmetic, in: Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer Vienna, Vienna, 1998, pp. 122–135.

[13] J. J. Robinson, Definability and decision problems in arithmetic, J. Symb. Log. 14 (1949) 98–114. `doi:10.2307/2266510`.

[14] J. Ferrante, C. Rackoff, A decision procedure for the first order theory of real addition with order, SIAM Journal of Computation (1975) 69–76doi:10.1137/0204006.

[15] M. Voigt, Decidable $\exists^*\forall^*$ first-order fragments of linear rational arithmetic with uninterpreted predicates, J. Autom. Reason. 65 (2021) 357–423. doi:10.1007/S10817-020-09567-8.

[16] A. Tarski, A decision method for elementary algebra and geometry, in: Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer Vienna, Vienna, 1998, pp. 24–84.

[17] A. Seidenberg, A new decision method for elementary algebra, Annals of Mathematics 60 (2) (1954) 365–374. doi:10.2307/2963604.

[18] G. E. Collins, Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: Proc. of the 2nd Conference in Automata Theory and Formal Languages (GI 1975), Vol. 33 of LNCS, Springer, 1975, pp. 134–183. doi:10.1007/3-540-07407-4_17.

[19] L. van den Dries, Alfred Tarski's elimination theory for real closed fields, The Journal of Symbolic Logic 53 (1) (1988) 7–19. doi:10.2307/2274424.

[20] D. Ierardi, Quantifier elimination in the theory of an algebraically-closed field, in: Proc. of the 21st Annual ACM Symposium on Theory of Computing, (STOC '89), 1989, p. 138–147. doi:10.1145/73007.73020.

[21] D. J. Ierardi, The complexity of quantifier elimination in the theory of an algebraically closed field, Thesis at Cornell University Press. 1989-08.

[22] A. Chistov, D. Grigoriev, Complexity of quantifier elimination in the theory of algebraically closed fields, in: Proc. of the Int'l Symp. on Mathematical Foundations of Computer Science (MFCS'84), Vol. 176 of LNCS, Springer, 1984, pp. 17–31. doi:10.1007/BFb0030287.

[23] J. Harrison, Complex quantifier elimination in HOL (2001) Division of Informatics, University of Edinburgh. Informatics Report Series EDI–INF–RR–0046, 159–174.

[24] S. Kripke, Semantical considerations on modal logic, Acta Philosophica Fennica 16 (1963) 83–94.

[25] E. M. Clarke, O. Grumberg, D. A. Peled, Model checking, 1st Edition, MIT Press, 2001. `doi:978-0-262-03270-4`.

[26] W. Thomas, Church's Problem and a Tour through Automata Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 635–655. `doi:10.1007/978-3-540-78127-1_35`.

[27] N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive(1) designs, in: Proc. of the 7th International Conference in Verification, Model Checking, and Abstract Interpretation (VMCAI'06), 2006, pp. 364–380. `doi:10.1007/11609773\_24`.

[28] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive(1) designs, Journal of Computer and System Sciences 78 (3) (2012) 911–938, in Commemoration of Amir Pnueli. `doi:10.1016/j.jcss.2011.08.007`.

[29] B. Finkbeiner, Synthesis of reactive systems, in: Dependable Software Systems Engineering, Vol. 45 of NATO Science for Peace and Security Series - D: Information and Communication Security, IOS Press, 2016, pp. 72–98. `doi:10.3233/978-1-61499-627-9-72`.

[30] R. Bloem, H. Chockler, M. Ebrahimi, O. Strichman, Vacuity in synthesis, Formal Methods Syst. Des. 57 (3) (2021) 473–495. `doi:10.1007/s10703-021-00381-5`.

[31] E. Grädel, W. Thomas, T. Wilke, Automata, logics, and infinite games: A guide to current research [outcome of a dagstuhl seminar, february 2001],

Vol. 2500 of Lecture Notes in Computer Science, Springer, 2002. doi:10.1007/3-540-36387-4.

[32] A. Rodriguez, C. Sánchez, Boolean abstractions for realizabilty modulo theories, in: Proc. of the 35th International Conference on Computer Aided Verification (CAV'23), Vol. 13966 of LNCS, Springer, Cham, 2023. doi:10.1007/978-3-031-37709-9_15.

[33] R. Ehlers, S. A. Seshia, H. Kress-Gazit, Synthesis with identifiers, in: Proc. of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'14), Vol. 8318 of LNCS, Springer, 2014, pp. 415–433. doi:10.1007/978-3-642-54013-4_23.

[34] L. Exibard, E. Filiot, A. Khalimov, Church synthesis on register automata over linearly ordered data domains, in: Proc. of the 38th International Symposium on Theoretical Aspects of Computer Science, (STACS 2021), Vol. 187 of LIPIcs, 2021, pp. 28:1–28:16. doi:10.4230/LIPIcs.STACS.2021.28.

[35] A. Khalimov, B. Maderbacher, R. Bloem, Bounded synthesis of register transducers, in: Proc. of the 16th International Symposium on Automated Technology for Verification and Analysis, (ATVA 2018), Vol. 11138 of LNCS, Springer, 2018, pp. 494–510. doi:10.1007/978-3-030-01090-4_29.

[36] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, S. A. Seshia, Reactive synthesis from signal temporal logic specifications, in: Proc. of the 18th International Conference on Hybrid Systems: Computation and Control, (HSCC '15), 2015, p. 239–248. doi:10.1145/2728606.2728628.

[37] D. Cyrluk, P. Narendran, Ground temporal logic: A logic for hardware verification, in: Proc. of the 6th International Conference on Computer Aided Verification, , (CAV '94), Vol. 818 of LNCS, Springer, 1994, pp. 247–259. doi:10.1007/3-540-58179-0_59.

[38] Z. Manna, A. Pnueli, Verification of concurrent programs: Temporal proof principles, in: Logics of Programs, Springer Berlin Heidelberg, 1982, pp. 200–252. `doi:10.1007/BFB0025785`.

[39] S. Demri, D. D'Souza, An automata-theoretic approach to constraint LTL, in: M. Agrawal, A. Seth (Eds.), Proc. of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science, (FST TCS 2002), India, December 12-14, 2002, Proceedings, Vol. 2556 of Lecture Notes in Computer Science, Springer, 2002, pp. 121–132. `doi:10.1007/3-540-36206-1\_12`.

[40] A. Gianola, N. Gigante, LTL modulo theories over finite traces: modeling, verification, open questions, in: L. Geatti, G. Sciavicco, A. Umbrico (Eds.), Short Paper Proceedings of the 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, Vol. 3311 of CEUR Workshop Proceedings, CEUR-WS.org, 2022, pp. 13–19.
URL `https://ceur-ws.org/Vol-3311/paper3.pdf`

[41] B. Finkbeiner, F. Klein, R. Piskac, M. Santolucito, Temporal stream logic: Synthesis beyond the bools, in: I. Dillig, S. Tasiran (Eds.), Proc. of the 31st International Conference in Computer Aided Verification - , (CAV 2019), Part I, Vol. 11561 of Lecture Notes in Computer Science, Springer, 2019, pp. 609–629. `doi:10.1007/978-3-030-25540-4\_35`.

[42] B. Finkbeiner, P. Heim, N. Passing, Temporal stream logic modulo theories, in: Proc. of the 25th International Conference in Foundations of Software Science and Computation Structures, (FOSSACS 2022), Vol. 13242 of LNCS, Springer, 2022, pp. 325–346. `doi:10.1007/978-3-030-99253-8\_17`.

[43] B. Maderbacher, R. Bloem, Reactive synthesis modulo theories using abstraction refinement, in: Proc. of the 22nd Formal Methods in Computer-Aided Design, (FMCAD 2022), IEEE, 2022, pp. 315–324. `doi:10.34727/2022/isbn.978-3-85448-053-2\_38`.

[44] W. Choi, B. Finkbeiner, R. Piskac, M. Santolucito, 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 2022), ACM, 2022, pp. 229–243. doi:10.1145/3519939.3523429.

[45] R. Alur, R. Bodík, E. Dallal, D. Fisman, P. Garg, G. Juniwal, H. Kress-Gazit, P. Madhusudan, M. M. K. Martin, M. Raghothaman, S. Saha, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, A. Udupa, Syntax-guided synthesis, in: Dependable Software Systems Engineering, Vol. 40 of NATO Science for Peace and Security Series, D: Information and Communication Security, IOS Press, 2015, pp. 1–25. doi:10.3233/978-1-61499-495-4-1. URL https://doi.org/10.3233/978-1-61499-495-4-1

[46] B. Caulfield, M. N. Rabe, S. A. Seshia, S. Tripakis, What's decidable about syntax-guided synthesis?, CoRR abs/1510.08393 (2015). arXiv:1510.08393.

[47] C. Baier, N. Coenen, B. Finkbeiner, F. Funke, S. Jantsch, J. Siber, Causality-based game solving, in: Proc. of the 33rd International Conference in Computer Aided Verification, (CAV 2021), Part I, Vol. 12759 of LNCS, Springer, 2021, pp. 894–917. doi:10.1007/978-3-030-81685-8_42.

[48] S. Edelkamp, P. Kissmann, Symbolic classification of general two-player games, in: 31st Annual German Conference on Advances in Artificial Intelligence(KI 2008), Vol. 5243 of LNCS, Springer, 2008, pp. 185–192. doi:10.1007/978-3-540-85845-4_23.

[49] T. A. Beyene, S. Chaudhuri, C. Popeea, A. Rybalchenko, A constraint-based approach to solving games on infinite graphs, in: Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL '14), ACM, 2014, pp. 221–234. doi:10.1145/2535838.2535860.

[50] A. Walker, L. Ryzhyk, Predicate abstraction for reactive synthesis, in: Proc. of the 14th Formal Methods in Computer-Aided Design, (FMCAD 2014), IEEE, 2014, pp. 219–226. `doi:10.1109/FMCAD.2014.6987617`.

[51] A. Katis, G. Fedyukovich, J. Chen, D. Greve, S. Rayadurgam, M. W. Whalen, Synthesis of infinite-state systems with random behavior, in: Proc. of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020), ACM, 2020. `doi:10.1145/3324884.3416586`.

[52] R. Faran, O. 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), Vol. 57 of EPiC Series in Computing, EasyChair, 2018, pp. 343–362. `doi:10.29007/wpg3`.

[53] R. Faran, O. Kupferman, On synthesis of specifications with arithmetic, in: Proc. of the 46th International Conference on Current Trends in Theory and Practice of Informatics, (SOFSEM 2020), Springer-Verlag, Berlin, Heidelberg, 2020, p. 161–173. `doi:10.1007/978-3-030-38919-2_14`.

[54] C. Cheng, E. A. Lee, Numerical LTL synthesis for cyber-physical systems, CoRR abs/1307.3722. `arXiv:1307.3722`.

[55] A. Farzan, Z. Kincaid, Strategy synthesis for linear arithmetic games, Proceedings of the ACM on Programming Languages 2 (2017) 1–30. `doi:10.1145/3158149`.

[56] A. Katis, G. Fedyukovich, A. Gacek, J. D. Backes, A. Gurfinkel, M. W. Whalen, Synthesis from assume-guarantee contracts using skolemized proofs of realizability, CoRR abs/1610.05867. `arXiv:1610.05867`.

[57] A. Katis, G. Fedyukovich, H. Guo, A. Gacek, J. Backes, A. Gurfinkel, M. W. Whalen, 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, Vol. 10806 of LNCS, Springer, 2018, pp. 176–193. doi:10.1007/978-3-319-89963-3_10.

[58] A. Gacek, A. Katis, M. W. Whalen, J. Backes, D. D. Cofer, Towards realizability checking of contracts using theories, in: Proc. of the 7th International Symposium NASA Formal Methods (NFM'15), Vol. 9058 of LNCS, Springer, 2015, pp. 173–187. doi:10.1007/978-3-319-17524-9_13.

[59] A. Bradley, Z. Manna, H. Sipma, What's decidable about arrays?, Vol. 3855, 2006, pp. 427–442. doi:10.1007/11609773_28.

[60] F. Gorostiaga, C. Sánchez, Hlola: a very functional tool for extensible stream runtime verification, in: Proc. of the 27th International Conference in Tools and Algorithms for the Construction and Analysis of Systems, (TACAS 2021), Part II, Vol. 12652 of LNCS, Springer, 2021, pp. 349–356. doi:10.1007/978-3-030-72013-1_18.

[61] A. Rodriguez, C. Sánchez, Adaptive Reactive Synthesis for LTL and LTLf Modulo Theories, in: Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024), AAAI Press, 2024, pp. 10679–10686. doi:10.1609/AAAI.V38I9.28939.