Shield Synthesis for LTL Modulo Theories

Andoni Rodríguez [1,2,*], Guy Amir [3,*], Davide Corsi[4], César Sánchez[1], Guy Katz [5]
[1] IMDEA Software Institute, Spain
[2] Universidad Politécnica de Madrid, Spain
[3] Cornell University, USA
[4] University of California, Irvine, USA
[5] The Hebrew University of Jerusalem, Israel
andoni.rodriguez@imdea.org, gda42@cornell.edu, dcorsi@uci.edu, cesar.sanchez@imdea.org, guykatz@cs.huji.ac.il

Abstract

In recent years, Machine Learning (ML) models have achieved remarkable success in various domains. However, these models also tend to demonstrate unsafe behaviors, precluding their deployment in safety-critical systems. To cope with this issue, ample research focuses on developing methods that guarantee the safe behaviour of a given ML model. A prominent example is *shielding* which incorporates an external component (a "shield") that blocks unwanted behavior. Despite significant progress, shielding suffers from a main setback: it is currently geared towards properties encoded solely in propositional logics (e.g., LTL) and is unsuitable for richer logics. This, in turn, limits the widespread applicability of shielding in many real-world systems. In this work, we address this gap, and extend shielding to LTL modulo theories, by building upon recent advances in reactive synthesis modulo theories. This allowed us to develop a novel approach for generating shields conforming to complex safety specifications in these more expressive, logics. We evaluated our shields and demonstrate their ability to handle rich data with temporal dynamics. To the best of our knowledge, this is the first approach for synthesizing shields for such expressivity.

Introduction

Recently, DNN-based agents trained using Deep Reinforcement Learning (DRL) have been shown to successfully control reactive systems of high complexity (e.g., (Marchesini and Farinelli 2020)) , such as robotic platforms. However, despite their success, DRL controllers still suffer from various safety issues; e.g., small perturbations to their inputs, resulting either from noise or from a malicious adversary, can cause even state-of-the-art agents to react unexpectedly (e.g., (Goodfellow, Shlens, and Szegedy 2014)) . This issue raises severe concerns regarding the deployment of DRL-based agents in safety-critical reactive systems.

In order to cope with these DNN reliability concerns, the formal methods community has recently put forth various tools and techniques that rigorously ensure the safe behaviour of DNNs (e.g., (Katz et al. 2017)) , and specifically, of DRL-controlled reactive systems (e.g., (Bassan et al. 2023)) . One of the main approaches that is gaining popularity, is *shielding* (Bloem et al. 2015; Alshiekh et al. 2018), i.e., the incorporation of an external component (a "shield") that *forces* an agent to behave safely according to a given specification. This specification φ is usually expressed as a propositional formula, in which the atomic propositions represent the inputs (I) and outputs (O) of the system, controlled by the DNN in question. Once φ is available, shielding seeks to guarantee that *all* behaviors of the given system D satisfy φ through the means of a shield S: whenever the system encounters an input I that triggers an erroneous output (i.e., $O : D(I)$ for which $\varphi(I, O)$ does not hold), S corrects O and replaces it with another action O', to ensure that φ(I, O') does hold. Thus, the combined system $D \cdot S$ never violates φ. Shields are appealing for multiple reasons: they do not require "white box" access to D, a single shield S can be used for multiple variants of D, it is usually computationally cheaper than static methods like DNN verification etc. Moreover, shields are intuitive for practitioners, since they are synthesized based on the required φ. However, despite significant progress, modern shielding methods still suffer from a main setback: they are only applicable to specifications in which the inputs/outputs are over Boolean atomic propositions. This allows users to encode only discrete specifications, typically in *Linear Temporal Logic* (LTL). Thus, as most real-world systems rely on rich data specifications, this precludes the use of shielding in various such domains, such as continuous input spaces.

In this work, we address this gap and present a novel approach for shield synthesis that makes use of LTL modulo theories (LTL$_\mathcal{T}$), where Boolean propositions are extended to literals from a (multi-sorted) first-order theory $\mathcal{T}$. Concretely, we leverage Boolean abstraction methods (Rodriguez and Sánchez 2023), which transform LTL$_\mathcal{T}$ specifications into equi-realizable pure (Boolean) LTL specifications. We combine Boolean abstraction with reactive LTL$_\mathcal{T}$ synthesis (Rodriguez and Sánchez 2024), extending the common LTL shielding theory into a LTL$_\mathcal{T}$ shielding theory. Using LTL$_\mathcal{T}$ shielding, we are able to construct shields for more expressive specifications. This, in turn, allows us to override unwanted actions in a (possibly infinite) domain of $\mathcal{T}$, and guarantee the safety of DNN-controlled systems in such complex scenarios. In summary, our contributions are: (1) developing two methods for shield synthesis over LTL$_\mathcal{T}$ and presenting their proof of correctness; (2) an

* Main co-authors.

analysis of the impact of the Boolean abstractions in the precision of shields; (3) a formalization of how to construct optimal shields using objective functions; and (4) an empirical evaluation that shows the applicability of our techniques.

Preliminaries

LTL and LTL$_\mathcal{T}$. We start from LTL (Pnueli 1977; Manna and Pnueli 1995), which has the following syntax:

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

where $a \in \mathsf{AP}$ is an *atomic proposition*, $\{\wedge, \neg\}$ are the common Boolean operators of *conjunction* and *negation*, respectively, and $\{\bigcirc, \mathcal{U}\}$ are the *next* and *until* temporal operators, respectively. Additional temporal operators include $\mathcal{R}$ (*release*), $\diamondsuit$(*finally*), and $\square$ (*always*), which can be derived from the syntax above. Given a set of atomic propositions $\overline{a}$ we use *val*$(\overline{a})$ for a set of possible valuations of variables in $\overline{a}$ (i.e. *val*$(\overline{a}) = 2^{\overline{a}}$), and we use $v_{\overline{a}}$ to range over *val*$(\overline{a})$. We use $\Sigma = \mathit{val}(\mathsf{AP})$. The semantics of LTL formulas associates traces $\sigma \in \Sigma^\omega$ with LTL fomulas (where $\sigma \models \top$ always holds, and $\vee$ and $\neg$ are standard):

$$\begin{array}{lll} \sigma \models a & \text{iff} & a \in \sigma(0) \\ \sigma \models \bigcirc\varphi & \text{iff} & \sigma^1 \models \varphi \\ \sigma \models \varphi_1\,\mathcal{U}\,\varphi_2 & \text{iff} & \text{for some } i \geq 0\ \ \sigma^i \models \varphi_2, \text{ and} \\ & & \quad \text{for all for all } 0 \leq j < i, \sigma^j \models \varphi_1 \end{array}$$

A safety formula φ is such that for every failing trace $\sigma \not\models \varphi$ there is a finite prefix u of σ, such that all σ' extending u also falsify φ (i.e. $\sigma' \not\models \varphi$).

The syntax of LTL modulo theory (LTL$_\mathcal{T}$) replaces atoms a by literals l from some theory $\mathcal{T}$. Even though we use multi-sorted theories, for clarity of explanation we assume that $\mathcal{T}$ has only one sort and use $\mathbb{D}$ for the domain, the set that populates the sort of its variables. For example, the domain of linear integer arithmetic $\mathcal{T}_\mathbb{Z}$ is $\mathbb{Z}$ and we denote this by $\mathbb{D}(\mathcal{T}_\mathbb{Z}) = \mathbb{Z}$. Given an LTL$_\mathcal{T}$ formula $\varphi(\overline{z})$ with variables $\overline{z}$ the semantics of LTL$_\mathcal{T}$ now associate traces σ (where each letter is a valuation of $\overline{z}$, i.e., a mapping from $\overline{z}$ into $\mathbb{D}$) with LTL$_\mathcal{T}$ formulae. The semantics of the Boolean and temporal operators are as in LTL, and for literals:

$$\sigma \models l \text{ iff the valuation } \sigma(0) \text{ of } \overline{z} \text{ makes } l \text{ true according to } \mathcal{T}$$

The Synthesis Problem. Reactive LTL synthesis (Thomas 2008; Piterman, Pnueli, and Sa'ar 2006) is the task of producing a system that satisfies a given LTL specification φ, where atomic propositions in φ are split into variables controlled by the environment ("input variables") and by the system ("output variables"), denoted by $\overline{e}$ and $\overline{s}$, respectively. Synthesis corresponds to a game where, in each turn, the environment player produces values for the input propositions, and the system player responds with values of the output propositions. A play is an infinite sequence of turns, i.e., an infinite interaction of the system with the environment. A strategy for the system is a tuple $C_\mathbb{B} : \langle Q, q_0, \delta, o\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the inital state, $\delta : Q \times \mathit{val}(\overline{e}) \to Q$ is the transition function and $o : Q \times \mathit{val}(\overline{e}) \to \mathit{val}(\overline{s})$ is the output function. $C_\mathbb{B}$ is said to be *winning* for the system if all the possible plays played according to the strategy satisfy the LTL formula $\varphi_\mathbb{B}$. In this paper we use "strategy" and "controller" interchangeably.

We also introduce the notion of *winning region* (WR), which encompasses all possible winning moves for the system in safety formulae. A winning region *WR* : $\langle Q, I, T\rangle$ is a tuple where Q is a finite set of states, $I \subseteq Q$ is a set of initial states and $T : Q \times \mathit{val}(\overline{e}) \to 2^{(Q \times \mathit{val}(\overline{s}))}$ is the transition relation, which provides for a given state q and input $v_{\overline{e}}$, all the possible pairs of legal successor and output $(q', v_{\overline{s}})$. For a safety specification, every winning strategy is "included" into the WR (i.e., there is an embedding map). LTL realizability is the decision problem of whether there is a winning strategy for the system (i.e., check if WR $\neq \emptyset$), while LTL synthesis is the computational problem of producing one.

However, in LTL$_\mathcal{T}$ synthesis (Rodríguez and Sánchez 2024; Rodriguez and Sánchez 2024), the specification is expressed in a richer logic where propositions are replaced by literals from some $\mathcal{T}$. In LTL$_\mathcal{T}$ the (first order) variables in specification $\varphi_\mathcal{T}$ are still split into those controlled by the environment ($\overline{x}$), and those controlled by the system ($\overline{y}$), where $\overline{x} \cap \overline{y} = \emptyset$. We use $\varphi_\mathcal{T}(\overline{x}, \overline{y})$ to empasize that $\overline{x} \cup \overline{y}$ are all the variables occurring in $\varphi_\mathcal{T}$. The alphabet is now $\Sigma_\mathcal{T} = \mathit{val}(\overline{x} \cup \overline{y})$ (note that now valuations map a variable x to $\mathbb{D}(\mathcal{T})$). We denote by $t[\overline{x} \leftarrow v_{\overline{x}}]$, the substitution in t of variables $\overline{x}$ by values $v_{\overline{x}}$ (similarly for $t[\overline{y} \leftarrow v_{\overline{y}}]$), and also $t[\overline{e} \leftarrow v_{\overline{e}}]$ and $t[\overline{s} \leftarrow v_{\overline{s}}]$ for Boolean variables (propositions). A trace π is an infinite sequence of valuations in $\mathbb{D}(\mathcal{T})$, which induces an infinite sequence of Boolean values of the literals occurring in $\varphi_\mathcal{T}$ and, in turn, an evaluation of $\varphi_\mathcal{T}$ using the semantics of the temporal operators. For example, given $\psi = \square(y < x)$ the trace π $\{(x : 2, y : 6), (x : 15, y : 27)\ldots\}$ induces $\{(\mathit{false}), (\mathit{true})\ldots\}$. We use π_x to denote the projection of π to the values of only x (resp. π_y for y). A strategy or controller for the system in $\mathcal{T}$ is now a tuple $C_\mathcal{T} : \langle Q, q_0, \delta, o\rangle$ where Q and $q_0 \in Q$ are as before, and $\delta : Q \times \mathit{val}(\overline{x}) \to Q$ and $o : Q \times \mathit{val}(\overline{x}) \to \mathit{val}(\overline{y})$.

Boolean Abstraction. Boolean abstraction (Rodriguez and Sánchez 2023) transforms an LTL$_\mathcal{T}$ specification $\varphi_\mathcal{T}$ into an LTL specification $\varphi_\mathbb{B}$ in the same temporal fragment (e.g., safety to safety) that preserves realizability, i.e., $\varphi_\mathcal{T}$ and $\varphi_\mathbb{B}$ are *equi-realizable*. Then, $\varphi_\mathbb{B}$ can be fed into an off-the-shelf synthesis engine, which generates a controller or a WR for realizable instances. Boolean abstraction transforms $\varphi_\mathcal{T}$, which contains literals l_i, into $\varphi_\mathbb{B} = \varphi_\mathcal{T}[l_i \leftarrow s_i] \wedge \varphi^{\mathit{extra}}$, where $\overline{s} = \{s_i | \text{for each } l_i\}$ is a set of fresh atomic propositions controlled by the system—such that s_i replaces l_i—and where φ^{extra} is an additional sub-formula that captures the dependencies between the $\overline{s}$ variables. The formula φ^{extra} also includes additional environment variables $\overline{e}$ that encode that the environment can leave the system with the power to choose certain valuations of the variables $\overline{s}$.

We often represent a valuation $v_{\overline{s}}$ of the Boolean variables $\overline{s}$ (which map each variable in $\overline{s}$ to *true* or *false*) as a *choice* c (an element of $2^{\overline{s}}$), where $s_i \in c$ means that $v_{\overline{s}}(s_i) = \mathit{true}$. The characteristic formula $f_c(\overline{x}, \overline{y})$ of a choice c is $f_c = \bigwedge_{s_i \in c} l_i \wedge \bigwedge_{s_i \notin c} \neg l_i$. We use $\mathcal{C}$ for the set of choices, i.e., sets of sets of $\overline{s}$. A *reaction* $r \subset \mathcal{C}$ is a set of choices, which characterizes the possible responses of the system as the result to a move of the environment. The *characteristic formula* $f_r(\overline{x})$ of a reaction r is: $(\bigwedge_{c\in r} \exists\overline{y}.f_c) \wedge (\bigwedge_{c\notin r} \forall\overline{y}\neg f_c)$. We say that r is a valid reaction whenever $\exists\overline{x}.f_r(\overline{x})$ is valid. Intuitively, f_r states that for some $v_{\overline{x}}$ by the environment, the system can respond with $v_{\overline{y}}$ making the literals in some choice $c \in r$ but cannot respond with $v_{\overline{y}}$ making the literals in choices $c \notin r$. The set *VR* of valid reactions partitions precisely the moves of the environment in terms of the reaction power left to the system. For each valid reaction r there is a fresh environment variable $e \in \overline{e}$ used in φ^{extra} to capture the move of the environment that chooses reaction r. The formula $f_r(\overline{x})[\overline{x} \leftarrow v_{\overline{x}}]$ is true if a given valuation $v_{\overline{x}}$ of $\overline{x}$ is one move of the environment characterized by r. The formula φ^{extra} restricts the environment such that exactly one of the variables in $\overline{e}$ is true, which forces that the environment chooses precisely one valid reaction r when the variable e that corresponds to r is true.

Example 1 (Running example and abstraction)**.** *In this paper, we address shields for general safety* LTL$_\mathcal{T}$*, but for the sake of simplicity in the presentation, we consider a simpler example. Let* $\varphi_\mathcal{T} = \Box(R_0 \wedge R_1)$ *where:*

$$R_0 : (x < 10) \rightarrow \bigcirc(y > 9) \qquad R_1 : (x \geq 10) \rightarrow (y \leq x),$$

where $\overline{x} = \{x\}$ *is controlled by the environment and* $\overline{y} = \{y\}$ *by the system. In integer theory* $\mathcal{T}_\mathbb{Z}$ *this specification is realizable (consider the strategy to always play* $y : 10$*) and the Boolean abstraction first introduces* s_0 *to abstract* $(x < 10)$*,* s_1 *to abstract* $(y > 9)$ *and* s_2 *to abstract* $(y \leq x)$*. Then* $\varphi_\mathbb{B} = \varphi'' \wedge \Box(\varphi^{legal} \rightarrow \varphi^{extra})$ *where* $\varphi'' = (s_0 \rightarrow \bigcirc s_1) \wedge (\neg s_0 \rightarrow s_2)$ *is a direct abstraction of* $\varphi_\mathcal{T}$*. Finally,* φ^{extra} *captures the dependencies between the abstracted variables:*

$$\varphi^{extra} : \left(\begin{array}{ccccc} & (e_0 & \rightarrow & (f_{c_1} \vee f_{c_2}) & \\ \wedge & (e_{0+} & \rightarrow & (f_{c_1} \vee f_{c_2} \vee f_{c_3}) & \\ \wedge & (\ e_1 & \rightarrow & (f_{c_4} \vee f_{c_5} \vee f_{c_6}) & \end{array} \right),$$

where $f_{c_1} = (s_0 \wedge s_1 \wedge \neg s_2)$*,* $f_{c_2} = (s_0 \wedge \neg s_1 \wedge s_2)$*,* $f_{c_3} = (s_0 \wedge \neg s_1 \wedge \neg s_2)$*,* $f_{c_4} = (\neg s_0 \wedge s_1 \wedge s_2)$*,* $f_{c_5} = (s_0 \wedge s_1 \wedge \neg s_2)$ *and* $f_{c_6} = (\neg s_0 \wedge \neg s_1 \wedge s_2)$ *and where* $c_0 = \{s_0, s_1, s_2\}$*,* $c_1 = \{s_0, s_1\}$*,* $c_2 = \{s_0, s_2\}$*,* $c_3 = \{s_0\}$*,* $c_4 = \{s_1, s_2\}$*,* $c_5 = \{s_1\}$*,* $c_6 = \{s_1\}$ *and* $c_7 = \emptyset$*. Also,* $\overline{e} = \{e_0, e_{0+}, e_1\}$ *belong to the environment and represent* $(x < 10)$*,* $(x < 9)$ *and* $(x \geq 10)$*, respectively.* $\varphi^{legal} : (e_0 \wedge \neg e_1 \wedge \neg e_2) \vee (\neg e_0 \wedge e_1 \wedge \neg e_2) \vee (\neg e_0 \wedge \neg e_1 \wedge e_2)$ *encodes that* $\overline{e}$ *characterizes a partition of the (infinite) input valuations of the environment and that only one of the* $\overline{e}$ *is true in every move; e.g., the valuation* $v_{\overline{e}} : \langle e_0 : \textit{false}, e_0^+ : \textit{true}, e_1 : \textit{false}\rangle$ *of* $\overline{e}$ *corresponds to the choice of the environment where only* e_0^+ *is true (and we use* $v_{\overline{e}} : e_0^+$ *for a shorter notation). Sub-formulae such as* $(\neg s_0 \wedge s_1 \wedge s_2)$ *represent the choices of the system (in this case,* $c = \{s_1, s_2\}$*), that is, given a decision of the environment (a valuation of* $\overline{e}$ *that makes exactly one variable* $e \in \overline{e}$ *true), the system can* react *with one of the choices c in the disjunction implied by e. Note that* $f_c = \neg(x < 2) \wedge (y > 1) \wedge (y \leq x)$*. Also, note that c can be represented as* $v_{\overline{s}} : \langle s_0 : \textit{false}, s_1 : \textit{true}, s_2 : \textit{true}\rangle$*.*

LTL$_\mathcal{T}$ Shield Computation

Problem overview. In shielding , at every step, the external design D —e.g., a DRL sub-system— produces an output $v_{\overline{s}}$ from a given input $v_{\overline{e}}$ provided by the environment. The pair $(v_{\overline{e}}, v_{\overline{s}})$ is passed to the shield S which decides, at every step, whether $v_{\overline{s}}$ proposed by D is safe with respect to some specification φ. The combined system $D \cdot S$ is guaranteed to satisfy φ. Using LTL$_\mathcal{T}$ we can define richer properties than in propositional LTL (which has been previously used in shielding). For instance, consider a classic $D \cdot S$ context in which the D is a robotic navigation platform and $\varphi : \Box(\texttt{LEFT} \rightarrow \bigcirc\neg\texttt{LEFT})$. Then, if D chooses to turn `LEFT` after a `LEFT` action, the shield will consider this second action to be dangerous, and will override it with e.g., `RIGHT`. If LTL$_\mathcal{T}$ is used instead of LTL, specifications can be more sophisticated including, for example, numeric data.

Shielding modulo theories is the problem of building a shield $S_\mathcal{T}$ *from a specification* $\varphi_\mathcal{T}$ *in which at least one of the input variables* $\overline{x}$ *or one of the output variables* $\overline{y}$ *are not Boolean.*

A shield $S_\mathcal{T}$ conforming to $\varphi_\mathcal{T}$ will evaluate if a pair $(\overline{x} : v_{\overline{x}}, \overline{y} : v_{\overline{y}})$ violates $\varphi_\mathcal{T}$ and propose an overriding output $\overline{y} : v'_{\overline{y}}$ if so. This way, $D \cdot S_\mathcal{T}$ never violates $\varphi_\mathcal{T}$. Note that it is possible that $\varphi_\mathcal{T}$ is unrealizable, which means that some environment plays will inevitably lead to violations, and hence $S_\mathcal{T}$ cannot be constructed, because WR $= \emptyset$.

Example 2 (Running example as shield)**.** *Recall* $\varphi_\mathcal{T}$ *from Ex. 1. Also, consider D receives an input trace* $\pi_x = \langle 15, 15, 7, 5, 10\rangle$*, and produces the output trace* $\pi_y = \langle 6, 5, 13, 16, 11\rangle$*. We can see D violates* $\varphi_\mathcal{T}$ *in the fifth step, since* $(x < 10)$ *holds in fourth step (so* v_y *has to be such that* $(y : v_y > 9)$ *in the fourth step and* $(v_y :\leq 10)$ *must hold in the fifth step) but* $(y : 11 \leq 10)$ *is not true. Instead, a* $S_\mathcal{T}$ *conforming to* $\varphi_\mathcal{T}$ *would notice that* $(x : 10, y : 11)$ *violates* $\varphi_\mathcal{T}$ *in this fifth step and would override* v_y *with* v'_y *to produce* $(y' : 10)$*, which is the only possible valuation of* y' *that does not violate* $\varphi_\mathcal{T}$*. Note that* $S_\mathcal{T}$ *did not intervene in the remaining steps. Thus,* $D \cdot S_\mathcal{T}$ *satisfies* $\varphi_\mathcal{T}$ *in the example.*

We propose two different architectures for $S_\mathcal{T}$: one following a deterministic strategy and another one that is non-deterministic. Both start from $\varphi_\mathcal{T}$ and use Boolean abstraction as the core for the temporal information, but they vary on the way to detect erroneous outputs from D and how to provide corrections.

Shields as Controllers. The first method leverages LTL$_\mathcal{T}$ controller synthesis (see Fig. 1(a)) together with a component to detect errors in D. The process is as follows, starting from a specification $\varphi_\mathcal{T}$:

1. Boolean abstraction (Rodriguez and Sánchez 2023) transforms $\varphi_\mathcal{T}$ into an equi-realizable LTL $\varphi_\mathbb{B}$.
2. A Boolean controller $C_\mathbb{B}$ is synthesized from $\varphi_\mathbb{B}$ (using e.g., (Meyer, Sickert, and Luttenberger 2018)). $C_\mathbb{B}$ receives Boolean inputs $v_{\overline{e}}$ and provides Boolean outputs $v_{\overline{s}}$.
3. We synthesize a richer controller $C_\mathcal{T}$ that receives $v_{\overline{x}}$ and produces outputs $v'_{\overline{y}}$ in $\mathbb{D}(\mathcal{T})$. Note the apostrophe in $v'_{\overline{y}}$, since $v_{\overline{y}}$ is the output of D.

4. $S_{\mathcal{T}}$ receives the output $v_{\overline{y}}$ provided by D and checks if the pair $(v_{\overline{x}}, v_{\overline{y}})$ violates $\varphi_{\mathcal{T}}$: if there is a violation, it overrides $v_{\overline{y}}$ with $v'_{\overline{y}}$; otherwise, it permits $v_{\overline{y}}$.

We now elaborate on the last steps. To construct $C_{\mathcal{T}}$ from $C_{\mathbb{B}}$ we use (Rodríguez and Sánchez 2024) , where $C_{\mathcal{T}}$ is composed of three sub-components:

- A **partitioner** function, which computes Boolean inputs $v_{\overline{e}}$ from the richer inputs $v_{\overline{x}}$ via partitioning, checking whether $f_r(v_{\overline{x}})$ is true. Recall that every $v_{\overline{x}}$ is guaranteed to belong to exactly one reaction r.
- A **Boolean controller** $C_{\mathbb{B}}$, which receives $v_{\overline{e}}$ and provides Boolean outputs $v_{\overline{s}}$. Also recall that each Boolean variable in $v_{\overline{s}}$ corresponds exactly to a literal in $\varphi_{\mathcal{T}}$ and that f_c is the characteristic formula of c.
- A **provider**, which computes an output $v'_{\overline{y}}$ from f_c and r such that $v'_{\overline{y}}$ is a model of the formula $\exists \overline{y}'.\big(f_r(\overline{x}) \rightarrow f_c(\overline{y}', \overline{x})\big)[\overline{x} \leftarrow v_{\overline{x}}]$.

The composition of the **partitioner**, $C_{\mathbb{B}}$ and the **provider** implements $C_{\mathcal{T}}$ (see (Rodríguez and Sánchez 2024; Rodriguez and Sánchez 2024)). For each of these components g, let us denote with $g(t_0) = t_1$ the application of g with input t_0. Then, the following holds:

Lemma 1. *Let $\varphi_{\mathcal{T}}$ be an* $\text{LTL}_{\mathcal{T}}$ *specification, $\varphi_{\mathbb{B}}$ its equi-realizable Boolean abstraction and $C_{\mathbb{B}}$ a controller for $\varphi_{\mathbb{B}}$. Let q be a state of $C_{\mathbb{B}}$ after processing inputs $v^0_{\overline{x}} \dots v^n_{\overline{x}}$, and let $v_{\overline{x}}$ be the next input from the environment. Let* **partitioner**$(v_{\overline{x}}) = v_{\overline{e}}$ *be the partition corresponding to $v_{\overline{x}}$, $o(q, v_{\overline{e}}) = v_{\overline{s}}$ be the output of $C_{\mathbb{B}}$ and c be the choice associated to $v_{\overline{s}}$. Then, if r is a valid reaction, the formula $\exists \overline{y}. f_r(v_{\overline{x}}) \rightarrow f_c(\overline{y}, v_{\overline{x}})$ is satisfiable in $\mathcal{T}$, where $c \in r$.*

In other words, $C_{\mathcal{T}}$ never has to face with moves from $C_{\mathbb{B}}$ that cannot be mimicked by **provider** with appropriate values for $v_{\overline{y}}$. Thus, a shield $S_{\mathcal{T}}$ based on $C_{\mathcal{T}}$ (see Fig. 1(a)) first detects whether a valuation pair $(\overline{x} : v_{\overline{x}}, \overline{y} : v_{\overline{y}})$ suggested by D violates $\varphi_{\mathcal{T}}$. To do so, at each time-step, $C_{\mathcal{T}}$ obtains $v_{\overline{s}}$ by $C_{\mathbb{B}}$, with associated c, and it checks whether $f_r(v_{\overline{x}}) \rightarrow f_c(v_{\overline{x}}, v_{\overline{y}})$ is valid, that is, whether the output proposed is equivalent to the move of $C_{\mathbb{B}}$ (in the sense that every literal l_i from $(v_{\overline{x}}, v_{\overline{y}})$ has the same valuation $v_{\overline{s}}$ given by $C_{\mathbb{B}}$). If the formula is valid, then $v_{\overline{y}}$ is maintained. Otherwise, the **provider** is invoked to compute a $v'_{\overline{y}}$ that matches the move of $C_{\mathbb{B}}$. In both cases, $S_{\mathcal{T}}$ guarantees that $v_{\overline{s}}$ remains unaltered in $C_{\mathbb{B}}$ and therefore $\varphi_{\mathcal{T}}$ is guaranteed. For instance, in the hypothetical simplistic case where $f_r = \mathit{true}$, and consider that $C_{\mathbb{B}}$ outputs a $v_{\overline{s}}$ with associated c such that $f_c(x, y) = (y > 2) \wedge (x < y)$ in $\mathcal{T}_{\mathbb{Z}}$. Then if the input is $x : 5$, a candidate output $y : 4$ of D would result in $f_r(5) \rightarrow f_c(5, 4) = (4 > 2) \wedge (5 < 4)$, which does not hold. Thus, $y : 4$ must be overridden, so $S_{\mathcal{T}}$ will return a model of $\exists y'. f_r(x) \rightarrow (y' > 2) \wedge (x < y')[x \leftarrow 5]$. One such possibility is $y' : 6$.

Lemma 2. *Let $\varphi_{\mathbb{B}}$ be an equi-realizable abstraction of $\varphi_{\mathcal{T}}$, $C_{\mathbb{B}}$ a controller for $\varphi_{\mathbb{B}}$ and $C_{\mathcal{T}}$ a corresponding controller for $\varphi_{\mathcal{T}}$. Also, consider input $v_{\overline{x}}$ and output $v_{\overline{y}}$ by D and $v'_{\overline{y}}$ by* ***provider*** *of $C_{\mathcal{T}}$. If both $f_r(v_{\overline{x}}) \rightarrow f_c(v_{\overline{x}}, v_{\overline{y}})$ and $f_r(v_{\overline{x}}) \rightarrow f_c(v_{\overline{x}}, v'_{\overline{y}})$ hold, then $v_{\overline{y}}$ and $v'_{\overline{y}}$ correspond to the same move of $C_{\mathbb{B}}$.*

In other words, if $v_{\overline{y}}$ and $v'_{\overline{y}}$ make the same literals true, a shield $S_{\mathcal{T}}$ based on $C_{\mathcal{T}}$ will not override $v_{\overline{y}}$; whereas, otherwise, it will output $v'_{\overline{y}}$.

Theorem 1 (Correctness of $D \cdot S_{\mathcal{T}}$ based on $C_{\mathcal{T}}$)**.** *Let D be an external controller and $S_{\mathcal{T}}$ be a shield constructed from a synthetized $C_{\mathcal{T}}$ from a specification $\varphi_{\mathcal{T}}$. Then, $D \cdot S_{\mathcal{T}}$ is also a controller for $\varphi_{\mathcal{T}}$.*

In other words, given $\varphi_{\mathcal{T}}$, if $\mathcal{T}$ is decidable in the $\exists^*\forall^*$-fragment (a condition for the Boolean abstraction (Rodriguez and Sánchez 2023)), then, leveraging $C_{\mathcal{T}}$, a shield $S_{\mathcal{T}}$ conforming to $\varphi_{\mathcal{T}}$ can be computed for every external D whose input-output domain is $\mathcal{T}$.

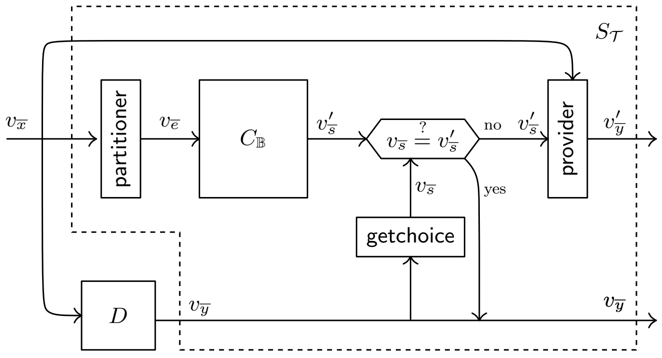
More Flexible Shields from Winning Regions. $S_{\mathcal{T}}$ based on $C_{\mathcal{T}}$ are sometimes too intrusive in the sense that they can cause unnecessary corrections, because some $(v_{\overline{x}}, v_{\overline{y}})$ may be labelled as a violation when they are not, only because it satisfies a different collection of literals than the ones chosen by the internal controller $C_{\mathbb{B}}$. Given input $v_{\overline{x}}$, the output $v_{\overline{s}}$ provided by $C_{\mathbb{B}}$ and the output $v_{\overline{y}}$ suggest by D, $(v_{\overline{x}}, v_{\overline{y}})$ is considered to be a *dangerous output* (i.e., a potential violation of $\varphi_{\mathcal{T}}$) whenever a $f_r(v_{\overline{x}})$ holds but the $f_c(v_{\overline{x}}, v_{\overline{y}})$ of the c associated to $C_{\mathbb{B}}$(**partitioner**$(v_{\overline{x}})) = v_{\overline{s}}$ does not hold. Thus, we study now permissive shields $S_{\mathcal{T}}$ with more precise characterizations of unsafe actions.

The *winning region* (*WR*) is the most general characterization of the winning states and moves for a safety LTL specification. This second construction can be applied to any subset of *WR* as long as every state has at least one successor for every input (for example, using the combination of a collection of controllers). The process of computing $S_{\mathcal{T}}$ based on a given winning region $W : \langle Q, I, T\rangle$ is as follows (see Fig. 1(b)) and maintains at each instant a set of current states $Q_{\mathit{now}} \subseteq Q$ of W, starting from the initial set of states I.

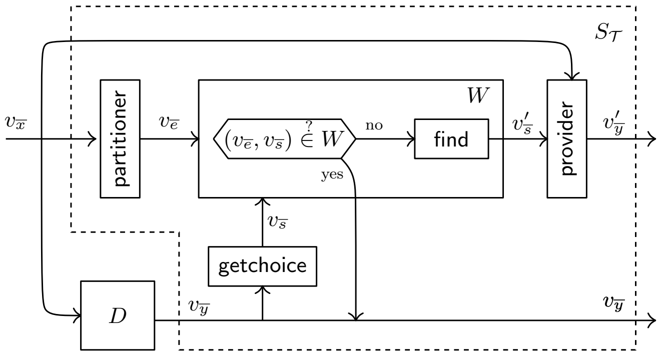
1. Compute $\varphi_{\mathbb{B}}$ from $\varphi_{\mathcal{T}}$ using abstraction, as before.
2. We compute the winning region W from $\varphi_{\mathbb{B}}$ (e.g., using (Brenguier et al. 2014)).
3. The **partitioner** computes $v_{\overline{e}}$ from the input values $v_{\overline{x}}$; and, given $v_{\overline{y}}$ provided by D, we produce the unique $v_{\overline{s}}$ from $(v_{\overline{x}}, v_{\overline{y}})$ or equivalently $c = \{s_i | l_i(v_{\overline{x}}, v_{\overline{y}}) \text{ holds}\}$. This is computed by component $\mathsf{getchoice}$ in Fig. 1(b).
4. From some of the current states $q \in Q_{\mathit{now}}$ maintained, we decide whether $\delta(q, v_{\overline{e}})$ has some successor $(q', v_{\overline{s}})$ or not. If it does, output $v_{\overline{y}}$. If it does not, choose a successor output $v'_{\overline{s}}$ such that there is $(q', v'_{\overline{s}}) \in \delta(q, v_{\overline{e}})$ for some $q \in Q_{\mathit{now}}$, and use a **provider** to produce an output $v'_{\overline{y}}$ (this is implemented by find in Fig. 1(b)). In either case, use appropriate valuation $\overline{v}$ of $\overline{s}$ (i.e., $v_{\overline{s}}$ or $v'_{\overline{s}}$) to update Q_{now} to $\{q' | (q', \overline{v}) \in \delta(q, v_{\overline{e}}) \text{ for some } q \in Q_{\mathit{now}}\}$.

Note that if the environment generates an input $v_{\overline{x}}$ (with corresponding $v_{\overline{e}}$) and D responds with an output $v_{\overline{y}}$ such that $\mathsf{getchoice}(v_{\overline{y}}) = v_{\overline{s}}$, if W permits $(v_{\overline{e}}, v_{\overline{s}})$ then the suggested $v_{\overline{y}}$ remains. Otherwise, we use a component find

See the extended version of this paper (Rodriguez et al. 2024) for a for additional details.



(a) Using $C_{\mathcal{T}}$ as $S_{\mathcal{T}}$.



(b) Using W as $S_{\mathcal{T}}$.

Figure 1: Architectures for shields modulo theory.

which computes an alternative $v'_{\overline{s}}$ such that $(v_{\overline{e}}, v'_{\overline{s}})$ is in W and a proper value $v_{\overline{y}}$ that corresponds to $v_{\overline{s}}$ is output. Also, note that find can always identify a value $v'_{\overline{s}}$ from any $q \in Q_{now}$ such that there is a successor $(q', v'_{\overline{s}}) \in \delta(q, v_{\overline{e}})$ by the definition of *WR*. Given two $v_{\overline{s}}$ and $v'_{\overline{s}}$ such that $(v_{\overline{e}}, v_{\overline{s}}) \in W$ and $(v_{\overline{e}}, v'_{\overline{s}}) \in W$, the criteria (e.g., *choice-logic* (Bernreiter, Maly, and Woltran 2021)) to choose one is not relevant for this paper and we leave it for future work. Finally, given $v_{\overline{x}}$, the W cannot distinguish two values $v_{\overline{y}}$ and $v'_{\overline{y}}$ if $\mathsf{getchoice}(v_{\overline{x}}, v_{\overline{y}}) = \mathsf{getchoice}(v_{\overline{x}}, v'_{\overline{y}})$.

Theorem 2 (Correctness of $D \cdot S_{\mathcal{T}}$ based on WR)**.** *Let $\varphi_{\mathcal{T}}$ be a safety specification and $\varphi_{\mathbb{B}}$ an abstraction of $\varphi_{\mathcal{T}}$. Let $S_{\mathcal{T}}$ be synthesized using the winning region W of $\varphi_{\mathbb{B}}$. Then, $D \cdot S_{\mathcal{T}}$ satisfies $\varphi_{\mathcal{T}}$, for any external controller D.*

Thm. 2 holds because the valuations of the literals of any trace π of $D \cdot S_{\mathcal{T}}$ is guaranteed to be a path in W and therefore will satisfy $\varphi_{\mathbb{B}}$. Hence, $\pi \models \varphi_{\mathcal{T}}$. The main practical difference between using a controller $C_{\mathbb{B}}$ and a winning region W, is expressed by the following lemma:

Lemma 3. *Let C_1 be controller obtained using a controller $C_{\mathbb{B}}$ for $\varphi_{\mathbb{B}}$ and C_2 obtained from the WR of $\varphi_{\mathbb{B}}$. Let D be an external controller. Let π be a trace obtained by $D \cdot C_1$ for some input $\pi_{\overline{x}}$. Then, π is a trace of $(D \cdot C_1) \cdot C_2$ for $\pi_{\overline{x}}$.*

Lemma 3 essentially states that a WR controller is less intrusive than one based on a specific Boolean controller $C_{\mathbb{B}}$. However, in practice, a WR is harder to interpret and slower to compute than $C_{\mathbb{B}}$. Note that we presented $S_{\mathcal{T}}$ based on $C_{\mathcal{T}}$ and $S_{\mathcal{T}}$ based on WR, but for classic LTL shielding only the latter case makes sense. This is because in classic LTL using shield S based on $C_{\mathbb{B}}$ is equivalent to forcing D to coincide with S on all time-steps, so S could be directly used ignoring D. However, in LTL$_{\mathcal{T}}$ there are potentially many different $v_{\overline{y}}$ by D that correspond to a given Boolean $v'_{\overline{s}}$ by $C_{\mathbb{B}}$, so even when this architecture forces to mimick $C_{\mathbb{B}}$ in the Boolean domain, it is reasonable to use an external source D of rich value output candidates. Thus, not only we present an extension of shielding to LTL$_{\mathcal{T}}$ (with the WR approach), but also this is the first work that is capable to synthetise shields leveraging all the power of classic synthesis (with the $C_{\mathcal{T}}$ approach): e.g., we can use bounded synthesis (Schewe and Finkbeiner 2007) instead of computing a WR.

Permissive and Intrusive Shields

Minimal and Feasible Reactions. Computing a Boolean Abstraction boils down to calculating the set *VR* of valid reactions. In order to speed up the calculation, optimizations are sometimes used (see (Rodriguez and Sánchez 2023)). For instance, Ex. 1 shows a formula $\varphi_{\mathcal{T}}$ that can be Booleanized into a $\varphi_{\mathbb{B}}$ that contains three environment variables (e_0, e_0^+ and e_1) that correspond to the three valid reactions. A closer inspection reveals that the valid reaction associated to e_0^+ (which represents $(x < 1)$) is "subsumed" by the reaction associated to e_0 (which represents $(x < 2)$), which is witnessed by e_0 leaving to the system a strictly smaller set of choices than e_0^+. In the game theoretic sense, an environment that plays e_0^+ is leaving the system strictly more options, so a resulting Booleanizaed formula $\varphi'_{\mathbb{B}}$ that simply ignores e_0^+ only limits the environment from playing sub-optimal moves, and $\varphi'_{\mathbb{B}}$ is also equi-realizable to $\varphi_{\mathcal{T}}$. From the equi-realizability point of view, one can neglect valid reactions that provide strictly more choices than other valid reactions. However, even though using $\varphi'_{\mathbb{B}}$ does not compromise the correctness of the resulting $S_{\mathcal{T}}$ (using either method described before), the resulting $S_{\mathcal{T}}$ will again be more intrusive than if one uses $\varphi_{\mathbb{B}}$, as some precision is lost (in particular for inputs $(x < 1)$, which are treated by $\varphi'_{\mathbb{B}}$ as $(x < 2)$). We now rigorously formalize this intuition.

A reaction r is *above* another reaction r' whenever $r' \subseteq r$. Note that two reactions r and r' can be not comparable, since neither contains the same or a strictly larger set of playable choices than the other. We now define two sets of reactions that are smaller than *VR* but still guarantee that a $\varphi_{\mathbb{B}}$ obtained from $\varphi_{\mathcal{T}}$ by Boolean abstraction is equi-realizable. Recall that the set of valid reactions is *VR* $= \{r | \exists \overline{x}. f_r(\overline{x})$ is valid$\}$.

Definition 1 (*MVR* and Feasible)**.** *The set of* minimal valid reactions *is MVR* $= \{r \in VR \mid$ *there is no* $r' \in$ *VR such that* $r' \subsetneq r\}$. *A set of reactions R is a feasible 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. In order to see whether a set of reactions R is *below VR* or is indeed *VR*, we need to check two properties.

Definition 2 (Legitimacy and Strict Covering)**.** *Let R be a set of reactions:*

- *R is* legitimate *iff for all $r \in R$, $\exists \overline{x}.\ f_r(\overline{x})$ is valid.*
- *R is a* strict covering *iff $\forall \overline{x}. \bigvee_{r \in R} f_r(\overline{x})$ is valid.*

If R is legitimate, then $R \subseteq$ *VR*. If additionally R is a strict covering, then $R =$ *VR*. Strict covering implies that all possible moves of the environment are 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. A *non-strict* covering does not necessarily consider all the possible moves of the environment in the game, but it still considers at least all optimal environment moves. A non-strict covering can be evaluated by checking that with regard to R, for all $\overline{x}$, the disjunction of the **playable** choices (see $\bigwedge_{c \in r} \exists \overline{y}. f_c$ of f_r in Preliminaries) of its reactions holds.

Definition 3 (Covering)**.** *The playable formula for a reaction r is defined as $f_r^P(\overline{x}) \overset{def}{=} \bigwedge_{c \in r} \exists \overline{y}. f_c(\overline{x}, \overline{y})$. 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.*

Note that a playable formula removes from the characteristic formula f_r the sub-formula $(\bigwedge_{c \notin r} \forall \overline{y} \neg f_c)$ that captures that the choices not in r cannot be achieved by any $v_{\overline{y}}$. We can easily check whether a set of reactions R is feasible, as follows. First, R must be legitimate. Second, if $\varphi_{cov}(R)$ is valid, then R contains a subset of valid reactions that makes R covering in the sense that it considers all the "clever" moves for the environment (considering that a move that leaves less playable choices to the system is more clever for the environment).

Theorem 3. *Let R be a feasible set of reactions and let $\varphi_{\mathbb{B}} = \varphi_{\mathcal{T}}[l_i \leftarrow s_i] \wedge \varphi^{extra}(R)$ be the Booleanization of $\varphi_{\mathcal{T}}$ using R. Then, $\varphi_{\mathcal{T}}$ and $\varphi_{\mathbb{B}}$ are equi-realizable.*

Impact of Boolean Abstractions on Permissivity. To obtain an equi-realizable Boolean abstraction, it is not necessary to consider *VR*, and instead a feasible set is sufficient. The computation of a feasible set is faster, and generates smaller $\varphi_{\mathbb{B}}$ formulae (Rodriguez and Sánchez 2023). However, there is a price to pay in terms of how permissive the shield is. In practice, it regularly happens that the environment plays moves that are not optimal in the sense that other moves would leave less choice to the system.

Example 3. *Recall Ex. 1. Note that a $C_{\mathbb{B}}$ from $\varphi_{\mathbb{B}}$ can respond with f_c of choices for e_0, e_0^+ and e_1. Now consider a $\varphi'_{\mathbb{B}}$ that ignores e_0^+ and its eligible f_c. Thus, a $C_{\mathbb{B}}$' from $\varphi'_{\mathbb{B}}$ can only respond with f_c for e_0 and e_1. For the sake of the argument, consider the input $x : 0$ forces to satisfy f_{c_2} in $C_{\mathbb{B}}$, or f_{c_2} or f_{c_3} in $C_{\mathbb{B}}$'. Thus, a candidate output $y : 2$ corresponds to holding $((x < 10) \wedge \neg(y > 9) \wedge \neg(y \leq x))$ which is exactly f_{c_3} allowed by e_0^+, but not by e_0 (considered by $\varphi_{\mathbb{B}}$ but not by $\varphi'_{\mathbb{B}}$). Therefore, the corresponding $S_{\mathcal{T}}$ using the winning region W for $\varphi'_{\mathbb{B}}$ would override the output candidate $y : 2$ provided by D (generating an output $v'_{\overline{y}}$ such that $v'_{\overline{y}}$ holds f_{c_2} ; e.g., $y' : -1$); i.e., it incorrectly interprets that candidate $v_{\overline{y}}$ is dangerous, whereas $S_{\mathcal{T}}$ using the winning region W for $\varphi_{\mathbb{B}}$ would not override $v_{\overline{y}}$.*

The most permissive shield uses the *WR* of the complete set of valid reactions *VR*, but note that computing *WR* and *VR* is more expensive than $C_{\mathbb{B}}$ and *MVR*. However, for efficiency reasons the most permissive shield is usually not computed, either because (1) the abstraction algorithm for computing valid reactions computes a non-strict covering or because (2) the synthesis of *WR* does not terminate (specially in liveness specifications). Note that not only the cost of constructing $S_{\mathcal{T}}$ is relevant, but also other design decisions: if we want the policy of D to dominate, then we need $S_{\mathcal{T}}$ to be as permissive as possible, whereas if we want $\varphi_{\mathcal{T}}$ to dominate (e.g., in specially critical tasks), then we want $S_{\mathcal{T}}$ to be intrusive. Moreover, we can guarantee maximal permissivity computing *VR* (and *WR*), whereas we can also guarantee maximal intrusion computing *MVR* (and $C_{\mathbb{B}}$). Indeed, it is always possible to compute such *MVR*.

Theorem 4. *Maximally intrusive shield synthesis (i.e., MVR and $C_{\mathbb{B}}$) is decidable.*

In Ex. 3 the alternative $\varphi'_{\mathbb{B}}$ that ignores the (playable) choice e_0^+ is exactly comparing $r_{e_0} = \{c_1, c_2\}$ and $r_{e_0^+} = \{c_1, c_2, c_3\}$ and constructing a feasible R', where $r_{e_0^+} \notin R'$, since $r_{e_0} \subset r_{e_0^+}$. In this case, R' is an *MVR*.

Optimizations in LTL$_{\mathcal{T}}$ Shields

Shields Optimized in $\mathcal{T}$. We know that, given $v_{\overline{x}}$, the **provider** component computes an output $v'_{\overline{y}}$ from c and r such that $v'_{\overline{y}}$ is a model of the formula $\psi = \exists \overline{y}'.\big(f_r(\overline{x}) \rightarrow f_c(\overline{y}', \overline{x})\big)[\overline{x} \leftarrow v_{\overline{x}}]$. Moreover, not only is ψ guaranteed to be satisfiable (by Lemma. 1), but usually has several models. This implies that the engineer can select some $v'_{\overline{y}}$ that are preferable over others, depending on different criteria represented by objective functions, such as ψ_{f^+}, where $f^+ = \min(|y - y'|)$, i.e., the objective function that minimizes the distance between v_y by D and v'_y by $S'_{\mathcal{T}}$. This is because these are not linear properties. However, this optimization would be very relevant in the context of shielding, because, without loss of generality, it expresses that $v'_{\overline{y}}$ is the safe correction by $S_{\mathcal{T}}$ closest to the unsafe $v_{\overline{y}}$ by D. To solve this, we used maximum satisfiability, which adds *soft constraints* $\mathcal{M} = \{\phi_1, \phi_2, ...\}$ to ψ, such that $\psi(\mathcal{M}) = \exists \overline{y}'.(f_c(\overline{x}, \overline{y}')^+ \bigwedge_{i=0}^{|\mathcal{M}|} \phi_i)[\overline{x} \leftarrow v_{\overline{x}}]$, where $^+ \bigwedge$ denotes a soft conjunction, meaning that the right-hand side is satisfied only if possible. To better illustrate this, we use a single variable y, although this concept can also be extended to other notions of distance with multiple variables $\overline{y}$ (e.g., Euclidean distance) .

Example 4. *Consider again Ex. 3 and let $\mathcal{M} = \{(y' > 5)\}$. Then $\psi(\mathcal{M}) = \exists y'.(f_c(x, y')^+ \wedge (y' > 5))[x \leftarrow 0]$ does not return $y' : 2$, but some $y' \in [6, 9]$. In addition, we use maximum satisfiability to express that v'_y is the safe correction that is the closest to an unsafe v_y, for which we add $\phi(\overline{x}, y, y') = \forall z.(f_c(\overline{x}, z) \rightarrow (|y' - y| < |z - y|))$ to $\mathcal{M}$ in $\psi(\mathcal{M})$, so that $\psi(\mathcal{M}) = (f_c(\overline{x}, y) \wedge (y' > 5) \wedge \phi(\overline{x}, y, y'))[\overline{x} \leftarrow v_{\overline{x}}, y \leftarrow v_y]$. Thus, given a $y : 4$ labelled*

In our extended paper (Rodriguez et al. 2024) we include a formal discussion on how to use optimizations depending on $\mathcal{T}$.

as unsafe, $\psi(\mathcal{M})$ will not return an arbitrary model $y' \in [6, 9]$, but the concrete $y' : 6$. This converges in $\mathcal{T}_{\mathbb{Z}}$.

Note that to enforce $v'_{\overline{y}}$ closest to $v_{\overline{y}}$, we need to additionally extended the **provider** component with input $v_{\overline{y}}$. Also, note that two soft constraints can be contrary to each other, in which case the engineer has to establish priorities using weights. It is also important to note that using soft constraints does not compromise correctness, as any solution found by the solver that supports soft constraints will satisfy the hard constraints as well. Hence, the correctness of Thm. 1 and Thm. 2 remains in-place. Note that not only $S_{\mathcal{T}}$ can minimize the distance to D in arithmetic $\mathcal{T}$, but in any $\mathcal{T}$ for which engineers define a metric space.

Permissive Optimization. We showed that a $S_{\mathcal{T}}$ can provide an output $v'_{\overline{y}}$ that is the safe correction by closest to the unsafe $v_{\overline{y}}$ by D. Moreover, previously we showed that we can generate multiple strategies using *WR*. Therefore, we can optimize $v'_{\overline{y}}$ using combinations in the *WR*; for instance, return $v'_{\overline{y}}$ such that distance to $v_{\overline{y}}$ is minimal and $v'_{\overline{y}}$ has been chosen among the strategies represented by automata with less than n states (i.e., using bounded synthesis with bound n). We illustrate this in (Rodriguez et al. 2024).

Theorem 5. *Let $\mathcal{T} = \mathcal{T}_{\mathbb{Z}}$ and let a candidate output $v_{\overline{y}}$ considered unsafe. Let a WR in an arbitrary state q_k and an input $v_{\overline{x}}$, which yields a set $C = \{c_k, c_j, c_i, ...\}$ of choices that are safe for the system. We denote with $F = \{f_{c_k}, f_{c_j}, fc_i\}$ the set of characteristic choice functions. There is always a value $v'_{\overline{y}}$ of y' satisfying $f_r(v_{\overline{x}}) \rightarrow f_c(y', v_{\overline{x}})$, where $f_c \in F$, such that for $v''_{\overline{y}}$ of y'' satisfying $f_r(v_{\overline{x}}) \rightarrow f'_c(y'', v_{\overline{x}})$, where $f'_c \in F$ and $f_c \neq f'_c$, then the distance from $v'_{\overline{y}}$ to $v_{\overline{y}}$ is smaller or equal than the distance from $v''_{\overline{y}}$ to $v_{\overline{y}}$.*

Note that Thm. 5 holds for other decidable $\mathcal{T}$. This advantage is unique to $S_{\mathcal{T}}$ and it offers yet another permissivity layer to measure distance with respect to the policy of D.

Related Work and Conclusion

Related Work. Classic shielding approaches (Bloem et al. 2015; Könighofer et al. 2017; Alshiekh et al. 2018; Avni et al. 2019) focus on properties expressed in Boolean LTL, and are incompatible for systems with richer-data domains: i.e., they need explicit **manual** discretization of the requirements. In the other hand, we have a sound procedure that directly takes LTL$_{\mathcal{T}}$ specifications (via (Rodriguez and Sánchez 2024)) which we adapted to shielding. This is fundamentally new, and also that we can also optimize outputs with respect to erroneous candidates.

Competing methods The approach by (Wu et al. 2019) is, to the best of our knowledge, the only previous successful attempt to compute rich-data shields. We compared our work with theirs: for the 13 different specifications from (Wu et al. 2019), we generated our shield and measured time for computing: (i) the Boolean step (clm. $\mathbb{B}$), and (ii) producing the final output (clm. $\mathcal{T}$). We compared our results at Tab. 1. For the first task, our approach, on average, requires less than $0.21\mu s$, while they take more than twice as long on average (with over $0.47\mu s$). Our approach was also more efficient in

Req.	(Wu et al.)		*Ours*					
	$\mathbb{B}$	$\mathcal{T}$	$\mathbb{B}$	$\mathbb{B}'$	$\mathcal{T}$	$\mathcal{T}_{\mathbb{Z}}$	$\mathcal{T}_A$	$\mathcal{T}_B$
φ_0	30	631	19	17	171	173	183	184
φ_1	41	590	22	21	190	192	199	212
φ_2	80	520	24	20	214	194	214	199
φ_3	45	340	22	18	106	105	120	105
φ_4	50	640	21	17	118	122	120	117
φ_5	80	520	21	16	124	124	127	156
φ_6	37	370	16	14	141	156	148	151
φ_7	49	710	20	20	163	159	165	173
φ_8	45	580	21	15	133	134	143	189
φ_9	18	610	14	11	170	179	172	203
φ_{10}	50	690	25	19	173	168	182	177
φ_{11}	31	530	21	17	177	178	191	177
φ_{12}	57	510	29	22	156	155	179	182

Table 1: Comparison of our approach with (Wu et al. 2019), measured in $0.1\mu s$ for $\mathbb{B}$ and in μs for $\mathcal{T}$, $\mathcal{T}_{\mathbb{Z}}$, $\mathcal{T}_A$ and $\mathcal{T}_B$.

the second task, taking on average about $0.157ms$, whereas they take required $0.557ms$ on average. Moreover, note that our ($\mathbb{B}$) contains both detection and correction of candidate outputs, whereas their ($\mathbb{B}$) is only detection. It is important to note another key advantage of our approach: (Wu et al. 2019) necessarily generates a *WR*, whereas we can also construct $C_{\mathcal{T}}$ (clm. $\mathbb{B}'$), performs considerably faster (and can operate leveraging highly matured techniques, e.g., bounded synthesis). In addition to this, we (Wu et al. 2019) presents a monolithic method that to be used with specifications containing only linear real arithmetic $\mathcal{T}_{\mathbb{Z}}$. Hence, if we slightly modify $\mathcal{T}$ to $\mathcal{T}_{\mathbb{Z}}$, then the method in (Wu et al. 2019) cannot provide an appropriate shield, whereas we do (clm. $\mathcal{T}_{\mathbb{Z}}$). Indeed, our method encodes *any* arbitrary $\exists^*\forall^*$ decidable fragment of $\mathcal{T}$ (e.g., non-linear arithmetic or the array property fragment (Bradley, Manna, and Sipma 2006)). Additionally, we can optimize the outputs of $S_{\mathcal{T}}$ with respect to different criteria, such as returning the smallest/greatest safe output (clm. $\mathcal{T}_A$) and the output closest to the candidate (clm. $\mathcal{T}_B$). For this, we used Z3 with optimization (Bjørner, Phan, and Fleckenstein 2015). Tab. 1 shows that our approach can manage both rich data and temporal dynamics.

Conclusion. In this work, we present the first general methods for shielding properties encoded in LTL$_{\mathcal{T}}$. These allows engineers to guarantee the safe behavior of DRL agents in complex, reactive environments. Specifically, we demonstrate how shields can be computed from a controller and from a winning region. This is not simply a direct application of synthesis, instead, we can guarantee maximally and minimally permissive shields, as well as shields optimized with respect to objective functions in arbitrary decidable $\mathcal{T}$. We also empirically demonstrate its applicability, (see (Corsi et al. 2024) for an actual application).

A next step is to adapt shields modulo theories to probabilistic settings (Pranger et al. 2021; Carr et al. 2023) and planning (Camacho, Bienvenu, and McIlraith 2019). We also want to leverage recent results on finite-trace LTL$_{\mathcal{T}}$ (Geatti et al. 2023), to model more expressive shields.

Acknowledgments

The work of Rodríguez and Sánchez was funded in part by PRODIGY Project (TED2021-132464B-I00) — funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR — by the DECO Project (PID2022-138072OB-I00) — funded by MCIN/AEI/10.13039/501100011033 and by the ESF, as well as by a research grant from Nomadic Labs and the Tezos Foundation. This work was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The work of Amir was further supported by a scholarship from the Clore Israel Foundation as well as a Rothschild Fellowship granted by the Yad Hanadiv Foundation.

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence*, 2669–2678.

Avni, G.; Bloem, R.; Chatterjee, K.; Henzinger, T. A.; Könighofer, B.; and Pranger, S. 2019. Run-Time Optimization for Learned Controllers Through Quantitative Games. In *Proc. of the 31st International Conference in Computer Aided Verification (CAV), Part I*, volume 11561 of *LNCS*, 630–649. Springer.

Bassan, S.; Amir, G.; Corsi, D.; Refaeli, I.; and Katz, G. 2023. Formally Explaining Neural Networks within Reactive Systems. In *Proc. 23rd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 10–22.

Bernreiter, M.; Maly, J.; and Woltran, S. 2021. Choice Logics and Their Computational Properties. In *Proc. of the 30th Int. Joint Conf. on Artificial Intelligence, (IJCAI 2021)*, 1794–1800.

Bjørner, N.; Phan, A.; and Fleckenstein, L. 2015. νZ - An Optimizing SMT Solver. In *Proc. of the 21st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9035, 194–199.

Bloem, R.; Könighofer, B.; Könighofer, R.; and Wang, C. 2015. Shield Synthesis: - Runtime Enforcement for Reactive Systems. In *Proc. of the 21st Int. Conf. in Tools and Algorithms for the Construction and Analysis of Systems, (TACAS)*, volume 9035, 533–548.

Bradley, A.; Manna, Z.; and Sipma, H. 2006. What's Decidable About Arrays? In *Proc. of the 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 427–442.

Brenguier, R.; Pérez, G. A.; Raskin, J.; and Sankur, O. 2014. AbsSynthe: abstract synthesis from succinct safety specifications. In *Proc. of the 3rd Workshop on Synthesis, (SYNT 2014)*, volume 157 of *EPTCS*, 100–116.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a Unified View of AI Planning and Reactive Synthesis. In *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 58–67. AAAI Press.

Carr, S.; Jansen, N.; Junges, S.; and Topcu, U. 2023. Safe Reinforcement Learning via Shielding under Partial Observability. In *Proc. of the 37th Conference on Artificial Intelligence (AAAI 2023)*, 14748–14756. AAAI Press.

Corsi, D.; Amir, G.; Rodríguez, A.; Katz, G.; Sánchez, C.; and Fox, R. 2024. Verification-Guided Shielding for Deep Reinforcement Learning. *RLJ*, 4: 1759–1780.

Geatti, L.; Gianola, A.; Gigante, N.; and Winkler, S. 2023. Decidable Fragments of $\mathrm{LTL_f}$ Modulo Theories. In *Proc. of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 811–818. IOS Press.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. Technical Report. http://arxiv.org/abs/1412.6572.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, 97–117.

Könighofer, B.; Alshiekh, M.; Bloem, R.; Humphrey, L. R.; Könighofer, R.; Topcu, U.; and Wang, C. 2017. Shield synthesis. *Formal Methods Syst. Des.*, 51(2): 332–361.

Manna, Z.; and Pnueli, A. 1995. *Temporal Verification of Reactive Systems: Safety*. Springer Science & Business Media.

Marchesini, E.; and Farinelli, A. 2020. Discrete deep reinforcement learning for mapless navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*.

Meyer, P.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit Reactive Synthesis Strikes Back! In *Proc. of the 30th Int. Conf. on Computer Aided Verification (CAV)*, 578–586.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of Reactive (1) Designs. In *Proc. 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 364–380.

Pnueli, A. 1977. The Temporal Logic of Programs. In *Proc. of 18th Annual Symposium on Foundations of Computer Science (SFCS)*, 46–57.

Pranger, S.; Könighofer, B.; Posch, L.; and Bloem, R. 2021. TEMPEST - Synthesis Tool for Reactive Systems and Shields in Probabilistic Environments. In *Proc. 19th Int. Symposium in Automated Technology for Verification and Analysis, (ATVA)*, volume 12971, 222–228.

Rodriguez, A.; Amir, G.; Corsi, D.; Sanchez, C.; and Katz, G. 2024. Shield Synthesis for LTL Modulo Theories . Technical Report. https://arxiv.org/abs/2406.04184.

Rodriguez, A.; and Sánchez, C. 2023. Boolean Abstractions for Realizabilty Modulo Theories. In *Proc. of the 35th International Conference on Computer Aided Verification (CAV'23)*, volume 13966 of *LNCS*. Springer, Cham.

Rodriguez, A.; and Sánchez, C. 2024. Adaptive Reactive Synthesis for LTL and LTLf Modulo Theories. In *Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024).*

Rodríguez, A.; and Sánchez, C. 2024. Realizability Modulo Theories. *Journal of Logical and Algebraic Methods in Programming*, 100971.

Schewe, S.; and Finkbeiner, B. 2007. Bounded Synthesis. In *Proc. of the 5th International Symposium in Automated Technology for Verification and Analysis (ATVA 2007)*, volume 4762 of *LNCS*, 474–488. Springer.

Thomas, W. 2008. Church's Problem and a Tour Through Automata Theory. In *Pillars of Computer Science*, 635–655. Springer.

Wu, M.; Wang, J.; Deshmukh, J.; and Wang, C. 2019. Shield Synthesis for Real: Enforcing Safety in Cyber-Physical Systems. In *Proc. of 19th Formal Methods in Computer Aided Design, (FMCAD 2019), San Jose, CA, USA, October 22-25, 2019*, 129–137. IEEE.