LTL_f Synthesis Under Environment Specifications for Reachability and Safety Properties

Benjamin Aminof[1](✉), Giuseppe De Giacomo[1,2](✉), Antonio Di Stasio[2](✉), Hugo Francon[3](✉), Sasha Rubin[4](✉), and Shufang Zhu[2](✉)

[1] Sapienza University of Rome, Rome, Italy
benj@forsyte.at
[2] University of Oxford, Oxford, UK
{giuseppe.degiacomo,antonio.distasio,shufang.zhu}@cs.ox.ac.uk
[3] ENS Rennes, Rennes, France
hugo.francon@ens-rennes.fr
[4] The University of Sydney, Camperdown, Australia
sasha.rubin@sydney.edu.au

Abstract. In this paper, we study LTL_f synthesis under environment specifications for arbitrary reachability and safety properties. We consider both kinds of properties for both agent tasks and environment specifications, providing a complete landscape of synthesis algorithms. For each case, we devise a specific algorithm (optimal wrt complexity of the problem) and prove its correctness. The algorithms combine common building blocks in different ways. While some cases are already studied in literature others are studied here for the first time.

1 Introduction

Synthesis under environment specifications consists of synthesizing an agent strategy (aka plan or program) that realizes a given task against all possible environment responses (i.e., environment strategies). The agent has some indirect knowledge of the possible environment strategies through an environment specification, and it will use such knowledge to its advantage when synthesizing its strategy [2,4,9,24]. This problem is tightly related to planning in adversarial nondeterministic domains [20], as discussed, e.g., in [10,15].

In this paper, we study synthesis under environment specifications, considering both *agent task specifications* and *environment specifications* expressed in Linear Temporal Logic on finite traces (LTL_f). These are logics that look at finite traces or finite prefixes of infinite traces. For concreteness, we focus on LTL_f [16,17], but the techniques presented here extend immediately to other temporal logics on finite traces, such as Linear Dynamic Logics on finite traces, which is more expressive than LTL_f [16], and Pure-Past LTL, which has the same expressiveness as LTL but evaluates a trace backward from the current instant [11].

Linear temporal logics on finite traces provide a nice embodiment of the notable triangle among Logics, Automata, and Games [21]. These logics are full-

V. Malvone and A. Murano (Eds.): EUMAS 2023, LNAI 14282, pp. 263–279, 2023.
https://doi.org/10.1007/978-3-031-43264-4_17

fledged logics with high expressiveness over finite traces, and they can be translated into classical regular finite state automata; moreover, they can be further converted into deterministic finite state automata (DFAs). This transformation yields a game represented on a graph. In this game, one can analyze scenarios where the objective is to reach certain final states. Finally, despite the fact that producing a DFA corresponding to an LTL$_f$ formula can require double-exponential time, the algorithms involved—generating alternating automata (linear), getting the nondeterministic one (exponential), determinizing it (exponential), solving reachability games (poly)—are particularly well-behaved from the practical computational point of view [26,28,32].

In this paper, however, we consider LTL$_f$ specifications in two contexts which we denote as

$$\exists\varphi \text{ and } \forall\varphi \text{ with } \varphi \text{ an arbitrary LTL}_f \text{ formula.}$$

The first one specifies a *reachability* property: there exists a finite prefix $\pi_{<k}$ of an infinite trace π such that $\pi_{<k} \models \varphi$. This is the classical use of LTL$_f$ to specify synthesis tasks [17]. The second one specifies a *safety* property: every finite prefix $\pi_{<k}$ of an infinite trace π is such that $\pi_{<k} \models \varphi$. This is the classical use of LTL$_f$ to specify environment behaviours [1,13]. The formulas $\forall\varphi$ and $\exists\varphi$ with φ in LTL$_f$ capture exactly two well-known classes of LTL properties in Manna and Pnueli's Temporal Hierarchy [23]. Specifically, $\exists\varphi$ captures the *co-safety properties* and $\forall\varphi$ captures the *safety properties* (in [23], expressed respectively as $\Diamond\psi$ and $\Box\psi$ with ψ an arbitrary Pure-Past LTL formulas, which consider only past operators.)

We let Env and Task denote an environment specification and a task specification, respectively, consisting of a safety ($\forall\varphi$) and/or reachability property ($\exists\varphi$). This gives rise to 12 possible cases: 3 without any environment specifications, 3 with safety environment specifications ($\forall\varphi$), 3 with reachability environment specifications ($\exists\varphi$), and 3 with both safety and reachability environment specifications ($\exists\varphi \wedge \forall\varphi$). For each of these, we provide an algorithm, which is optimal wrt the complexity of the problem, and prove its correctness. When the problem was already solved in literature, we give appropriate references (e.g., $\mathsf{Task} = \exists\varphi$ and $\mathsf{Env} = \mathit{true}$ is classical LTL$_f$ synthesis, solved in [17]). In fact, we handle all the cases involving reachability in the environment specifications by providing a novel algorithm that solves the most general case of $\mathsf{Env} = \exists\varphi_1 \wedge \forall\varphi_2$ and $\mathsf{Task} = \exists\varphi_3 \wedge \forall\varphi_4$[1].

These algorithms use the common building blocks (combining them in different ways): the construction of the DFAs of the LTL$_f$ formulas, Cartesian products of such DFAs, considering these DFAs as the game arena and solving games for reachability/safety objectives. Also, all these problems have a 2EXPTIME-complete complexity. The hardness comes from LTL$_f$ synthesis [17], and the membership comes from the LTL$_f$-to-DFA construction, which dominates the complexity since computing the Cartesian products and solving reachability/safety games

[1] In fact, this algorithm can solve all cases, but it's much more involved compared to the direct algorithms we provide for each case.

is polynomial[2]. Towards the actual application of our algorithms, we observe that although the DFAs of LTL$_f$ formulas are worst-case double-exponential, there is empirical evidence showing that the determinization of NFA, which causes one of the two exponential blow-ups, is often polynomial in the NFA [27,28,32]. Moreover, in several notable cases, e.g., in all DECLARE patterns [29], the DFAs are polynomial in the LTL$_f$ formulas, and so are our algorithms.

2 Preliminaries

Traces. For a finite set Σ, let Σ^ω (resp. Σ^+, Σ^*) denote the set of infinite strings (resp. non-empty finite strings, finite strings) over Σ. We may write concatenation of sets using $\cdot$, e.g., $\Sigma \cdot \Sigma$ denotes the set of strings over Σ of length 2. The length of a string is denoted $|\pi|$, and may be infinite. Strings are indexed starting at 0. For a string π and $k \in \mathbb{N}$ with $k < |\pi|$, let $\pi_{<k}$ denote the finite prefix of π of length k. For example, if $\pi = \pi_0\pi_1 \dots \pi_n$, then $|\pi| = n + 1$ and $\pi_{<2} = \pi_0\pi_1$. Typically, Σ will be the set of interpretations (i.e., assignments) over a set *Prop* of atomic propositions, i.e., $\Sigma = 2^{Prop}$. Non-empty strings will also be called *traces*.

Linear-Time Temporal Logic on Finite Traces. LTL$_f$ is a variant of Linear-time temporal logic (LTL) interpreted over *finite*, instead of infinite, traces [16]. Given a set *Prop* of atomic propositions, LTL$_f$ formulas φ are defined by the following grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi$ where $p \in Prop$ denotes an atomic proposition, $\bigcirc$ is read *next*, and $\mathcal{U}$ is read *until*. We abbreviate other Boolean connectives and operators.

For a finite trace $\pi \in (2^{Prop})^+$, an LTL$_f$ formula φ, and a position i ($0 \leq i < |\pi|$), define $\pi, i \models \varphi$ (read "φ *holds* at position i") by induction, as follows:

- $\pi, i \models p$ iff $p \in \pi_i$ (for $p \in Prop$);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \bigcirc\varphi$ iff $i < |\pi| - 1$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff for some j ($i \leq j < |\pi|$), $\pi, j \models \varphi_2$, and for all k ($i \leq k < j$), $\pi, k \models \varphi_1$.

We write $\pi \models \varphi$, if $\pi, 0 \models \varphi$ and say that π *satisfies* φ. Write $\mathcal{L}(\varphi)$ for the set of finite traces over $\Sigma = 2^{Prop}$ that satisfy φ. In addition, we define the *weak next* operator $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$. Note that: $\neg\bigcirc\varphi$ is not, in general, logically equivalent to $\bigcirc\neg\varphi$, but we have that $\neg\bigcirc\varphi \equiv \bullet\neg\varphi$.

Domains. A *domain (aka transition system, aka arena)* is a tuple $\mathcal{D} = (\Sigma, Q, \iota, \delta)$, where Σ is a finite alphabet, Q is a finite set of states, $\iota \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function. For an infinite string

[2] For pure-past LTL, obtaining the DFA from a pure-past LTL formula is single exponential [11], and indeed the problems and all our algorithms become EXPTIME-complete.

$w = w_0 w_1 w_2 \ldots \in \Sigma^\omega$ a *run* of $\mathcal{D}$ on w is a sequence $r = q_0 q_1 q_2 \ldots \in Q^\omega$ that $q_0 = \iota$ and $q_{i+1} \in \delta(q_i, w_i)$ for every i with $0 \leq i$. A *run* of $\mathcal{D}$ on a finite string $w = w_0 w_1 \ldots w_n$ over Σ is a sequence $q_0 q_1 \cdots q_{n+1}$ such that $q_0 = \iota$ and $q_{i+1} \in \delta(q_i, w_i)$ for every i with $0 \leq i < n+1$. Note that every string has exactly one run of $\mathcal{D}$.

Deterministic Finite Automaton (DFA). A *DFA* is a tuple $\mathcal{M} = (\mathcal{D}, F)$ where $\mathcal{D}$ is a domain and $F \subseteq Q$ is a set of *final* states. A finite word w over Σ is *accepted* by $\mathcal{M}$ if the run of $\mathcal{M}$ on w ends in a state of F. The set of all such finite strings is denoted $\mathcal{L}(\mathcal{M})$, and is called the *language* of $\mathcal{M}$.

Theorem 1. [17] *Every* LTL$_f$ *formula* φ *over atoms Prop can be translated into a* DFA $\mathcal{M}_\varphi$ *over alphabet* $\Sigma = 2^{Prop}$ *such that for every finite string* π *we have that* $\pi \in \mathcal{L}(\mathcal{M})$ *iff* $\pi \models \varphi$. *This translation takes time double-exponential in the size of* φ.

Properties of Infinite Strings. A *property* is a set P of infinite strings over Σ, i.e., $P \subseteq \Sigma^\omega$. We say that P is a *reachability* property if there exists a set $T \subseteq \Sigma^+$ of finite traces such that if $w \in P$ then some finite prefix of w is in T. We say that P is a *safety* property if there exists a set $T \subseteq \Sigma^+$ of finite traces such that if $w \in P$, then every finite prefix of w is in T. It is worth noting that the complement of a reachability property is a safety property, and vice versa.

An LTL$_f$ formula can be used to denote a reachability (resp., safety) property over $\Sigma = 2^{Prop}$ as follows.

Definition 1. *For an* LTL$_f$ *formula* φ, *let* $\exists\varphi$ *denote set of traces* π *such that some finite prefix of* π *satisfies* φ, *and let* $\forall\varphi$ *denote set of traces* π *such that every finite (non-empty) prefix of* π *satisfies* φ.

Note that $\exists\varphi$ denotes a reachability property, and $\forall\varphi$ denotes a safety property. From now on, "prefix" will mean "finite non-empty prefix". Note also that for an LTL$_f$ formula, $\mathcal{L}(\varphi)$ is a set of finite traces. On the other hand, $\mathcal{L}(\exists\varphi)$ (and similarly $\mathcal{L}(\psi)$ where ψ is a Boolean combination of formulas of the form $\exists\varphi$ for LTL$_f$ formulas φ) is a set of infinite traces. In this paper, we consider $\exists\varphi$, $\forall\varphi$, and $\exists\varphi \wedge \forall\varphi$ to specify both agent tasks and environment behaviours.

Deterministic Automata on Infinite Strings (DA). Following the automata-theoretic approach in formal methods, we will compile formulas to automata. We have already seen that we can compile LTL$_f$ formulas to DFAs. We now introduce automata over infinite words to handle certain properties of infinite words. A *deterministic automaton* (DA, for short) is a tuple $\mathcal{A} = (\mathcal{D}, \alpha)$ where $\mathcal{D}$ is a transition system, say with the state set Q, and $\alpha \subseteq Q^\omega$ is called an *acceptance condition*. An infinite string w is *accepted* by $\mathcal{A}$ if its run is in α. The set of all such infinite strings is denoted $\mathcal{L}(\mathcal{A})$, and is called the *language* of $\mathcal{A}$.

We consider reachability (reach) and safety (safe) acceptance conditions, parameterized by a set of target states $T \subseteq Q$:

- reach$(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \exists k \geq 0 : q_k \in T\}$. In this case, we call $\mathcal{A}$ a *reachability automaton.*
- safe$(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \forall k \geq 0 : q_k \in T\}$. In this case, we call $\mathcal{A}$ a *safety automaton.*

Remark 1. Every reachability (resp. safety) property expressible in LTL is the language of a reachability automaton (resp. safety automaton) [16,22,25].

3 Problem Description

Reactive Synthesis. Reactive Synthesis (aka Church's Synthesis) is the problem of turning a specification of an agent's task and of its environment into a strategy (aka policy). This strategy can be employed by the agent to achieve its task, regardless of how the environment behaves. In this framework, the agent and the environment are considered players in a turn-based game, in which players move by picking an evaluation of the propositions they control. Thus, we partition the set *Prop* of propositions into two disjoint sets of propositions $\mathcal{X}$ and $\mathcal{Y}$, and with a little abuse of notation, we denote such a partition as $Prop = \mathcal{Y} \cup \mathcal{X}$. Intuitively, the propositions in $\mathcal{X}$ are controlled by the environment, and those in $\mathcal{Y}$ are controlled by the agent. In this work (in contrast to the usual setting of reactive synthesis), the agent moves first. The agent moves by selecting an element of $2^{\mathcal{Y}}$, and the environment responds by selecting an element of $2^{\mathcal{X}}$. This is repeated forever, and results in an infinite trace (aka play). From now on, unless specified otherwise, we let $\Sigma = 2^{Prop}$ and $Prop = \mathcal{Y} \cup \mathcal{X}$. We remark that the games considered in this paper are games of perfect information with deterministic strategies.

An *agent strategy* is a function $\sigma_{\mathsf{ag}} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$. An environment strategy is a function $\sigma_{\mathsf{env}} : (2^{\mathcal{Y}})^+ \to 2^{\mathcal{X}}$. A strategy σ is *finite-state* (aka *finite-memory*) if it can be represented as a finite-state input/output automaton that, on reading an element h of the domain of σ, outputs the action $\sigma(h)$. A trace $\pi = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots \in (2^{\mathcal{Y} \cup \mathcal{X}})^\omega$ *follows an agent strategy* $\sigma_{\mathsf{ag}} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ if $Y_0 = \sigma_{\mathsf{ag}}(\epsilon)$ and $Y_{i+1} = \sigma_{\mathsf{ag}}(X_0 X_1 \ldots X_i)$ for every $i \geq 0$, and it *follows an environment strategy* σ_{env} if $X_i = \sigma_{\mathsf{env}}(Y_0 Y_1 \ldots Y_i)$ for all $i \geq 0$. We denote the unique infinite sequence (play) that follows σ_{ag} and σ_{env} as $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$. Let P be a property over the alphabet $\Sigma = 2^{Prop}$, specified by formula or DA. An agent strategy σ_{ag} (resp., environment strategy σ_{env}) *enforces* P if for every environment strategy σ_{env} (resp., agent strategy σ_{ag}), we have that $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$ is in P. In this case, we write $\sigma_{\mathsf{ag}} \rhd P$ (resp. $\sigma_{\mathsf{env}} \rhd P$). We say that P is *agent (resp., environment) realizable* if there is an agent (resp. environment) strategy that enforces P.

Synthesis Under Environment Specifications. Typically, an agent has some knowledge of how the environment works, represented as a fully observable model of the environment, which it can exploit to enforce its task [2]. Formally, let Env and Task be properties over alphabet $\Sigma = 2^{Prop}$, denoting the environment specification and the agent task, respectively.

Note that while the agent task Task denotes the set of desirable traces from the agent's perspective, the environment specification Env denotes the set of environment strategies that describe how the environment reacts to the agent's actions (no matter what the agent does) in order to enforce Env. Specifically, Env is treated as a set of traces when we reduce the problem of synthesis under environment specification to standard reactive synthesis.

We require a consistency condition of Env, i.e., there must exist at least one environment strategy $\sigma_{\mathsf{env}} \rhd \mathsf{Env}$. An agent strategy σ_{ag} enforces Task *under the environment specification* Env, written $\sigma_{\mathsf{ag}} \rhd_{\mathsf{Env}} \mathsf{Task}$, if for all $\sigma_{\mathsf{env}} \rhd \mathsf{Env}$ we have that $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \mathsf{Task}$. Note that if $\mathsf{Env} = \mathit{true}$ then this just says that σ_{ag} enforces Task (i.e., the environment specification is missing).

Definition 2 (Synthesis under environment specifications). *Let* Env *and* Task *be properties over alphabet* $\Sigma = 2^{Prop}$, *denoting the environment specification and the agent task, respectively. (i) The* realizability under environment specifications problem *asks, given* Task *and* Env, *to decide if there exists an agent strategy enforcing* Task *under the environment specification* Env. *(ii) The* synthesis under environment specifications problem *asks, given* Task *and* Env, *to return a finite-state agent strategy enforcing* Task *under the environment specification* Env, *or say that none exists.*

In [2] is shown that for any linear-time property[3], synthesis under environment specifications can be reduced to synthesis without environment specifications. Thus, in order to show that Task is realizable under Env it is sufficient to show that $\mathsf{Env} \rightarrow \mathsf{Task}$ is realizable. Moreover, to solve the synthesis problem for Task under Env, it is enough to return a strategy that enforces $\mathsf{Env} \rightarrow \mathsf{Task}$.

Table 1. Task and Env considered. Note that, from Alg. 7 we get the remaining cases involving reachability environment specifications by suitably setting $\varphi_1, \varphi_2, \varphi_4$ to *true*.

Task	Env	Alg
$\exists\varphi$	*true*	Algorithm 1
$\forall\varphi$	*true*	Algorithm 2
$\exists\varphi_1 \wedge \forall\varphi_2$	*true*	Algorithm 3
$\exists\varphi_1$	$\forall\varphi_2$	Algorithm 4
$\forall\varphi_1$	$\forall\varphi_2$	Algorithm 5
$\exists\varphi_1 \wedge \forall\varphi_2$	$\forall\varphi_3$	Algorithm 6
$\exists\varphi_1 \wedge \forall\varphi_2$	$\exists\varphi_3 \wedge \forall\varphi_4$	Algorithm 7

In the rest of the paper, we provide a landscape of algorithms for LTL_f synthesis considering reachability and safety properties for both agent tasks and

[3] Technically, the properties should be Borel, which all our properties are.

environment specifications. However, these synthesis problems are complex and challenging due to the combination of reachability and safety properties. To tackle this issue, one possible approach is to reduce LTL$_f$ synthesis problems to LTL synthesis problems through suitable translations, e.g., [12,14,30,31]. However, there is currently no methodology for performing such translations when considering combinations of reachability and safety properties[4]. Additionally, synthesis algorithms for LTL specifications are generally more challenging than those for LTL$_f$ specifications, both theoretically and practically [13,14,30,31]. In this paper, we show that for certain combinations, we can avoid the detour to LTL synthesis and keep the simplicity of LTL$_f$ synthesis. Specifically, we consider that Task and Env take the following forms: $\exists\varphi_1, \forall\varphi_1, \exists\varphi_1 \wedge \forall\varphi_2$ where the φ_i are LTL$_f$ formulas, and in addition we consider the case of no environment specification (formally, $\mathsf{Env} = \mathit{true}$). This results in 12 combinations. Algorithms 1–7, listed in Table 1, optimally solve all the combinations. All these algorithms adopt some common building blocks while linking them in different ways.

Theorem 2. *Let each of* Task *and* Env *be of the forms* $\forall\varphi$, $\exists\varphi$, *or* $\exists\varphi_1 \wedge \forall\varphi_2$. *Solving synthesis for an agent* Task *under environment specification* Env *is 2EXPTIME-complete.*

4 Building Blocks for the Algorithms

In this section, we describe the building blocks we will use to devise the algorithms for the problem described in the previous section.

DAs for $\exists\varphi$ *and* $\forall\varphi$. Here, we show how to build the DA whose language is exactly the infinite traces satisfying $\exists\varphi$ (resp. $\forall\varphi$). The first step is to convert the LTL$_f$ formula φ into a DFA $\mathcal{M}_\varphi = (\Sigma, Q, \iota, \delta, F)$ that accepts exactly the *finite* traces that satisfy φ as in Theorem 1. Then, to obtain a DA $\mathcal{A}_{\exists\varphi}$ for $\exists\varphi$ define $\mathcal{A}_{\exists\varphi} = (2^{\mathcal{X}\cup\mathcal{Y}}, Q, \iota, \delta, \mathrm{reach}(F))$. It is immediate that $\mathcal{L}(\exists\varphi) = \mathcal{L}(\mathcal{A}_{\exists\varphi})$. To obtain a DA $\mathcal{A}_{\forall\varphi}$ for $\forall\varphi$ define $\mathcal{A}_{\forall\varphi} = (2^{\mathcal{X}\cup\mathcal{Y}}, Q, \iota, \delta, \mathrm{safe}(F \cup \{\iota\}))$.

The reason ι is considered a part of the safe set is that the DFA $\mathcal{M}_\varphi$ does not accept the empty string since the semantics of LTL$_f$ precludes this. It is immediate that $\mathcal{L}(\forall\varphi) = \mathcal{L}(\mathcal{A}_{\forall\varphi})$. For $\psi \in \{\exists\varphi, \forall\varphi\}$, we let $\textsc{ConvertDA}(\psi)$ denote the resulting DA.

Lemma 1. *Let* φ *be an* LTL$_f$ *formula, and let* $\psi \in \{\exists\varphi, \forall\varphi\}$. *Then the languages* $\mathcal{L}(\psi)$ *and* $\mathcal{L}(\textsc{ConvertDA}(\psi))$ *are equal.*

For formulas of the form $\forall\varphi$ we will suppress the initial state in the objective and so $\textsc{ConvertDA}(\forall\varphi)$ will be written $(\mathcal{D}_{\forall\varphi}, \mathrm{safe}(T))$, i.e., T contains ι.

Games over DA. The synthesis problems we consider in this paper are solved by reducing them to two-player games. We will represent games by DAs $\mathcal{A} =$

[4] In [9] is shown that the case of LTL$_f$ synthesis under safety and reachability properties can be solved by reducing to games on infinite-word automata. This certain case is covered in our paper, nevertheless, we provide a direct approach that only involves games on finite-word automata.

$(\mathcal{D}, \alpha)$ where $\mathcal{D}$ is a transition system, sometimes called an 'arena', and α is an acceptance condition, sometimes called a 'winning condition'. The game is played between an *agent* (controlling $\mathcal{Y}$) and *environment* (controlling $\mathcal{X}$). Intuitively, a position in the game is a state $q \in Q$. The initial position is ι. From each position, first the agent moves by setting $Y \in 2^{\mathcal{Y}}$, then the environment moves by setting $X \in 2^{\mathcal{X}}$, and the next position is updated to the state $\delta(q, Y \cup X)$. This interaction results in an infinite run in $\mathcal{D}$, and the agent is declared the winner if the run is in α (otherwise, the environment is declared the winner).

Definition 3. *An agent strategy σ_{ag} is said to* win the game $(\mathcal{D}, \alpha)$ *if for every trace π that follows σ_{ag}, the run in $\mathcal{D}$ of π is in α.*

In other words, σ_{ag} wins the game if every trace π that follows σ_{ag} is in $L(\mathcal{D}, \alpha)$. For $q \in Q$, let $\mathcal{D}_q$ denote the transition system $\mathcal{D}$ with initial state q, i.e., $\mathcal{D}_q = (\Sigma, Q, q, \delta)$. We say that q is a *winning state* for the agent if there is an agent strategy that wins the game $(\mathcal{D}_q, \alpha)$; in this case, the strategy is said to *win starting from q*.

In the simplest settings, we represent agent strategies as functions of the form $f_{\mathsf{ag}} : Q \to 2^{\mathcal{Y}}$, called positional strategies. An agent positional strategy f_{ag} induces an agent strategy, $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}_q, f_{\mathsf{ag}})$, as follows: define $\sigma_{\mathsf{ag}}(\epsilon) = f_{\mathsf{ag}}(q)$, and for every finite trace π let ρ be the run of $\mathcal{D}_q$ on π (i.e., starting in state q), and define $\sigma_{\mathsf{ag}}(\pi) = f_{\mathsf{ag}}(q')$ where q' is the last state in ρ (i.e., $q' = \rho_{|\pi|}$). In more complex settings, e.g., in the Algorithm 7, we will construct functions of the form $f_{\mathsf{ag}} : Q \cdot (2^{\mathcal{Y}} \cdot 2^{\mathcal{X}} \cdot Q)^* \to 2^{\mathcal{Y}}$, which similarly induce agent strategies $\textsc{Strategy}(\mathcal{D}_q, f_{\mathsf{ag}})$ where for every finite trace $\pi = Y_0 \cup X_0, \cdots, Y_k \cup X_k$, and run $q_0, \cdots, q_{k+1}$ of π in $\mathcal{D}_q$, define $\sigma_{\mathsf{ag}}(\pi) = f_{\mathsf{ag}}(q_0, Y_0 \cup X_0, q_1, Y_1 \cup X_1, \cdots, q_{k+1})$. Below the agent strategy $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}_q, f_{\mathsf{ag}})$ returned by the various algorithms will be finite state, in the sense that it is representable as a transducer. For simplicity, with a little abuse of notation, we will return directly σ_{ag}, instead of its finite representation as a transducer.

Dual definitions can be given for the environment, with the only notable difference being that $f_{\mathsf{env}} : Q \times 2^{X} \to 2^{\mathcal{Y}}$ since the moves of the environment depend also on the last move of the agent (since the agent moves first).

In this paper, besides the terms 'environment' and 'agent', we also consider the terms 'protagonist' and 'antagonist'. If the DA $(\mathcal{D}, \alpha)$ is a specification for the agent, then the agent is called the protagonist and the environment is called the antagonist. On the other hand, if the DA $(\mathcal{D}, \alpha)$ is a specification for the environment, then the environment is called the protagonist, and the agent is called the antagonist. Intuitively, the protagonist is trying to make sure that the generated traces are in $\mathcal{L}(\mathcal{D}, \alpha)$, and the antagonist to make sure that the generated traces are not in $\mathcal{L}(\mathcal{D}, \alpha)$. Define Win_p (resp. Win_a) as the set of states $q \in Q$ such that q is a protagonist (resp. antagonist) winning state. This set is called protagonist's (resp. antagonist) *winning region.* In this paper, all our games (including reachability and safety games) are *determined.* Therefore:

Lemma 2. *For every state $q \in Q$, it holds that $q \in \mathsf{Win}_p$ iff $q \notin \mathsf{Win}_a$.*

The problem of *solving* a game $(\mathcal{D}, \alpha)$ for the protagonist is to compute the winning region Win_p and a function f_p such that $\textsc{Strategy}(\mathcal{D}, f_p)$ wins from every state in Win_p[5]. To do this, we will also sometimes compute a winning strategy for the antagonist (that wins starting in its winning region).

Solving Reachability Games and Safety Games. We repeatedly make use of solutions to reachability games and safety games given by DAs $\mathcal{A}$. Thus, for a protagonist $p \in \{\mathsf{ag}, \mathsf{env}\}$ let $\textsc{Solve}_p(\mathcal{A})$ denote the procedure for solving the game $\mathcal{A}$, i.e., p is trying to ensure the play is in $\mathcal{L}(\mathcal{A})$; this procedure returns the protagonist's winning region Win_p and a function f_p such that $\textsc{Strategy}(\mathcal{D}, f_p)$ wins starting from every state in Win_p [19].

Product of Transition Systems. Let $\mathcal{D}_i$ $(1 \leq i \leq k)$ be transition systems over alphabet Σ. Their *product*, denoted $\textsc{Product}(\mathcal{D}_1, \cdots, \mathcal{D}_k)$, is the transition system $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ defined as follows: (i) The alphabet is Σ. (ii) The state set is $Q = Q_1 \times \cdots \times Q_k$. (iii) The initial state is $\iota = (\iota_1, \cdots, \iota_k)$. (iv) The transition function δ maps a state $(q_1, \cdots, q_k)$ on input $z \in \Sigma$ to the state $(q'_1, \cdots, q'_k)$ where $q'_i = \delta_i(q_i, z)$ $(1 \leq i \leq k)$. Also, the *lift* of a set $F_i \subseteq Q_i$ to $\mathcal{D}$ is the set $\{(q_1, \cdots, q_k) : q_i \in F_i\} \subseteq Q$.

Restriction of a Transition System. The restriction of a transition system, defined as the procedure $\textsc{Restriction}(\mathcal{D}, S)$, restricts $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ to $S \subseteq Q$ is the transition system $\mathcal{D}' = (\Sigma, S \cup \{sink\}, \iota, \delta', \alpha')$ where for all $z \in \Sigma$, $\delta'(sink, z) = sink$, $\delta'(q, z) = \delta(q, z)$ if $\delta(q, z) \in S$, and $\delta'(q, z) = sink$ otherwise. Intuitively, $\mathcal{D}'$ redirect all transitions from S that leave S to a fresh *sink* state. We may denote the sink by $\bot$[6].

5 Reachability Tasks, No Env Spec

Algorithm 1 solves the realizability and synthesis for the case of reachability tasks and no environment specification. Formally, Task is of the form $\exists\varphi$ where φ is an LTL$_f$ formula, and $\mathsf{Env} = true$. This problem is solved in [17], but here we rephrase the problem in our notation.

Theorem 3. *Algorithm 1 solves the synthesis under environment specifications problem with* $\mathsf{Task} = \exists\varphi, \mathsf{Env} = true$*, where* φ *is an* LTL$_f$ *formula.*

[5] Since strategies can depend on the history, and thus on the starting state in particular, there is always a strategy that wins from every state in Win_p.

[6] We remark that (i) when we restrict the transition system of a DA $(\mathcal{D}, \alpha)$ we may need to revise the winning-condition α to express whether reaching *sink* is good for the protagonist or not (although many times it is not, e.g., when restricting to the winning-region for a safety condition); (ii) in one case, in Algorithm 7, we will add two sink states.

Algorithm 1. $\mathsf{Task} = \exists\varphi, \mathsf{Env} = \mathit{true}$

Input: LTL$_f$ formula φ

Output: agent strategy σ_{ag} that enforces $\exists\varphi$

```
1: $\mathcal{A} = \textsc{ConvertDA}(\exists\varphi)$, say $\mathcal{A} = (\mathcal{D}_{\exists\varphi}, \text{reach}(T))$
2: $(W, f_{\mathsf{ag}}) = \textsc{Solve}_{ag}(\mathcal{A})$
3: if $\iota \notin W$ return "Unrealisable" endif
4: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}_{\exists\varphi}, f_{\mathsf{ag}})$
```

6 Safety Tasks, No Env Spec

Algorithm 2 handles the case Task is of the form $\forall\varphi$ where φ is an LTL$_f$ formula, and $\mathsf{Env} = \mathit{true}$. We can use the result in [17] to solve the synthesis for $\forall\varphi$ from the point of view of the environment.

Algorithm 2. $\mathsf{Task} = \forall\varphi, \mathsf{Env} = \mathit{true}$

Input: LTL$_f$ formula φ

Output: agent strategy σ_{ag} that enforces $\forall\varphi$

```
1: $\mathcal{A}_1 = \textsc{ConvertDA}(\forall\varphi)$, say $\mathcal{A}_1 = (\mathcal{D}_{\forall\varphi}, \text{safe}(T_1))$
2: $(S_1, f_{\mathsf{ag}}) = \textsc{Solve}_{ag}(\mathcal{A}_1)$
3: if $\iota \notin S_1$ return "Unrealisable" endif
4: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}_{\forall\varphi}, f_{\mathsf{ag}})$
```

Theorem 4. *Algorithm 2 solves the synthesis under environment specifications problem with* $\mathsf{Task} = \forall\varphi, \mathsf{Env} = \mathit{true}$*, where* φ *is an* LTL$_f$ *formula.*

7 Reachability and Safety Tasks, No Env Spec

Algorithm 3 handles the case that Task is of the form $\exists\varphi_1 \wedge \forall\varphi_2$ where φ_1 and φ_2 are LTL$_f$ formulas, and $\mathsf{Env} = \mathit{true}$.

Intuitively, the algorithm proceeds as follows. First, it computes the corresponding DA for $\forall\varphi_2$ and solves the safety game over it. The resulting winning area represents the set of states from which the agent has a strategy to realize its safety task. Then, it restricts the game area to the agent's winning area. Finally, it solves the reachability game over the game product of the corresponding DA of $\exists\varphi_1$ and the remaining part of the DA for $\forall\varphi_2$.

In order to obtain the final strategy for the agent we need to refine the strategy f_{ag} to deal with the sink state, call it $\bot_2$, and combine it with g_{ag}. Given f_{ag} computed in Line 3, define $f''_{\mathsf{ag}} : Q_1 \times (S_2 \cup \{\bot_2\}) \to 2^{\mathcal{Y}}$ over $\mathcal{D}$ by $f''_{\mathsf{ag}}(q, s) = f_{\mathsf{ag}}(s)$ if $s \in S_2$, and $f''_{\mathsf{ag}}(q, s) = Y$ (for some arbitrary Y) otherwise. In words, f''_{ag} ensures the second component stays in S_2 (and thus in T_2). Recall that g_{ag} over $\mathcal{D}$ ensures that T_1 is reached in the first co-ordinate, while at the same time maintaining the second co-ordinate is in S_2. Finally, let $\textsc{Combine}(\mathcal{D}, R, g_{\mathsf{ag}}, f_{\mathsf{ag}})$

Algorithm 3. $\mathsf{Task} = \exists\varphi_1 \wedge \forall\varphi_2, \mathsf{Env} = \mathit{true}$

```
Input: LTL_f formulas $\varphi_1$ and $\varphi_2$
Output: agent strategy $\sigma_{\mathsf{ag}}$ that realizes $\exists\varphi_1$ and $\forall\varphi_2$
1: $\mathcal{A}_1 = \textsc{ConvertDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_{\exists\varphi_1}, \mathrm{reach}(T_1))$
2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_{\forall\varphi_2}, \mathrm{safe}(T_2))$
3: $(S_2, f_{\mathsf{ag}}) = \textsc{Solve}_{ag}(\mathcal{A}_2)$
4: $\mathcal{D}'_{\forall\varphi_2} = \textsc{Restrict}(\mathcal{D}_{\forall\varphi_2}, S_2)$, say the sink state is $\bot_2$
5: $\mathcal{D} = \textsc{Product}(\mathcal{D}_{\exists\varphi_1}, \mathcal{D}'_{\forall\varphi_2})$
6: $(R, g_{\mathsf{ag}}) = \textsc{Solve}_{ag}(\mathcal{D}, \mathrm{reach}(T_1 \times S_2))$
7: if $\iota \notin R$ return "Unrealisable" endif
8: $h_{\mathsf{ag}} = \textsc{Combine}(\mathcal{D}, R, g_{\mathsf{ag}}, f_{\mathsf{ag}})$
9: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}, h_{\mathsf{ag}})$
```

denote the final strategy $h_{\mathsf{ag}} : Q_1 \times (S_2 \cup \{\bot_2\}) \rightarrow 2^{\mathcal{Y}}$ defined as follows: $h_{\mathsf{ag}}((q,s)) = g_{\mathsf{ag}}((q,s))$ if $(q,s) \in R$, and $h_{\mathsf{ag}}((q,s)) = f''_{\mathsf{ag}}((q,s))$ otherwise. Intuitively, the agent following h_{ag} will achieve the reachability goal while staying safe, whenever this is possible, and stays safe otherwise.

Theorem 5. *Algorithm 3 solves synthesis under environment specifications problem with* $\mathsf{Task} = \exists\varphi_1 \wedge \forall\varphi_2, \mathsf{Env} = \mathit{true}$*, where the* φ_i *are* LTL$_f$ *formulas.*

8 Reachability Tasks, Safety Env Specs

Algorithm 4 handles the case that Task is of the form $\exists\varphi_1$ and $\mathsf{Env} = \forall\varphi_2$, where φ_1, φ_2 are LTL$_f$ formulas. A similar problem of this case was solved in [13], which, more specifically, considers only finite safety of the agent, i.e., the agent is required to stay safe until some point (the bound is related to an additional agent reachability task), and thus can actually be considered as reachability.

Intuitively, the algorithm first computes all the environment strategies that can enforce $\mathsf{Env} = \forall\varphi_2$ [7], represented as a restriction of the DA for $\forall\varphi_2$, as in the previous section. Then, based on restricting the game arena on these environment strategies, the algorithm solves the reachability game over the product of the corresponding DA of $\exists\varphi_1$ and the restricted part of the DA for $\forall\varphi_2$.

Theorem 6. *Algorithm 4 solves the synthesis under environment specifications problem with* $\mathsf{Task} = \exists\varphi_1, \mathsf{Env} = \forall\varphi_2$*, where the* φ_i *are* LTL$_f$ *formulas.*

Algorithm 4. $\mathsf{Task} = \exists\varphi_1, \mathsf{Env} = \forall\varphi_2$

```
Input: LTL_f formulas $\varphi_1, \varphi_2$
Output: agent strategy $\sigma_{\mathsf{ag}}$ that enforces $\exists\varphi_1$ under $\forall\varphi_2$
 1: $\mathcal{A}_1 = \textsc{ConvertDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_{\exists\varphi_1}, \mathrm{reach}(T_1))$
 2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_{\forall\varphi_2}, \mathrm{safe}(T_2))$
 3: $(S_2, f_{\mathsf{env}}) = \textsc{Solve}_{\mathsf{env}}(\mathcal{A}_2)$
 4: $\mathcal{D}_2' = \textsc{Restrict}(\mathcal{D}_2, S_2)$, say the sink state is $\bot_2$
 5: $\mathcal{D} = \textsc{Product}(\mathcal{D}_1, \mathcal{D}_2')$
 6: $(R, f_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathrm{reach}((T_1 \times S_2) \cup (Q_1 \times \{\bot_2\}))$
 7: if $\iota \notin R$ return "Unrealisable" endif
 8: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}, f_{\mathsf{ag}})$
```

9 Safety Tasks, Safety Env Specs

Algorithm 5 handles the case that Task is of the form $\forall\varphi_1$ and $\mathsf{Env} = \forall\varphi_2$, where φ_1, φ_2 are LTL$_f$ formulas.

Intuitively, the algorithm proceeds as follows. First, it computes the corresponding DA for $\forall\varphi_2$ and solves the safety game for the environment over it. The resulting winning area represents the set of states, from which the environment has a strategy to enforce the environment specification $\mathcal{L}(\forall\varphi_2)$. It is worth noting that restricting the DA to considering only such winning area, in fact, captures all the environment strategies that enforce $\mathcal{L}(\forall\varphi_2)$ [7]. Based on the restriction, the algorithm solves the safety game over the product of the corresponding DA of $\forall\varphi_1$ and the remaining part of the DA for $\forall\varphi_2$.

Algorithm 5. $\mathsf{Task} = \forall\varphi_1, \mathsf{Env} = \forall\varphi_2$

```
Input: LTL_f formulas $\varphi_1, \varphi_2$
Output: agent strategy $\sigma_{\mathsf{ag}}$ that enforces $\forall\varphi_1$ under $\forall\varphi_2$
 1: $\mathcal{A}_1 = \textsc{ConvertDA}(\forall\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \mathrm{safe}(T_1))$
 2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \mathrm{safe}(T_2))$
 3: $(S_2, f_{\mathsf{env}}) = \textsc{Solve}_{\mathsf{env}}(\mathcal{A}_2)$
 4: $\mathcal{D}_2' = \textsc{Restrict}(\mathcal{D}_2, S_2)$, call the sink $\bot_2$
 5: $\mathcal{D} = \textsc{Product}(\mathcal{D}_1, \mathcal{D}_2')$
 6: $(S, f_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathrm{safe}((T_1 \times S_2) \cup (Q_1 \times \{\bot_2\})))$
 7: if $\iota \notin S$ return "Unrealisable" endif
 8: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}, f_{\mathsf{ag}})$
```

Theorem 7. *Algorithm 5 solves the synthesis under environment specifications problem with* $\mathsf{Task} = \forall\varphi_1, \mathsf{Env} = \forall\varphi_2$*, where the* φ_i *are* LTL$_f$ *formulas.*

10 Reachability and Safety Tasks, Safety Env Specs

Algorithm 6 handles the case that Task is of the form $\exists\varphi_1 \wedge \forall\varphi_2$ and $\mathsf{Env} = \forall\varphi_3$, where $\varphi_1, \varphi_2, \varphi_3$ are LTL$_f$ formulas. As mentioned in the previous section, a similar problem of this case that considers only finite safety of the agent was solved in [13] by reducing Task to reachability properties only. Instead, we provide here an approach to the synthesis problem considering infinite agent safety.

Intuitively, the algorithm proceeds as follows. Following the algorithms presented in the previous sections, it first computes all the environment strategies that can enforce $\mathsf{Env} = \varphi_3$, represented as a restriction of the DA for $\forall\varphi_3$. Then, based on restricting the game arena on these environment strategies, the algorithm solves the safety game for the agent over the product of the corresponding DA of $\forall\varphi_2$ and the restricted part of the DA for $\forall\varphi_3$. This step is able to capture all the agent strategies that can realize $\forall\varphi_2$ under environment specification $\forall\varphi_3$. Next, we represent all these agent strategies by restricting the product automaton to considering only the computed agent winning states, thus obtaining $\mathcal{D}'$. Finally, the algorithm solves the reachability game over the product of the corresponding DA of $\exists\varphi_1$ and $\mathcal{D}'$. In order to abstract the final strategy for the agent, it is necessary to combine the two agent strategies: one is from the safety game for enforcing $\forall\varphi_2$ under $\forall\varphi_3$, the other one is from the final reachability game for enforcing $\exists\varphi_1$ while not violating $\forall\varphi_2$ under $\forall\varphi_3$.

Algorithm 6. $\mathsf{Task} = \exists\varphi_1 \wedge \forall\varphi_2, \mathsf{Env} = \forall\varphi_3$

```
Input: LTL_f formulas $\varphi_1, \varphi_2, \varphi_3$
Output: agent strategy $\sigma_{\mathsf{ag}}$ that enforces $\exists\varphi_1 \wedge \forall\varphi_2$ under $\forall\varphi_3$
 1: $\mathcal{A}_1 = \textsc{ConvertDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \mathrm{reach}(T_1))$
 2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \mathrm{safe}(T_2))$
 3: $\mathcal{A}_3 = \textsc{ConvertDA}(\forall\varphi_3)$, say $\mathcal{A}_3 = (\mathcal{D}_3, \mathrm{safe}(T_3))$
 4: $(S_3, f_{\mathsf{env}}) = \textsc{Solve}_{\mathsf{env}}(\mathcal{A}_3)$
 5: $\mathcal{D}_3' = \textsc{Restrict}(\mathcal{D}_3, S_3)$, call the sink $\bot_3$
 6: $\mathcal{D} = \textsc{Product}(\mathcal{D}_2, \mathcal{D}_3')$
 7: $(S_2, f^s_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathrm{safe}((T_2 \times S_3) \cup (Q_2 \times \{\bot_3\})))$
 8: $\mathcal{D}' = \textsc{Restrict}(\mathcal{D}, S_2)$, call the sink $\bot_2$
 9: $\mathcal{C} = \textsc{Product}(\mathcal{D}_1, \mathcal{D}')$
10: Let $f^{s'}_{\mathsf{ag}} : Q_1 \times (S_2 \cup \{\bot_2\}) \to 2^{\mathcal{Y}}$ map $(q_1, q_2)$ to $f^s_{\mathsf{ag}}(q_2)$ if $q_2 \in S_2$, and is arbitrary
    otherwise. $\{f^{s'}_{\mathsf{ag}}$ lifts $f^s_{\mathsf{ag}}$ to $\mathcal{C}\}$
11: $(R, f^r_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{C}, \mathrm{reach}((T_1 \times S_2 \times S_3) \cup (Q_1 \times (\eta(S_2) \cup \{\bot_2\}) \times \{\bot_3\}))$
    $\{\ \eta : Q_2 \times Q_3 \to Q_2$ is the projection onto $Q_2$, i.e., $(q_2, q_3) \mapsto q_2\}$
12: if $\iota \notin R$ return "Unrealisable" endif
13: Let $f_{\mathsf{ag}} : Q_1 \times (S_2 \cup \{\bot_2\}) \to 2^{\mathcal{Y}}$ on $\mathcal{C}$ map $q$ to $f^r_{\mathsf{ag}}(q)$ if $q \in R$, and to $f^{s'}_{\mathsf{ag}}(q)$
    otherwise. $\{f_{\mathsf{ag}}$ does $f^r_{\mathsf{ag}}$ on $R$, and $f^{s'}_{\mathsf{ag}}$ otherwise.$\}$
14: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{C}, f_{\mathsf{ag}})$
```

Theorem 8. *Algorithm 6 solves synthesis under environment specifications problem with* $\mathsf{Task} = \exists\varphi_1 \wedge \forall\varphi_2, \mathsf{Env} = \forall\varphi_3$*, where the* φ_i *are* LTL$_f$ *formulas.*

11 Reachability and Safety Tasks and Env Specs

Algorithm 7 handles the case that $\mathsf{Env} = \forall\varphi_1 \wedge \exists\varphi_2$ and $\mathsf{Task} = \exists\varphi_3 \wedge \forall\varphi_4$ by solving synthesis for the formula $\mathsf{Env} \rightarrow \mathsf{Task}$ [2], i.e., for $(\exists\neg\varphi_1 \vee \forall\neg\varphi_2) \vee (\exists\varphi_3 \wedge \forall\varphi_4)$. Note that, from the general case, we get all cases involving reachability environment specifications by suitably setting φ_1, φ_2 or φ_4 to *true*. We remark that for the case $\varphi_4 = \mathit{true}$ in which the safety and reachability specifications are presented in the safety-fragment and co-safety fragment of LTL is solved in [10].

We first define two constructions that will be used in the algorithm. Given a transition system $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ and a set of states $T \subseteq Q$, define FLAGGED$(\mathcal{D}, T)$ to be the transition system that, intuitively, records whether a state in T has been seen so far. Formally, FLAGGED$(\mathcal{D}, T)$ returns the transition system $D^f = (\Sigma, Q^f, \iota^f, \delta^f)$ defined as follows: 1. $Q^f = Q \times \{yes, no\}$. 2. $\iota^f = (\iota, b)$, where $b = no$ if $\iota \notin T$, and $b = yes$ if $\iota \in T$. 3. $\delta^f((q, b), z) = (q', b')$ if $\delta(q, z) = q'$ and one of the following conditions holds: (i) $b = b' = yes$, (ii) $b = b' = no, q' \notin T$, (iii) $b = no, b' = yes, q' \in T$. Given a transition system $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ and disjoint subsets V_0, V_1 of Q, define RESTRICTIONWITHSINKS$(\mathcal{D}, V_0, V_1)$ to be the transition system on state set V_0 that, intuitively, behaves like $\mathcal{D}$ on V_0, transitions from V_0 to V_1 are redirected to a new sink state $\bot$, and transitions from V_0 to $Q \setminus (V_0 \cup V_1)$ are redirected to a new sink state $\top$. Formally, RESTRICTIONWITHSINKS$(\mathcal{D}, V_0, V_1)$ is the transition system $(\Sigma, \hat{Q}, \hat{\iota}, \hat{\delta})$ defined as follows: 1. $\hat{Q} = V_0 \cup \{\top, \bot\}$. 2. $\hat{\iota} = \iota$. 3. $\hat{\delta}(q, z) = \delta(q, z)$ if $\delta(q, z) \in V_0$. Otherwise, define $\hat{\delta}(q, z) = \bot$ if $\delta(q, z) \in V_1$, and $\hat{\delta}(q, z) = \top$ if $\delta(q, z) \in Q \setminus (V_0 \cup V_1)$.

Intuitively, at Line 10, S_2 will form part of the agent's winning region since from here safe(T_2) can be ensured. At Line 12, R_3 will also form part of the agent's winning region since from R_3 in $\mathcal{D}'$ reach$(T_3)\cap$safe(T_4) can be ensured. In the following steps, we identify remaining ways that the agent can win, intuitively by maintaining $T_2 \cap T_4$ either forever (in which case safe(T_2) is ensured), or before the state leaves $T_2 \cap T_4$ either (i) it is in S_2 or R_3 (in which case we proceed as before), or otherwise (ii) it is in S_4 (but not in S_2 nor in R_3) and has already seen T_3 (in which case reach$(T_3) \cap$ safe(T_4) can be ensured).

At the end of the algorithm, we combine the four strategies $f^1_{\mathsf{ag}}, f^2_{\mathsf{ag}}, f^3_{\mathsf{ag}}$ and f^4_{ag} through procedure COMBINE$(\mathcal{D}^f,\ f^1_{\mathsf{ag}}, f^2_{\mathsf{ag}}, f^3_{\mathsf{ag}}, f^4_{\mathsf{ag}}, R_1, S_2, R_3, E)$ to obtain the final strategy $f_{\mathsf{ag}} : (Q^f)^+ \rightarrow 2^{\mathcal{Y}}$ as follows. For every history $h \in (Q^f)^+$, if the history ever enters R_1 then follow f^1_{ag}, ensuring reach(T_1), otherwise, writing q for the start state of h: 1. if $q \in S_2$ then use f^2_{ag}, which ensures safe(T_2); 2. if $q \in R_3$ then use f^3_{ag} until T_3 is reached and thereafter use f^4_{ag}, which ensures safe$(T_4) \cap$ reach(T_3); 3. if $q \in E$ then use f^e_{ag} while the states are in E, ensuring safe(T_2) if play stays in E; if ever, let q' be the first state in the history that is not in E; by construction, this corresponds to $\top$ in $\mathcal{D}^f$ and thus is (i) in S_2 or (ii) in R_3, and so proceed as before, or else (iii) in $(S_4 \setminus T_2) \setminus (R_3 \cup S_2)$ (which can be simplified to $S_4 \setminus (R_3 \cup T_2)$) with flag value *yes* in which case switch to strategy f^4_{ag}. Intuitively, case (i) ensures safe(T_2), and cases (ii) and (iii) each ensure safe$(T_4)\cap$reach(T_3); 4. and if none of these, then make an arbitrary move. Note that in spite of being a function of the whole history, f_{ag} can be represented

Algorithm 7. $\mathsf{Task} = \exists\varphi_3 \wedge \forall\varphi_4$, $\mathsf{Env} = \forall\varphi_1 \wedge \exists\varphi_2$

```
Input: LTL_f formulas $\varphi_1, \varphi_2, \varphi_3, \varphi_4$
Output: agent strategy $\sigma_{\mathsf{ag}}$ that enforces $\exists\varphi_3 \wedge \forall\varphi_4$ under $\forall\varphi_1 \wedge \exists\varphi_2$
 1: $\mathcal{A}_1 = \textsc{ConvertDA}(\exists\neg\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \text{reach}(B_1))$
 2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\neg\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \text{safe}(B_2))$
 3: $\mathcal{A}_3 = \textsc{ConvertDA}(\exists\varphi_3)$, say $\mathcal{A}_3 = (\mathcal{D}_3, \text{reach}(B_3))$
 4: $\mathcal{A}_4 = \textsc{ConvertDA}(\forall\varphi_4)$, say $\mathcal{A}_4 = (\mathcal{D}_4, \text{safe}(B_4))$
 5: $\mathcal{D}_p = \textsc{Product}(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4)$
 6: Let $Q_p$ be the state set of $\mathcal{D}_p$, and $T_i$ the lift of $B_i$ to $Q_p$ (for $i \leq 4$)
 7: $(R_1, f^1_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}_p, \text{reach}(T_1))$
 8: $\mathcal{D}'_p = \textsc{Restrict}(\mathcal{D}_p, Q \setminus R_1)$
 9: $\mathcal{D}^f = \textsc{Flagged}(\mathcal{D}'_p, T_3)$
10: $(S_2, f^2_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}^f, \text{safe}(T_2))$
11: $(S_4, f^4_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}^f, \text{safe}(T_4))$
12: $(R_3, f^3_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\textsc{Restrict}(\mathcal{D}^f, S_4), \text{reach}(T_3))$
13: $V_0 = (Q^f \setminus (S_2 \cup S_4)) \cup ((S_4 \cap T_2) \setminus (R_3 \cup S_2))$
14: $V_1$ is all states in $(S_4 \setminus T_2) \setminus (R_3 \cup S_2)$ whose flag is set to *no*
15: $\hat{\mathcal{D}} = \textsc{RestrictionWithSinks}(\mathcal{D}^f, V_0, V_1)$
16: $(E, f^e_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\hat{\mathcal{D}}, \text{safe}((T_2 \cap T_4) \cup \{\top\}))$
17: $W_{\mathsf{ag}} = S_2 \cup R_3 \cup E$ {Note that $W_{\mathsf{ag}} \subseteq Q^f \cup \{\top\}$}
18: if $\iota \notin W_{\mathsf{ag}}$ return "Unrealisable" endif
19: $f_{\mathsf{ag}} = \textsc{Combine}(\mathcal{D}^f, f^1_{\mathsf{ag}}, f^2_{\mathsf{ag}}, f^3_{\mathsf{ag}}, f^4_{\mathsf{ag}}, R_1, S_2, R_3, E)$ {See the definition below.}
20: return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}^f, f_{\mathsf{ag}})$
```

by a finite-state transducer. So in the Algorithm 7, as before, with a little abuse of notation we write directly $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}^f, f_{\mathsf{ag}})$, to mean that we return its representation as a transducer.

Theorem 9. *Algorithm 7 solves the synthesis under environment specifications problem with* $\mathsf{Task} = \exists\varphi_3 \wedge \forall\varphi_4$ *and* $\mathsf{Env} = \forall\varphi_1 \wedge \exists\varphi_2$*, where* φ_i *are* LTL$_f$ *formulas.*

Comparison to Algorithms 1–6. Note that Algorithm 7 can solve the other six variants by suitably instantiating some of $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ to *true*. Nevertheless, Algorithm 7 is much more sophisticated than Algorithms 1–6. Hence, in this paper, we present the algorithms deductively, starting with simpler variants and moving to the most difficult. Furthermore, instantiating Algorithm 7 does not always give the same algorithms as Algorithms 1–6. For instance, Algorithm 1 for the synthesis problem of $\mathsf{Task} = \exists\varphi$ (no environment specification) can be obtained from Algorithm 7 by setting $\varphi_1, \varphi_2, \varphi_4$ to *true*, but we cannot get Algorithm 4 for the synthesis problem of $\mathsf{Env} = \forall\varphi$ and $\mathsf{Task} = \exists\psi$ in this way. This is because Algorithm 7 solves the synthesis problem by reducing to $\mathsf{Env} \rightarrow \mathsf{Task}$ [2], but Algorithm 4 directly disregards all environment strategies that cannot enforce Env by first solving a safety game for the environment on Env and removing all the states that do not belong to the environment winning region to get a smaller game arena, hence obtaining optimal complexity. Analogously, in Algorithm 3 for the synthesis problem of $\mathsf{Env} = $ *true* and $\mathsf{Task} = \exists\varphi_1 \wedge$

$\forall\varphi_2$, we also first disregard all the agent strategies that are not able to enforce $\forall\varphi_2$, obtaining a smaller game arena for subsequent computations, hence getting an optimal complexity in practice compared to constructing the game arena considering the complete state space from the DA of $\forall\varphi_2$.

12 Conclusion

In this paper, we have studied the use of reachability and safety properties based on LTL$_f$ for both agent tasks and environment specifications. As mentioned in the introduction, though we have specifically focused on LTL$_f$, all algorithms presented here can be readily applied to other temporal logics on finite traces, such as Linear Dynamic Logics on finite traces (LDL$_f$), which is more expressive than LTL$_f$ [16], and Pure-Past LTL [11], as long as there exists a technique to associate formulas to equivalent DFAs.

It is worth noting that all the cases studied here are specific Boolean combinations of $\exists\varphi$. It is of interest to indeed devise algorithms to handle arbitrary Boolean combinations. Indeed, considering that LTL$_f$ is expressively equivalent to pure-past LTL, an arbitrary Boolean combination of $\exists\varphi$ would correspond to a precise class of LTL properties in Manna & Pnueli's Temporal Hierarchy [23]: the so-called *obligation* properties. We leave this interesting research direction for future work.

Another direction is to consider best-effort synthesis under assumptions for Boolean combinations of $\exists\varphi$, instead of (ordinary) synthesis under assumptions, in order to handle ignorance the agent has about the environment [3,5,6,8,18].

References

1. Aminof, B., De Giacomo, G., Murano, A., Rubin, S.: Planning and synthesis under assumptions. CoRR (2018)
2. Aminof, B., De Giacomo, G., Murano, A., Rubin, S.: Planning under LTL environment specifications. In: ICAPS, pp. 31–39 (2019)
3. Aminof, B., De Giacomo, G., Lomuscio, A., Murano, A., Rubin, S.: Synthesizing best-effort strategies under multiple environment specifications. In: KR, pp. 42–51 (2021)
4. Aminof, B., De Giacomo, G., Murano, A., Rubin, S.: Synthesis under assumptions. In: KR, pp. 615–616. AAAI Press (2018)
5. Aminof, B., De Giacomo, G., Rubin, S.: Best-effort synthesis: doing your best is not harder than giving up. In: IJCAI, pp. 1766–1772. ijcai.org (2021)
6. Aminof, B., De Giacomo, G., Rubin, S., Zuleger, F.: Stochastic best-effort strategies for Borel goals. In: LICS, pp. 1–13 (2023)
7. Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. RAIRO Theor. Inform. Appl. **36**(3), 261–275 (2002)
8. Berwanger, D.: Admissibility in infinite games. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 188–199. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70918-3_17
9. Camacho, A., Bienvenu, M., McIlraith, S.A.: Finite LTL synthesis with environment assumptions and quality measures. In: KR, pp. 454–463 (2018)

10. Camacho, A., Bienvenu, M., McIlraith, S.A.: Towards a unified view of AI planning and reactive synthesis. In: ICAPS, pp. 58–67 (2019)
11. De Giacomo, G., Di Stasio, A., Fuggitti, F., Rubin, S.: Pure-past linear temporal and dynamic logic on finite traces. In: IJCAI, pp. 4959–4965 (2020)
12. De Giacomo, G., Di Stasio, A., Perelli, G., Zhu, S.: Synthesis with mandatory stop actions. In: KR, pp. 237–246 (2021)
13. De Giacomo, G., Di Stasio, A., Tabajara, L.M., Vardi, M.Y., Zhu, S.: Finite-trace and generalized-reactivity specifications in temporal synthesis. In: IJCAI, pp. 1852–1858 (2021)
14. De Giacomo, G., Di Stasio, A., Vardi, M.Y., Zhu, S.: Two-stage technique for LTL_f synthesis under LTL assumptions. In: KR, pp. 304–314 (2020)
15. De Giacomo, G., Rubin, S.: Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In: IJCAI, pp. 4729–4735 (2018)
16. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp. 854–860 (2013)
17. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: IJCAI, pp. 1558–1564 (2015)
18. Faella, M.: Admissible strategies in infinite games over graphs. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 307–318. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03816-7_27
19. Fijalkow, N., et al.: Games on graphs (2023)
20. Geffner, H., Bonet, B.: A Coincise Introduction to Models and Methods for Automated Planning. Morgan & Claypool (2013)
21. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36387-4
22. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods Syst. Des. **19**(3), 291–314 (2001)
23. Manna, Z., Pnueli, A.: A hierarchy of temporal properties. In: PODC, pp. 377–410 (1990)
24. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190 (1989)
25. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959)
26. Tabajara, L.M., Vardi, M.Y.: LTLF synthesis under partial observability: from theory to practice. In: GandALF. EPTCS, vol. 326, pp. 1–17 (2020)
27. Tabajara, L.M., Vardi, M.Y.: Partitioning techniques in LTLF synthesis. In: IJCAI, pp. 5599–5606 (2019)
28. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 396–411. Springer, Heidelberg (2005). https://doi.org/10.1007/11591191_28
29. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using LTL. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 83–98. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_10
30. Zhu, S., De Giacomo, G., Pu, G., Vardi, M.Y.: LTL_f synthesis with fairness and stability assumptions. In: AAAI, pp. 3088–3095 (2020)
31. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTL_f synthesis. In: IJCAI, pp. 1362–1369 (2017)
32. Zhu, S., Tabajara, L.M., Pu, G., Vardi, M.Y.: On the power of automata minimization in temporal synthesis. In: GandALF. EPTCS, vol. 346, pp. 117–134 (2021)