Information and Computation 303 (2025) 105255

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

LTL_f synthesis under environment specifications for reachability and safety properties

Benjamin Aminof [a], Giuseppe De Giacomo [a,b], Antonio Di Stasio [e,*], Hugo Francon [d], Sasha Rubin [c], Shufang Zhu [f]

[a] Sapienza University of Rome, Rome, Italy
[b] University of Oxford, Oxford, UK
[c] University of Sydney, Sydney, Australia
[d] ENS Rennes, France
[e] City St George's, University of London, London, UK
[f] University of Liverpool, Liverpool, UK

ARTICLE INFO

Article history:
Received 6 February 2024
Received in revised form 30 October 2024
Accepted 22 November 2024
Available online 28 November 2024

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 [4,2,10,28]. This problem is tightly related to planning in adversarial nondeterministic domains [22], as discussed, e.g., in [11,17].

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 [18,19], 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 [18], and Pure-Past LTL, which has the same expressiveness as LTL but evaluates a trace backward from the current instant [13].

Linear temporal logics on finite traces provide a nice embodiment of the notable triangle among Logics, Automata, and Games [23]. These logics are full-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 ob-

* Corresponding author.
E-mail addresses: benj@forsyte.at (B. Aminof), giuseppe.degiacomo@cs.ox.ac.uk (G. De Giacomo), antonio.di-stasio@city.ac.uk (A. Di Stasio), hugo.francon@ens-rennes.fr (H. Francon), sasha.rubin@sydney.edu.au (S. Rubin), shufang.zhu@liverpool.ac.uk (S. Zhu).

https://doi.org/10.1016/j.ic.2024.105255

jective 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 [30,32,36].

In this paper, however, we consider two kind of LTL_f specifications:

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

The first kind of specifications denote 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 [19]. 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 [1,15]. 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 Safety-Progress LTL Hierarchy [12]. Specifically, $\exists\varphi$ captures the *guarantee properties* and $\forall\varphi$ captures the *safety properties* (in [25], 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 [19]). 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 [19], 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 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 [31,32,36]. Moreover, in several notable cases, e.g., in all DECLARE patterns [33], 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 $\pi = \pi_0\pi_1 \ldots \pi_n$, 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 (LTL). LTL is one of the most popular logics for temporal properties [27]. Given a set of propositions *Prop*, the formulas of LTL are generated as follows:

$$\varphi ::= a \mid (\varphi \wedge \varphi) \mid (\neg\varphi) \mid (\bigcirc\varphi) \mid (\varphi\, \mathcal{U}\, \varphi)$$

where $a \in Prop$, $\bigcirc$ (*next*) and $\mathcal{U}$ (*until*) are temporal operators. We use common abbreviations, so we have *eventually* as $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ and *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

LTL formulas are interpreted over infinite traces $\pi \in (2^{Prop})^\omega$. A *trace* $\pi = \pi_0, \pi_1, \ldots$ is a sequence of propositional interpretations (sets), where for every $i \geq 0$, $\pi_i \in 2^{Prop}$ is the i-th interpretation of π. Intuitively, π_i is interpreted as the set of propositions that are *true* at instant i. Given π, we define when an LTL formula φ *holds* at position i, written as $\pi, i \models \varphi$, inductively on the structure of φ, as:

- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in Prop$);
- $\pi, i \models \neg\varphi$ 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 $\pi, i + 1 \models \varphi$;
- $\pi, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff there exists $j \geq i$ such that $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

[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.

[2] For pure-past LTL, obtaining the DFA from a pure-past LTL formula is single exponential [13], and indeed the problems and all our algorithms become EXPTIME-complete.

We say π *satisfies* φ, written as $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

Linear-time Temporal Logic on finite traces (LTL$_f$). LTL$_f$ is a variant of Linear-time temporal logic (LTL) interpreted over *finite*, instead of infinite traces [18]. The syntax of LTL$_f$ is exactly the same as for LTL. For a finite trace $\pi \in (2^{Prop})^+$, an LTL$_f$ formula φ, and a position i ($0 \le i < |\pi|$), define $\pi, i \models \varphi$ (read "φ *holds* at position i") by induction as for LTL expect that for the temporal operators we have:

- $\pi, i \models \bigcirc\varphi$ iff $i < |\pi| - 1$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff for some j ($i \le j < |\pi|$), $\pi, j \models \varphi_2$, and for all k ($i \le k < j$), $\pi, k \models \varphi_1$.

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 $w = w_0 w_1 w_2 \ldots \in \Sigma^\omega$ 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 $0 \le i$. 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 $0 \le i < n+1$. 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. *[19] 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*, (also called *guarantee* [25] or *co-safety* [24]), 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 the 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 $T \subseteq Q$:

- $\text{reach}(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \exists k \ge 0 : q_k \in T\}$. In this case, we call $\mathcal{A}$ a *reachability automaton*.
- $\text{safe}(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \forall k \ge 0 : q_k \in T\}$. 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) [18,24,29].

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 $Y_{i+1} = \sigma_{\mathsf{ag}}(X_0X_1\ldots X_i)$ for every $i \geq 0$, and it *follows an environment strategy* σ_{env} if $X_i = \sigma_{\mathsf{env}}(Y_0Y_1\ldots Y_i)$ for all $i \geq 0$. We denote the unique infinite sequence (play) that follows σ_{ag} and σ_{env} as $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathtt{env}})$. 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 $\sigma_{\mathtt{env}} \rhd \mathsf{Env}$ we have that $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathtt{env}}) \models \mathsf{Task}$. Note that if $\mathsf{Env} = true$ 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 $\mathsf{Env} \to \mathsf{Task}$ is realizable. Moreover, to solve the synthesis problem for Task under Env, it is enough to return a strategy that enforces $\mathsf{Env} \to \mathsf{Task}$.

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 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., [14,16,35,34]. 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 [15,16,34,35]. 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, $\mathsf{Env} = true$). 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.*

Proof. *Upper Bound.* The solutions provided in Algorithms 1-7 rely on constructing the corresponding DFAs of Task and Env, cartesian products of a bounded number of DFAs, as well as solving reachability and safety games. The corresponding DFAs of Task and Env can be constructed in 2EXPTIME in the size of Task and Env, and solving reachability and safety games can be accomplished in polynomial time in the size of the DFA. In summary, the upper bound for Algorithms 1-7 is 2EXPTIME.

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

[4] In [10] 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.

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

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

Lower Bound. The following problem is known (see [19]) to be 2EXPTIME-hard (*): given an LTL_f formula φ decide if there exists an agent strategy that enforces $\exists\varphi$. Thus, e.g., we get hardness for synthesis for Task $= \exists\varphi_1 \wedge \forall\varphi_2$ under Env $=$ *true* (just let $\varphi_2 = true$). To handle the missing cases, we will show that the following problem is 2EXPTIME-hard (+): given an LTL_f formula φ decide if there is an agent strategy that enforces $\forall\varphi$.

First note that if L is in 2EXPTIME, then so is L^c, the complement of L (this is because 2EXPTIME is a deterministic complexity class). Thus, if L is 2EXPTIME-hard, then so is L^c (indeed, if X is in 2EXPTIME, then so is X^c, which reduces to L by assumption, and thus X reduces to L^c). Thus, by (*), we have that the following problem is 2EXPTIME-hard: given an LTL_f formula φ decide if there does *not* exist an agent strategy that enforces $\exists\varphi$. By determinacy of the corresponding game [26], there does not exist an agent strategy that enforces $\exists\varphi$ if and only if there exists an environment strategy that enforces $\forall\neg\varphi$. Thus, we have that the following problem is 2EXPTIME-hard (**): given an LTL_f formula φ, decide if there is an environment strategy that enforces $\forall\varphi$. To finish the proof, we reduce (**) to (+). Take an instance φ of (**); suppose $AP = X \cup Y$ where the agent controls the variables in Y and the environment controls the variables in X. Define $AP' = X \cup Y \cup Y'$ where $Y' = \{y' : y \in Y\}$ (i.e., Y' is a copy of Y, call its elements 'primed'), and let the agent control the variables in X and the environment control the variables in Y. Define

$$\varphi' = (\bullet \mathit{false}) \vee ((\Box \wedge_{y \in Y} (y' \to \bullet y)) \to (\circ\varphi))$$

It is not hard to see that the environment can enforce $\forall\varphi$ iff the agent can enforce $\forall\varphi'$. Intuitively, this is because the new objective φ' checks if φ holds from the second step onwards as long as the environment correctly copies and unprimes its moves from each previous time-step. $\square$

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 $\mathcal{A}_{\exists\varphi} = (2^{\mathcal{X}\cup\mathcal{Y}}, Q, \iota, \delta, \text{reach}(F))$. 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 $\mathcal{A}_{\forall\varphi} = (2^{\mathcal{X}\cup\mathcal{Y}}, Q, \iota, \delta, \text{safe}(F \cup \{\iota\}))$.

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 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 $\mathcal{L}(\text{ConvertDA}(\psi))$ are equal.*

For formulas of the form $\forall\varphi$ we will suppress the initial state in the objective and so ConvertDA($\forall\varphi$) will be written $(\mathcal{D}_{\forall\varphi}, \text{safe}(T))$, 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 $\mathcal{A} = (\mathcal{D}, \alpha)$ 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). We now show how to solve reachability and safety games.

Preimage. In order to compute the set of states from which the agent can force a visit to a given set S in one step, we need to define the *controllable/uncontrollable preimage*, which is the main step for solving reachability and safety games. We define the controllable preimage $Pre_{\mathsf{ag}}(E)$ of a set $E \subseteq Q$ as the set of states, from which there exists an agent action $Y \in 2^{\mathcal{Y}}$ such that for all environment response $X \in 2^{\mathcal{X}}$, the corresponding successor state $\delta(q, X \cup Y)$ is in E. Analogously, $Pre_a(E)$ denotes the set of states, from which for all $Y \in 2^{\mathcal{Y}}$, there exists $X \in 2^{\mathcal{X}}$ such that $\delta(q, X \cup Y)$ is in E. Formally,

$$Pre_{\mathsf{ag}}(E) = \{q \in Q \mid \exists Y \forall X . \delta(q, X \cup Y) \in E\}$$
$$Pre_{\mathsf{env}}(E) = \{q \in Q \mid \forall Y \exists X . \delta(q, X \cup Y) \in E\}$$

Reachability Games. Given a game arena $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ and a reachability condition (reach, T), $\textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathrm{reach}, T)$ solves the reachability game over $\mathcal{D}$ for the agent by computing the least fixed-point as follows.

$$\mathcal{Z}_0(\mathcal{D}) = T$$
$$\mathcal{Z}_{i+1}(\mathcal{D}) = \mathcal{Z}_i(\mathcal{D}) \cup Pre_{\mathsf{ag}}(\mathcal{Z}_i(\mathcal{D}))$$

The computation reaches a fixed point when $\mathcal{Z}_i = \mathcal{Z}_{i+1}$, hence $\mathcal{Z} = \mathcal{Z}_{i+1}$ collects all the winning states for the protagonist. If $\iota \in \mathcal{Z}$, we compute a positional strategy as follows. Define $f_{\mathsf{ag}} : Q \to 2^{\mathcal{Y}}$ such that for every $q \in \mathcal{Z} \setminus \mathcal{Z}_0$, if q is added for the first time in the $(i+1)$-*th* iteration of the fixpoint computation (i.e., $q \in \mathcal{Z}_{i+1} \setminus \mathcal{Z}_i$), define $f_{\mathsf{ag}}(q) = Y$, where Y is any element such that $\forall X \in 2^{\mathcal{X}} . \delta(q, Y \cup X) \in \mathcal{Z}_i$ (If there is more than one such Y, arbitrarily choose one); if $q \in \mathcal{Z}_0$ or $q \notin \mathcal{Z}$, define $f_{\mathsf{ag}}(q)$ arbitrarily.

Solving the reachability game considering the environment as the protagonist can be defined analogously, in particular replacing $Pre_{\mathsf{ag}}(\mathcal{Z}_i(\mathcal{D}))$ by $Pre_{\mathsf{env}}(\mathcal{Z}_i(\mathcal{D}))$. In this case, the positional strategy $f_{\mathsf{env}} : Q \times 2^Y \to 2^{\mathcal{X}}$ is such that for every $q \in \mathcal{Z}_{i+1} \setminus \mathcal{Z}_i$ and $Y \in 2^{\mathcal{Y}}$, $f_{\mathsf{env}}(q, Y) = X$, where X holds that $\delta(q, Y \cup X) \in \mathcal{Z}_i$ (Similarly, if there is more than one such X, arbitrarily choose one). Moreover, if $q \in \mathcal{Z}_0$ or $q \notin \mathcal{Z}$, $f_{\mathsf{env}}(q, Y) = X$, for $Y \in 2^{\mathcal{Y}}$ and $X \in 2^{\mathcal{X}}$.

Safety Games. Given a game arena $\mathcal{D} = (\Sigma, Q, \iota, \delta)$ and a safety condition (safe, T), $\textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathrm{safe}, T)$ solves the safety game over $\mathcal{D}$, considering the agent as the protagonist, by computing the greatest fixed point as follows:

[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.

$$\mathcal{Z}_0(\mathcal{D}) = T$$
$$\mathcal{Z}_{i+1}(\mathcal{D}) = \mathcal{Z}_i(\mathcal{D}) \cap Pre_{\mathsf{ag}}(\mathcal{Z}_i(\mathcal{D}))$$

Solving the safety game considering the environment as the protagonist $\textsc{Solve}_{\mathsf{env}}$ can be defined analogously by replacing $Pre_{\mathsf{ag}}(\mathcal{Z}_i(\mathcal{A}))$ with $Pre_{\mathsf{env}}(\mathcal{Z}_i(\mathcal{A}))$.

The computation reaches fixpoint when $\mathcal{Z}_i = \mathcal{Z}_{i+1}$, hence $\mathcal{Z} = \mathcal{Z}_{i+1}$ collects all the winning states for the protagonist. If $\iota \in \mathcal{Z}$, we compute a positional strategy as follows. Considering the agent as the protagonist, we have $f_{\mathsf{ag}} : Q \to 2^{\mathcal{Y}}$ such that for every $q \in \mathcal{Z}$, $f_{\mathsf{ag}}(q) = Y$, where Y holds that $\forall X \in 2^{\mathcal{X}}.\delta(q, Y \cup X) \in \mathcal{Z}$ (If there are more than one such Y, arbitrarily choose one). For the case of the environment being the protagonist, the positional strategy $f_{\mathsf{env}} : Q \times 2^{Y} \to 2^{\mathcal{X}}$ is such that for every $q \in \mathcal{Z}$ and $Y \in 2^{\mathcal{Y}}$, $f_{\mathsf{env}}(q, Y) = X$, where X holds that $\delta(q, Y \cup X) \in \mathcal{Z}$ (If there are more than one such X, arbitrarily choose one).

Lemma 3. *[21] The procedure $\textsc{Solve}_p(\mathcal{D}, \alpha)$, where α is a reachability/safety winning condition, returns the winning region W_p and a function f_p such that the protagonist strategy $\textsc{Strategy}(f_p)$ wins the game $(\mathcal{D}, \alpha)$.*

Remark 2. Given a transition system $\mathcal{D}$ with state set Q, and a set $T \subseteq Q$, consider the safety DA $\mathcal{A} = (\mathcal{D}, \mathsf{safe}(T))$. Let Win_p be the winning region of the protagonist. Consider the restricted transition system $\mathcal{D}' := \textsc{Restrict}(\mathcal{D}, \mathsf{Win}_p)$. Note that $\mathcal{A}' = (\mathcal{D}', \mathsf{safe}(T))$ is a well-defined DA (i.e., reaching the *sink* violates the safety condition). A strategy for the protagonist is winning in $\mathcal{A}$ iff it is winning in $\mathcal{A}'$. Intuitively, this is because winning strategies for a safety condition cannot leave the winning region, and this is the only requirement for them to be winning. Thus, intuitively, the restriction $\mathcal{D}'$ represents all the strategies that enforce $\mathcal{L}(\mathcal{D}, \mathsf{safe}, T)$ [8].

Product of Transition Systems. Let $\mathcal{D}_i$ $(1 \le i \le k)$ 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 $q'_i = \delta_i(q_i, z)$ $(1 \le i \le k)$. Also, the *lift* of a set $F_i \subseteq Q_i$ to $\mathcal{D}$ is the set $\{(q_1, \cdots, q_k) : q_i \in F_i\} \subseteq Q$.

Lemma 4. *A trace in $\mathcal{D} = \textsc{Product}(\mathcal{D}_1, \cdots, \mathcal{D}_k)$ follows the lift of a strategy $f_{\mathsf{ag}}/f_{\mathsf{env}}$ on $\mathcal{D}_i$ to $\mathcal{D}$ if and only if the trace follows the strategy $f_{\mathsf{ag}}/f_{\mathsf{env}}$ on $\mathcal{D}$, for some $i \in \{1, \ldots, k\}$.*

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 $\perp$.[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 equivalent to solving synthesis for LTL_f specifications solved in [18]. We rephrase the problem using our notation, integrating it into our solution framework.

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

Input: LTL_f formula φ
Output: agent strategy σ_{ag} that enforces $\exists\varphi$
1: $\mathcal{A} = \textsc{ConvertDA}(\exists\varphi)$, say $\mathcal{A} = (\mathcal{D}_{\exists\varphi}, \mathsf{reach}(T))$
2: $(W, f_{\mathsf{ag}}) = \textsc{Solve}_{ag}(\mathcal{A})$
3: **if** $\iota \notin W$ **return** "Unrealisable" **endif**
4: **return** $\sigma_{\mathsf{ag}} = \textsc{Strategy}\ (\mathcal{D}_{\exists\varphi}, f_{\mathsf{ag}})$

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

[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.

Proof. By Lemma 1, we have (*): $L(\mathcal{D}, \text{reach}(T)) = L(\exists\varphi)$.

Suppose $\iota \in W$. We must show that the agent strategy $\sigma_{\text{ag}} = \text{STRATEGY}(f_{\text{ag}})$ returned by the algorithm enforces $\exists\varphi$. So, let π be a trace that follows σ_{ag} (we will show that $\pi \models \exists\varphi$). By Lemma 3, σ_{ag} wins the game $(\mathcal{D}, \text{reach}(T))$. By Definition 3, this means that $\pi \in L(\mathcal{D}, \text{reach}(T))$. By (*), $\pi \in L(\exists\varphi)$.

Conversely, suppose $\iota \notin W$. The algorithm returns "unrealisable", and we must show that there is no agent strategy that enforces $\exists\varphi$. Suppose towards a contradiction that there were, call it σ. By determinacy (Lemma 2), there is an environment strategy σ_{env} that wins the game $(\mathcal{D}, \text{reach}(T))$ for the antagonist. By definition this means that for every trace π that follows σ_{env}, the run of π on $\mathcal{D}$ satisfies that $\pi_i \notin T$ for every i, and thus $\pi \notin L(\mathcal{D}, \text{reach}(T))$, and so by (*) $\pi \not\models \exists\varphi$. But if σ enforces $\exists\varphi$ then $\pi = \text{play}(\sigma, \sigma_{\text{env}}) \models \exists\varphi$, a contradiction. □

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 $\text{Env} = \mathit{true}$. This problem can be approached using the result from [19] solving the reachability game over the corresponding DFA of $\neg\varphi$ for the environment. We present a direct solution to the problem using our framework.

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

Input: LTL_f formula φ
Output: agent strategy σ_{ag} that enforces $\forall\varphi$
1: $\mathcal{A} = \text{CONVERTDA}(\forall\varphi)$, say $\mathcal{A} = (\mathcal{D}_{\forall\varphi}, \text{safe}(T))$
2: $(S, f_{\text{ag}}) = \text{SOLVE}_{ag}(\mathcal{A})$
3: **if** $\iota \notin S$ **return** "Unrealisable" **endif**
4: **return** $\sigma_{\text{ag}} = \text{STRATEGY}(\mathcal{D}_{\forall\varphi}, f_{\text{ag}})$

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

Proof. By Lemma 1, we have (*): $L(\mathcal{D}, \text{safe}(T)) = L(\forall\varphi)$.

Suppose $\iota \in W$. We must show that the agent strategy $\sigma_{\text{ag}} = \text{STRATEGY}(f_{\text{ag}})$ returned by the algorithm enforces $\forall\varphi$. So, let π be a trace that follows σ_{ag} (we will show that $\pi \models \forall\varphi$). By Lemma 3, σ_{ag} wins the game $(\mathcal{D}, \text{safe}(T))$. By Definition 3, this means that $\pi \in L(\mathcal{D}, \text{reach}(T))$. By (*), $\pi \in L(\exists\varphi)$.

Conversely, suppose $\iota \notin W$. The algorithm returns "unrealisable", and we must show that there is no agent strategy that enforces $\forall\varphi$. Suppose towards a contradiction that there were, call it σ. By determinacy (Lemma 2), there is an environment strategy σ_{env} that wins the game $(\mathcal{D}, \text{reach}(T))$ for the antagonist. By definition this means that for every trace π that follows σ_{env}, the run of π on $\mathcal{D}$ satisfies that $\pi_i \notin T$ for some i, and thus $\pi \notin L(\mathcal{D}, \text{safe}(T))$, and so by (*) $\pi \not\models \forall\varphi$. But if σ enforces $\forall\varphi$ then $\pi = \text{play}(\sigma, \sigma_{\text{env}}) \models \forall\varphi$, a contradiction. □

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 $\text{Env} = \mathit{true}$. 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 $\forall\varphi_2$.

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

Input: LTL_f formulas φ_1 and φ_2
Output: agent strategy σ_{ag} that realizes $\exists\varphi_1$ and $\forall\varphi_2$
1: $\mathcal{A}_1 = \text{CONVERTDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_{\exists\varphi_1}, \text{reach}(T_1))$
2: $\mathcal{A}_2 = \text{CONVERTDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_{\forall\varphi_2}, \text{safe}(T_2))$
3: $(S_2, f_{\text{ag}}) = \text{SOLVE}_{ag}(\mathcal{A}_2)$
4: $\mathcal{D}'_{\forall\varphi_2} = \text{RESTRICT}(\mathcal{D}_{\forall\varphi_2}, S_2)$, say the sink state is $\bot_2$
5: $\mathcal{D} = \text{PRODUCT}(\mathcal{D}_{\exists\varphi_1}, \mathcal{D}'_{\forall\varphi_2})$
6: $(R, g_{\text{ag}}) = \text{SOLVE}_{ag}(\mathcal{D}, \text{reach}(T_1 \times S_2))$
7: **if** $\iota \notin R$ **return** "Unrealisable" **endif**
8: $h_{\text{ag}} = \text{COMBINE}(\mathcal{D}, R, g_{\text{ag}}, f_{\text{ag}})$
9: **return** $\sigma_{\text{ag}} = \text{STRATEGY}(\mathcal{D}, h_{\text{ag}})$

Lemma 5. *[7] Let* $\mathcal{A} = (\mathcal{D}, \text{safe}(T))$ *be a safety game and* S *the set of winning states for the agent in* $\mathcal{A}$. *Let* $\mathcal{B} = \mathit{Restricted}(\mathcal{A}, S)$ *be the restriction of* $\mathcal{A}$ *on* S. *Then, every winning strategy in* $\mathcal{A}$ *is a winning strategy in* $\mathcal{B}$, *and viceversa.*

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 $\textsc{Combine}(\mathcal{D}, R, g_{\mathsf{ag}}, f_{\mathsf{ag}})$ denote the final strategy $h_{\mathsf{ag}} : Q_1 \times (S_2 \cup \{\bot_2\}) \to 2^{\mathcal{Y}}$ 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* $\mathsf{Env} = true$, $\mathsf{Task} = \exists\varphi_1 \wedge \forall\varphi_2$, *where the* φ_i *are* LTL$_f$ *formulas.*

Proof. By Lemma 1, we have: $L(\mathcal{D}, \mathsf{reach}(T_1)) = L(\exists\varphi_1)$ and $L(\mathcal{D}, \mathsf{safe}(T_2)) = L(\forall\varphi_2)$.

We must show that the agent strategy $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}, h_{\mathsf{ag}})$ returned by the algorithm enforces $\exists\varphi_1 \wedge \forall\varphi_2$. Then, let a trace π that follows σ_{ag}. We must show that π satisfies $\exists\varphi_1 \wedge \exists\varphi_2$. First, we have that, by (*), $\pi \models \exists\varphi_1$ iff $\pi \in \mathcal{L}(\mathcal{D}_{\exists\varphi_1}, \mathsf{reach}(T_1))$ iff, by Lemma 4, $\pi \in \mathcal{L}(\mathcal{D}_{\exists\varphi_1} \times \mathcal{D}'_{\forall\varphi_2}, \mathsf{reach}(T_1 \times S_2))$. Note that, by construction, π follows g_{ag} until a state $q \in T_1 \times S_2$ is reached. Moreover, no *sink* state is visited. We need to show that $\pi \models \forall\varphi_2$. Since σ_{ag} is winning in $(\mathcal{D}_{\exists\varphi_1} \times \mathcal{D}_{\forall\varphi_2}, \mathsf{reach}(T_1 \times S_2))$, we have that, after reaching $q \in T_1 \times S_2$, π follows f''_{ag} and the projection of q on the components from Q_2 is in S_2. Therefore, $\pi \in \mathcal{L}(\mathcal{D}_{\exists\varphi_1} \times \mathcal{D}_{\forall\varphi_2}, \mathsf{safe}(T_1 \times S_2))$. We know that $\pi \models \forall\varphi_2$ by Lemma 1, $\pi \models \forall\varphi_2$ iff $\pi \in \mathcal{L}(\mathcal{D}_{\forall\varphi_2}, \mathsf{safe}, T_2)$ iff, by Lemma 5, $\pi \in \mathcal{L}(\mathcal{D}'_{\forall\varphi_2}, \mathsf{safe}(S_2))$ iff, by Lemma 4, $\pi \in \mathcal{L}(\mathcal{D}_{\exists\varphi_1} \times \mathcal{D}'_{\forall\varphi_2}, \mathsf{safe}(T_1 \times S_2))$. This concludes the proof. □

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 problem encompassing this case was solved in [15], 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. Here, we give the direct solution to this problem.

Intuitively, the algorithm first computes all the environment strategies that can enforce $\mathsf{Env} = \forall\varphi_2$ [8], 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 $\forall\varphi_2$.

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

Input: LTL$_f$ formulas φ_1, φ_2
Output: agent strategy σ_{ag} that enforces $\exists\varphi_1$ under $\forall\varphi_2$

1: $\mathcal{A}_1 = \textsc{ConvertDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_{\exists\varphi_1}, \mathsf{reach}(T_1))$
2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_{\forall\varphi_2}, \mathsf{safe}(T_2))$
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: $(R, f_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathsf{reach}((T_1 \times S_2) \cup (Q_1 \times \{\bot_2\}))$
7: **if** $\iota \notin R$ **return** "Unrealisable" **endif**
8: **return** $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}, f_{\mathsf{ag}})$

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

Proof. We show that σ_{ag} returned by Algorithm 4 is winning in $(\mathcal{D}, \mathsf{reach}(T_1 \times S_2) \cup (Q_1 \times \{\bot\}))$ iff $\sigma_{\mathsf{ag}} \rhd_{\mathsf{Env}} \mathsf{Task}$.

We need to show that for every $\sigma_{\mathsf{env}} \rhd \mathsf{Env}$, $\pi = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \mathsf{Task}$. By Lemma 1, the language of the DA $\mathcal{A}_1 = (\mathcal{D}_1, \mathsf{reach}(T_1))$ in Line 1 is $\mathcal{L}(\exists\varphi_1)$ and the language of the DA $\mathcal{A}_2 = (\mathcal{D}_2, \mathsf{safe}(T_2))$ in Line 2 is $\mathcal{L}(\forall\varphi_2)$.

Take an environment strategy $\sigma_{\mathsf{env}} \rhd \mathsf{Env}$. We have that, by Lemma 1, $\pi \models \mathsf{Env}$ iff $\pi \in \mathcal{L}(\mathcal{D}_2, \mathsf{safe}(T_2))$, by Lemma 5, iff $\pi \in \mathcal{L}(\mathcal{D}'_2, \mathsf{safe}(S_2))$, by Lemma 4, iff $\pi \in \mathcal{L}(\mathcal{D}_1 \times \mathcal{D}'_2, \mathsf{safe}(T_1 \times S_2))$.

Now, we need to show that $\pi \models \mathsf{Task}$. σ_{ag} is winning in $(\mathcal{D}, \mathsf{reach}(T_1 \times S_2) \cup (Q_1 \times \{\bot\}))$ then, by Lemma 2, $\pi \in \mathcal{L}(\mathcal{D}, \mathsf{reach}(T_1 \times S_2) \cup (Q_1 \times \{\bot\}))$. However, $\pi \models \mathsf{Env}$, then we need to show that $\pi \in \mathcal{L}(\mathcal{D}, \mathsf{reach}(T_1 \times S_2))$ and $\pi \notin \mathcal{L}(\mathcal{D}, \mathsf{reach}(Q_1 \times \{\bot\}))$. We have that, $\pi \models \mathsf{Task}$ iff $\pi \in \mathcal{L}(\mathcal{D}_1, \mathsf{reach}(T_1))$, by Lemma 1, iff $\pi \in \mathcal{L}(\mathcal{D}_1 \times \mathcal{D}'_2, \mathsf{reach}(T_1 \times S_2))$, by Lemma 4. Then, $\pi \models \mathsf{Task}$. Moreover, we have that if $\pi \in \mathcal{L}(\mathcal{D}'_2, \mathsf{safe}(S_2))$ then $\pi \notin \mathcal{L}(\mathcal{D}'_2, \mathsf{reach}(\{\bot\}))$, and by Lemma 4, $\pi \notin \mathcal{L}(\mathcal{D}_1 \times D'_2, \mathsf{reach}(Q_1 \times \{\bot\}))$. This concludes the proof. □

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 $\mathcal{L}(\forall\varphi_2)$ [8]. 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 $\mathsf{Env} = \forall\varphi_2$, $\mathsf{Task} = \forall\varphi_1$.

Input: LTL$_f$ formulas φ_1, φ_2
Output: agent strategy σ_{ag} that enforces $\forall\varphi_1$ under $\forall\varphi_2$

1: $\mathcal{A}_1 = \textsc{ConvertDA}(\forall\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \mathsf{safe}(T_1))$
2: $\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \mathsf{safe}(T_2))$
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 $\perp_2$
5: $\mathcal{D} = \textsc{Product}(\mathcal{D}_1, \mathcal{D}'_2)$
6: $(S, f_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathsf{safe}((T_1 \times S_2) \cup (Q_1 \times \{\perp_2\})))$
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* $\mathsf{Env} = \forall\varphi_2$, $\mathsf{Task} = \forall\varphi_1$, *where the* φ_i *are* LTL$_f$ *formulas.*

Proof. By Lemma 1, the language of the DA $A_1 = (\mathcal{D}_1, \mathsf{safe}(T_1))$ in Line 1 is $\mathcal{L}(\forall\varphi_1)$ and the language of the DA $A_2 = (\mathcal{D}_2, \mathsf{safe}(T_2))$ in Line 2 is $\mathcal{L}(\forall\varphi_2)$. Finding all the environment strategies that enforce $\mathcal{L}(\mathcal{A}_2)$ through safety games considering the environment as the protagonist is done in Lines 3&4 Remark 2. Restricting all the environment strategies to considering only those that enforce $\mathcal{L}(\mathcal{A}_2)$ is done in Line 5 using Lemma 4. By Lemma 3 it is enough to find check ι is a winning state, and in this case to compute a winning strategy f_{ag} and return $\textsc{Strategy}(f_{\mathsf{ag}})$. But finding the winning region and a winning strategy (for every state in the winning region) for safety games is done in Line 6. □

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 [15] 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 $\forall\varphi_3$.

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

Proof. By Lemma 1, the language of the DA $\mathcal{A}_1 = (\mathcal{D}_1, \mathsf{reach}(T_1))$ in Line 1 is $\mathcal{L}(\exists\varphi_1)$, the language of the DA $\mathcal{A}_2 = (\mathcal{D}_2, \mathsf{safe}, T_2)$ in Line 2 is $\mathcal{L}(\forall\varphi_2)$, and the language of the DA $\mathcal{A}_3 = (\mathcal{D}_3, \mathsf{safe}(T_3))$ in Line 1 is $\mathcal{L}(\forall\varphi_3)$. Finding all the environment strategies that enforce $\mathcal{L}(\mathcal{A}_3)$ through safety games considering the environment as the protagonist is done in Lines 4&5 using Remark 2. Restricting all the environment strategies to considering only those that enforce $\mathcal{L}(\mathcal{A}_3)$ is done in Line 6 using Lemma 4. Finding all the agent strategies that realize $\mathcal{A}_2$ under environment specification $\mathcal{A}_3$ through safety games is done in Lines 7&8 using Remark 2. Note that this step also returns one agent strategy f^s_{ag}. Restricting all the environment strategies to considering only those that enforce $\mathcal{L}(\mathcal{A}_3)$ and agent strategies to considering only those that realize $\mathcal{A}_2$ under environment specification $\mathcal{A}_3$ is done in Line 9 using Lemma 4. Finding the winning region and a winning strategy f^r_{ag} (for every state in the winning region) for reachability games is done in Line 10. By Lemma 3 it is enough to check ι is a winning state, and in this case to compute a winning strategy f_{ag} by combining strategies f^s_{ag} and f^r_{ag} and

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

Input: $\textsc{ltl}_f$ formulas $\varphi_1, \varphi_2, \varphi_3$
Output: agent strategy σ_{ag} that enforces $\exists\varphi_1 \wedge \forall\varphi_2$ under $\forall\varphi_3$

$\mathcal{A}_1 = \textsc{ConvertDA}(\exists\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \mathsf{reach}(T_1))$
$\mathcal{A}_2 = \textsc{ConvertDA}(\forall\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \mathsf{safe}(T_2))$
$\mathcal{A}_3 = \textsc{ConvertDA}(\forall\varphi_3)$, say $\mathcal{A}_3 = (\mathcal{D}_3, \mathsf{safe}(T_3))$
$(S_3, f_{\mathsf{env}}) = \textsc{Solve}_{\mathsf{env}}(\mathcal{A}_3)$
$\mathcal{D}_3' = \textsc{Restrict}(\mathcal{D}_3, S_3)$, call the sink $\bot_3$
$\mathcal{D} = \textsc{Product}(\mathcal{D}_2, \mathcal{D}_3')$
$(S_2, f^s_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}, \mathsf{safe}((T_2 \times S_3) \cup (Q_2 \times \{\bot_3\})))$
$\mathcal{D}' = \textsc{Restrict}(\mathcal{D}, S_2)$, call the sink $\bot_2$
$\mathcal{C} = \textsc{Product}(\mathcal{D}_1, \mathcal{D}')$
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_{ag} to $\mathcal{C}\}$
$(R, f^r_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{C}, \mathsf{reach}((T_1 \times S_2 \times S_3) \cup (Q_1 \times (\eta(S_2) \cup \{\bot_2\}) \times \{\bot_3\}))$
$\{\ \eta : Q_2 \times Q_3 \to Q_2$ is the projection onto Q_2, i.e., $(q_2, q_3) \mapsto q_2\}$
if $\iota \notin R$ **return** "Unrealisable" **endif**
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_{ag} on R, and $f^{s'}_{\mathsf{ag}}$ otherwise.$\}$
return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{C}, f_{\mathsf{ag}})$

return $\textsc{Strategy}(\mathcal{C}, f_{\mathsf{ag}})$. The correctness of $\textsc{Combine}(\mathcal{C}, S, f^s_{\mathsf{ag}}, R, f^r_{\mathsf{ag}})$ is shown by construction: let π be a trace that starts at ι and follows f_{ag}. Since $\iota \in \mathcal{R}$, $\pi \in \mathcal{L}(\mathcal{A}_3)$ holds. Moreover, note that for $q \in R$, the projection of q on the components from Q_2 is in S, hence $\pi \in \mathcal{L}(\mathcal{A}_2)$. $\square$

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 $\mathsf{Env} \to \mathsf{Task}$ [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 $\textsc{ltl}$ is solved in [11].

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 $\textsc{Flagged}(\mathcal{D}, T)$ to be the transition system that, intuitively, records whether a state in T has been seen so far. Formally, $\textsc{Flagged}(\mathcal{D}, T)$ 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 $\textsc{RestrictionWithSinks}(\mathcal{D}, V_0, V_1)$ 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, $\textsc{RestrictionWithSinks}(\mathcal{D}, V_0, V_1)$ 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 $\delta(q, z) \in Q \setminus (V_0 \cup V_1)$.

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

Input: $\textsc{ltl}_f$ formulas $\varphi_1, \varphi_2, \varphi_3, \varphi_4$
Output: agent strategy σ_{ag} that enforces $\exists\varphi_3 \wedge \forall\varphi_4$ under $\forall\varphi_1 \wedge \exists\varphi_2$

$\mathcal{A}_1 = \textsc{ConvertDA}(\exists\neg\varphi_1)$, say $\mathcal{A}_1 = (\mathcal{D}_1, \mathsf{reach}(B_1))$
$\mathcal{A}_2 = \textsc{ConvertDA}(\forall\neg\varphi_2)$, say $\mathcal{A}_2 = (\mathcal{D}_2, \mathsf{safe}(B_2))$
$\mathcal{A}_3 = \textsc{ConvertDA}(\exists\varphi_3)$, say $\mathcal{A}_3 = (\mathcal{D}_3, \mathsf{reach}(B_3))$
$\mathcal{A}_4 = \textsc{ConvertDA}(\forall\varphi_4)$, say $\mathcal{A}_4 = (\mathcal{D}_4, \mathsf{safe}(B_4))$
$\mathcal{D}_p = \textsc{Product}(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4)$
Let Q_p be the state set of $\mathcal{D}_p$, and T_i the lift of B_i to Q_p (for $i \le 4$)
$(R_1, f^1_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}_p, \mathsf{reach}(T_1))$
$\mathcal{D}_p' = \textsc{Restrict}(\mathcal{D}_p, Q \setminus R_1)$
$\mathcal{D}^f = \textsc{Flagged}(\mathcal{D}_p', T_3)$
$(S_2, f^2_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}^f, \mathsf{safe}(T_2))$
$(S_4, f^4_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\mathcal{D}^f, \mathsf{safe}(T_4))$
$(R_3, f^3_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\textsc{Restrict}(\mathcal{D}^f, S_4), \mathsf{reach}(T_3))$
$V_0 = (Q^f \setminus (S_2 \cup S_4)) \cup ((S_4 \cap T_2) \setminus (R_3 \cup S_2))$
V_1 is all states in $(S_4 \setminus T_2) \setminus (R_3 \cup S_2)$ whose flag is set to *no*
$\hat{\mathcal{D}} = \textsc{RestrictionWithSinks}(\mathcal{D}^f, V_0, V_1)$
$(E, f^e_{\mathsf{ag}}) = \textsc{Solve}_{\mathsf{ag}}(\hat{\mathcal{D}}, \mathsf{safe}((T_2 \cap T_4) \cup \{\top\}))$
$W_{\mathsf{ag}} = S_2 \cup R_3 \cup E$ {Note that $W_{\mathsf{ag}} \subseteq Q^f \cup \{\top\}$}
if $\iota \notin W_{\mathsf{ag}}$ **return** "Unrealisable" **endif**
$f_{\mathsf{ag}} = \textsc{Combine}(\mathcal{D}^f, f^1_{\mathsf{ag}}, f^2_{\mathsf{ag}}, f^3_{\mathsf{ag}}, f^4_{\mathsf{ag}}, R_1, S_2, R_3, E)$ {See the definition below.}
return $\sigma_{\mathsf{ag}} = \textsc{Strategy}(\mathcal{D}^f, f_{\mathsf{ag}})$

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 $f_{\mathsf{ag}} : (Q^f)^+ \to 2^{\mathcal{Y}}$ 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 by a finite-state transducer. So in the Algorithm 7, as before, with a little abuse of notation we write directly $\sigma_{\mathsf{ag}} =$ Strategy($\mathcal{D}^f$, f_{ag}), to mean that we return its representation as a transducer.

Lemma 6. *To solve any game with objective $(\Diamond T) \vee \beta$ (for $\beta \subseteq Q^\omega$) proceed as follows:*

1. *Let $(M, m_{\mathsf{ag}}) =$ Solve($\mathcal{D}$, reach(T)).*
2. *Let $\mathcal{D}'$ be the restriction of $\mathcal{D}$ to $Q \setminus M$ (this introduces a new sink state).*
3. *Let $(N, n_{\mathsf{ag}}) =$ Solve($\mathcal{D}'$, β).*
4. *Let $W = M \cup N$.*
5. *Define f_{ag} as follows: if the history ever enters M follow m_{ag}; if the starting state is in N, follow n_{ag}; otherwise make an arbitrary move.*

Proof. To see that this is correct, let ρ be a path in $\mathcal{D}$ that starts in $q \in W$ and is consistent with f_{ag}. Note that if $\rho_i \in M$ for some i, then $\rho_{\geq i}$ is consistent with m_{ag} and thus is in (reach, T), and thus $\rho \in$ reach(T). In particular, if $q \in M$ then $\rho \in$ reach(T). If $q \in N$ then $\rho_i \in Q \setminus M$ for all i, and so it is consistent with n_{ag}, and so satisfies β. Thus we have shown that f_{ag} enforces the objective from W.

We must now show that the opponent enforces the negation of the objective, i.e., $(\Box \neg T) \wedge \neg\beta$, from $Q \setminus W$. For this, let f_{env} be the opponent's strategy that enforces $\neg\varphi$ in $\mathcal{D}'$, i.e., every play in $\mathcal{D}'$ consistent with f_{env} that starts in $Q \setminus W$ satisfies $\neg\varphi$. Let ρ be a path in $\mathcal{D}$ that starts in $q \in Q \setminus W$ and is consistent with f_{env}. Note that the play stays in $\mathcal{D}'$ since $Q \setminus M$ is a trap for the player; thus ρ satisfies $\Box \neg T$ (since $T \subseteq M$) and $\neg\varphi$, as required. □

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

Proof. By Lemma 6, it is enough to show that the algorithm correctly solves the game produced at the end of Line 8: the transition system is $\mathcal{D}'_p$ and the winning condition is safe(T_2) $\cup$ (safe(T_4) $\cap$ reach(T_3)). First, note that Line 9 only refines the state space (by adding the flag component), and so it is enough to show that the rest of the algorithm correctly solves the game on $\mathcal{D}^f$ produced at Line 9.

To do this we will show that f_{ag} enforces $\alpha =$ safe(T_2) $\cup$ (safe(T_4) $\cap$ reach(T_3)) from W_{ag}, and that the environment has a strategy f_{env} that enforces the complement of α, i.e., $\overline{\text{safe}(T_2)} \cap (\overline{\text{safe}(T_4)} \cup \overline{\text{reach}(T_3)})$, from the rest of the states.

Let ρ be a path in $\mathcal{D}^f$ consistent with f_{ag}.

- If ρ starts in S_2 then it satisfies safe(T_2) since it uses f^2_{ag}.
- If ρ starts in R_3 then it satisfies reach(R_3) $\cap$ safe(T_4) since it uses f^3_{ag} followed by f^4_{ag}.
- Suppose ρ starts in E. If it stays in V_0 then it satisfies safe(T_2) $\cap$ safe(T_4), and thus safe(T_2), since it follows f^e_{ag}. If it leaves V_0 then the corresponding path in $\mathcal{D}^f$ reaches $\top$. Let $s \in Q^f \setminus V_0$ be the target of this last transition in $\mathcal{D}^f$. Note that every state on the path up to (but not necessarily including s) is in $T_2 \cap T_4$. There are three cases.
  - If $s \in S_2$ proceed as before and use f^2_{ag}; since T_2 held up till now, $\rho \in$ safe(T_2).
  - If $s \in R_3$ proceed as before and use f^3_{ag} until T_3 is visited and then switch to f^4_{ag}; since T_4 held up till now, $\rho \in$ safe(T_4) $\cap$ reach(T_3).
  - If $s \in (S_4 \setminus T_2) \setminus (R_3 \cup S_2)$ then the value of the flag at s is *yes* and thus T_3 was seen, so switch to f^4_{ag} to also ensure safe(T_4).

To define the environment strategy f_{env} we need some strategies from the construction.

- Let S'_2 and f^2_{env} be the winning region and winning strategy in $\mathcal{D}^f$, respectively, for the environment in Line 10. So, a play that follows f^2_{env} from S'_2 violates safe(T_2).
- Let S'_4 and f^4_{env} be the winning region and winning strategy in $\mathcal{D}^f$, respectively, for the environment in Line 11. So, a play that follows f^4_{env} from S'_4 violates safe(T_4).
- Let R'_3 and f^3_{env} be the winning region and winning strategy in $\mathcal{D}^f$, respectively, for the environment in Line 12. So, a play that follows f^3_{env} from R'_3 will never visit T_3 if it stays in the restricted arena (Line 12), and if it leaves S_4 then it is in S'_4, a previous case in which the environment can violate safe(T_4).
- Let E' be such that $E' \cup \{\bot\}$ is the winning region for the environment in Line 16, and let f^e_{env} be a winning strategy. So, for a play that follows f^e_{env} starting in E', if it stays in this domain then eventually a state is reached that is either not in T_2 or not in T_4, and if it leaves this domain it enters a state in $S_4 \setminus (R_3 \cup S_2 \cup T_2) = S_4 \setminus (R_3 \cup T_2)$ in which the value of the flag is *no* meaning that T_3 has not been visited yet.

Define $W_{\mathsf{env}} := Q^f \setminus W_{\mathsf{ag}} = (S_4 \setminus (R_3 \cup S_2 \cup T_2)) \cup E' = (S_4 \setminus (R_3 \cup T_2)) \cup E'$. We now define f_{env}. Suppose q is the first state of the history.

1. If $q \notin W_{\mathsf{env}}$ then define f_{env} arbitrarily.
2. If $q \in S_4 \setminus (R_3 \cup T_2)$ then in particular $q \notin T_2$ and already safe(T_2) is violated so f_{env} follows f^3_{env} to ensure that reach(R_3) is violated (or if play leaves S_4 then safe(T_4) is violated).
3. If q is in E' then f_{env} follows f^e_{env}. This ensures that eventually a state s is reached such that at least one of the following cases hold:
   (a) $s \notin T_4$ (so in particular safe(T_4) is violated). Since S_2 is disjoint from V, let f_{env} switch to f^2_{env} so that safe(T_2) is violated.
   (b) $s \notin T_2$ (so in particular safe(T_2) is violated). Since $S_4 \setminus T_2$ is disjoint from V_0, let f_{env} switch to f^4_{env} so that safe(T_4) is violated.
   (c) $s \in S_4 \setminus (R_3 \cup T_2)$ and its flag value is *no*. In this case f_{env} switches to f^e_{env} and wins as if it started in this state (as above). □

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 $\mathsf{Env} \to \mathsf{Task}$ [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 $\mathsf{Env} = \mathit{true}$ 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$_f$ [18], and Pure-Past LTL [13], 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 [25]: 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 [5,3,6,9,20].

CRediT authorship contribution statement

Benjamin Aminof: Conceptualization. **Giuseppe De Giacomo:** Supervision. **Antonio Di Stasio:** Formal analysis, Conceptualization. **Hugo Francon:** Formal analysis. **Sasha Rubin:** Formal analysis, Conceptualization. **Shufang Zhu:** Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: ANTONIO DI STASIO reports travel was provided by European Research Council. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by the ERC Advanced Grant WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR, Italy (No. PE0000013).

References

[1] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, Sasha Rubin, Planning and synthesis under assumptions, CoRR (2018).
[2] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, Sasha Rubin, Planning under LTL environment specifications, in: ICAPS, 2019, pp. 31–39.
[3] Benjamin Aminof, Giuseppe De Giacomo, Alessio Lomuscio, Aniello Murano, Sasha Rubin, Synthesizing best-effort strategies under multiple environment specifications, in: KR, 2021, pp. 42–51.
[4] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, Sasha Rubin, Synthesis under assumptions, in: KR, AAAI Press, 2018, pp. 615–616.
[5] Benjamin Aminof, Giuseppe De Giacomo, Sasha Rubin, Best-effort synthesis: doing your best is not harder than giving up, in: IJCAI, ijcai.org, 2021, pp. 1766–1772.
[6] Benjamin Aminof, Giuseppe De Giacomo, Sasha Rubin, Florian Zuleger, Stochastic best-effort strategies for Borel goals, in: LICS, 2023, pp. 1–13.
[7] Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi, Shufang Zhu, Compositional safety LTL synthesis, in: Akash Lal, Stefano Tonetta (Eds.), VSTTE 2022, vol. 13800, 2022, pp. 1–19.
[8] Julien Bernet, David Janin, Igor Walukiewicz, Permissive strategies: from parity games to safety games, RAIRO Theor. Inform. Appl. 36 (3) (2002) 261–275.
[9] Dietmar Berwanger, Admissibility in infinite games, in: STACS, in: Lecture Notes in Computer Science, vol. 4393, Springer, 2007, pp. 188–199.
[10] Alberto Camacho, Meghyn Bienvenu, Sheila A. McIlraith, Finite LTL synthesis with environment assumptions and quality measures, in: KR, 2018, pp. 454–463.
[11] Alberto Camacho, Meghyn Bienvenu, Sheila A. McIlraith, Towards a unified view of AI planning and reactive synthesis, in: ICAPS, 2019, pp. 58–67.
[12] Edward Chang, Zohar Manna, Amir Pnueli, The safety-progress classification, in: Logic and Algebra of Specification, 1993, pp. 143–202.
[13] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, Sasha Rubin, Pure-past linear temporal and dynamic logic on finite traces, in: IJCAI, 2020, pp. 4959–4965.
[14] Giuseppe De Giacomo, Antonio Di Stasio, Giuseppe Perelli, Shufang Zhu, Synthesis with mandatory stop actions, in: KR, 2021, pp. 237–246.
[15] Giuseppe De Giacomo, Antonio Di Stasio, Luca M. Tabajara, Moshe Y. Vardi, Shufang Zhu, Finite-trace and generalized-reactivity specifications in temporal synthesis, in: IJCAI, 2021, pp. 1852–1858.
[16] Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y. Vardi, Shufang Zhu, Two-stage technique for LTL_f synthesis under LTL assumptions, in: KR, 2020, pp. 304–314.
[17] Giuseppe De Giacomo, Sasha Rubin, Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals, in: IJCAI, 2018, pp. 4729–4735.
[18] Giuseppe De Giacomo, Moshe Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013, pp. 854–860.
[19] Giuseppe De Giacomo, Moshe Y. Vardi, Synthesis for LTL and LDL on finite traces, in: IJCAI, 2015, pp. 1558–1564.
[20] Marco Faella, Admissible strategies in infinite games over graphs, in: MFCS, in: Lecture Notes in Computer Science, vol. 5734, Springer, 2009, pp. 307–318.
[21] Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, Mateusz Skomra, Games on Graphs, 2023.
[22] Hector Geffner, Blai Bonet, A Concise Introduction to Models and Methods for Automated Planning, Morgan&Claypool, 2013.
[23] Erich Grädel, Wolfgang Thomas, Thomas Wilke (Eds.), Automata, Logics, and Infinite Games: A Guide to Current Research [Outcome of a Dagstuhl Seminar, February 2001], Lecture Notes in Computer Science, vol. 2500, Springer, 2002.
[24] Orna Kupferman, Moshe Y. Vardi, Model checking of safety properties, Form. Methods Syst. Des. 19 (3) (2001) 291–314.
[25] Zohar Manna, Amir Pnueli, A hierarchy of temporal properties, in: PODC, 1990, pp. 377–410.
[26] D.A. Martin, Borel determinacy, Ann. Math. 65 (1975) 363–371.
[27] Amir Pnueli, The temporal logic of programs, in: FOCS, 1977, pp. 46–57.
[28] Amir Pnueli, Roni Rosner, On the synthesis of a reactive module, in: POPL, 1989, pp. 179–190.
[29] Michael O. Rabin, Dana S. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (2) (1959) 114–125.
[30] Lucas M. Tabajara, Moshe Y. Vardi, Ltlf synthesis under partial observability: from theory to practice, in: GandALF, in: EPTCS, vol. 326, 2020, pp. 1–17.
[31] Lucas Martinelli Tabajara, Moshe Y. Vardi, Partitioning techniques in ltlf synthesis, in: IJCAI, 2019, pp. 5599–5606.
[32] Tabakov Deian, Moshe Y. Vardi, Experimental evaluation of classical automata constructions, in: LPAR, in: Lecture Notes in Computer Science, vol. 3835, 2005, pp. 396–411.
[33] Michael Westergaard, Better algorithms for analyzing and enacting declarative workflow languages using LTL, in: BPM, in: Lecture Notes in Computer Science, vol. 6896, 2011, pp. 83–98.
[34] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, Moshe Y. Vardi, LTL_f synthesis with fairness and stability assumptions, in: AAAI, 2020, pp. 3088–3095.
[35] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, Moshe Y. Vardi, Symbolic LTL_f synthesis, in: IJCAI, 2017, pp. 1362–1369.
[36] Shufang Zhu, Lucas M. Tabajara, Geguang Pu, Moshe Y. Vardi, On the power of automata minimization in temporal synthesis, in: GandALF, in: EPTCS, vol. 346, 2021, pp. 117–134.