Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Synthesizing Best-effort Strategies under Multiple Environment Specifications

Benjamin Aminof[1] , **Giuseppe De Giacomo**[2] , **Alessio Lomuscio**[3] ,
Aniello Murano[4] and **Sasha Rubin**[5]
[1] TU Vienna, Austria
[2] University of Rome "La Sapienza", Italy
[3] Imperial College London, UK
[4] University of Naples "Federico II", Italy
[5] University of Sydney, Australia
aminof@forsyte.at, degiacomo@diag.uniroma1.it, a.lomuscio@imperial.ac.uk, aniello.murano@unina.it, sasha.rubin@sydney.edu.au

Abstract

We formally introduce and solve the synthesis problem for LTL goals in the case of multiple, even contradicting, assumptions about the environment. Our solution concept is based on "best-effort strategies" which are agent plans that, for each of the environment specifications individually, achieve the agent goal against a maximal set of environments satisfying that specification. By means of a novel automata theoretic characterization we demonstrate that this best-effort synthesis for multiple environments is 2EXPTIME-complete, i.e., no harder than plain LTL synthesis. We study an important case in which the environment specifications are increasingly indeterminate, and show that as in the case of a single environment, best-effort strategies always exist for this setting. Moreover, we show that in this setting the set of solutions are exactly the strategies formed as follows: *amongst* the best-effort agent strategies for φ under the environment specification $\mathcal{E}_1$, find those that do a best-effort for φ under (the more indeterminate) environment specification $\mathcal{E}_2$, and *amongst those* find those that do a best-effort for φ under the environment specification $\mathcal{E}_3$, etc.

1 Introduction

An artificial agent should have automated reasoning capabilities to decide and plan which actions to perform to achieve its goals in its environment. While early work in planning focused on reachability goals in environments described as planning domains, recent approaches consider more sophisticated goals and environment specifications expressed via automata or temporal logic (Bacchus and Kabanza 1996; D'Ippolito, Rodríguez, and Sardiña 2018; De Giacomo and Rubin 2018; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2019a). These have strong correspondences with the reactive synthesis problem in theoretical computer science, see e.g., (Camacho, Bienvenu, and McIlraith 2019). In the classic formulation (Pnueli and Rosner 1989), the synthesis problem concerns finding an agent strategy that guarantees that a specification formula, expressed in linear- time temporal logic (LTL) (Pnueli 1977), is satisfied irrespective of the actions of the environment.

In most cases, the agent has one model of the environment (specified, e.g., in LTL) which it uses to deliberate to achieve its goals. Yet, some recent work (Aminof et al. 2020; Ciolek et al. 2020) has argued that it is not realistic in complex AI scenarios to have a single specification of the environment. Indeed, an agent may have multiple (possibly even contradictory) assumptions about the environment's behavior. For example, an agent may assume certain expected behavior from the environment, yet may still want to consider deteriorated environments where certain exceptional faults may arise. In these cases, one solution is to plan or synthesize against the environment that is considered to be most likely, but this may not lead to an ideal response.

In this paper, instead, we study how to do synthesis simultaneously for all such models. Humans often reason in this way when choosing actions, preferring strategies that may be more resilient to a variety of circumstances, should these materialize. Simply put, not only do we want to succeed if the environment behaves as we expect, but also if it does not, or at least we also want to do the *best we can*. Indeed, it may simply be impossible to derive a strategy that succeeds under all circumstances. Yet, we are interested in defining and understanding what the best behavior for the agent may be under these conditions. We focus on the general case where the agent tasks and alternative environment specifications are expressed in LTL. In this setting we study *best-effort synthesis under multiple environment specifications*. Our main contributions are:

1. We define a general notion of best-effort synthesis under multiple environment specifications.
2. We study the important natural case of chains of environment specifications, where the first, the nominal environment, can enact less behaviors (or strategies), and the further one goes down the chain the more environment behaviors the agent considers possible. We show that in this case there is always an agent strategy that is *simultaneously* best effort under all environments in the chain.
3. We give a characterization of best-effort strategies in terms of tree-automata.
4. We use this automata characterization to achieve an optimal 2EXPTIME algorithm for best-effort synthesis under multiple environment specifications.

Notably, the latter result shows that the increased expressivity in our formulation does not result in an increase of worst-case complexity with respect to classical LTL synthesis (Pnueli and Rosner 1989).

2 Preliminaries

Given a sequence x, we write x_i for its ith element; the first element is x_0; the length of a finite sequence is $|x|$; the prefix of x of length $0 \leq i \leq |x|$ is denoted by $x_{<i}$ or $x_{\leq i-1}$.

Linear-time Temporal Logic (LTL) For a set AP of atomic propositions, *formulas of* LTL *over* AP are defined by the following BNF (where $p \in AP$):

$$\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

We use the usual abbreviations, $\varphi \supset \varphi' \doteq \neg\varphi \vee \varphi'$, $\mathsf{true} \doteq p \vee \neg p$, $\mathsf{F}\,\varphi \doteq \mathsf{true}\,\mathsf{U}\,\varphi$, $\mathsf{G}\,\varphi \doteq \neg\mathsf{F}\neg\varphi$, etc. The *size* $|\varphi|$ of a formula φ is the number of symbols in it. A *trace* $\tau \in (2^{AP})^\omega$ is an infinite sequence of valuations of the atoms. For $n \geq 0$, write τ_n for the valuation at position n. Given a trace τ, an integer n, and an LTL formula φ, the satisfaction relation $(\tau, n) \models \varphi$, stating that φ holds at step n of the sequence τ, is defined as follows:

- $(\tau, n) \models p$ iff $p \in \tau_n$;
- $(\tau, n) \models \varphi_1 \vee \varphi_2$ iff $(\tau, n) \models \varphi_1$ or $(\tau, n) \models \varphi_2$;
- $(\tau, n) \models \neg\varphi$ iff it is not the case that $(\tau, n) \models \varphi$;
- $(\tau, n) \models \mathsf{X}\,\varphi$ iff $(\tau, n+1) \models \varphi$; and
- $(\tau, n) \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff there exists $m \geq n$ such that: $(\tau, m) \models \varphi_2$ and $(\tau, j) \models \varphi_1$ for all $n \leq j < m$.

We write $\tau \models \varphi$ if $(\tau, 0) \models \varphi$, read τ *satisfies* φ.

Reactive Synthesis The problem of (reactive) synthesis is to construct an agent that achieves a goal while continually interacting with its environment. We specify goals and environments by LTL formulas. Let X and Y be disjoint finite sets of Boolean variables, called the *environment variables* and *agent variables*, respectively. Then 2^X (resp. 2^Y) is the set of environment (resp. agent) *moves*. A *play* $\pi \doteq X_0 \cdot Y_0 \cdot X_1 \cdot Y_1 \cdots$ is an element of $(2^X \cdot 2^Y)^\omega$. The trace induced by π is the infinite sequence $(X_i \cup Y_i)_{i \geq 0}$. Given an LTL formula φ over $AP \doteq X \cup Y$, say that π *satisfies* φ, *written* $\pi \models \varphi$, if the trace induced by π satisfies φ.

A *history* h is a finite prefix of a play. An *agent strategy* is a function $\sigma_{\mathsf{ag}} : (2^X \cdot 2^Y)^* \cdot 2^X \to 2^Y$ that maps histories ending in environment moves to agent moves. Similarly, an *environment strategy* is a function $\sigma_{\mathsf{env}} : (2^X \cdot 2^Y)^* \to 2^X$ that maps histories ending in agent moves (including, since the environment moves first, the empty history λ) to environment moves. $\Sigma^{all}_{\mathsf{ag}}$ denotes the set of all agent strategies.

While our definition of strategies of a player as functions from (joint) histories is quite common in the game theory literature, one can also define strategies as functions from histories projected on opponent moves only. The former definition is more general, as it also specifies what to do on histories that are not consistent with the strategy. Note, however, that one can easily obtain the latter from the former. For example, given an agent strategy $\sigma_{\mathsf{ag}} : (2^X \cdot 2^Y)^* \cdot 2^X \to 2^Y$, one can obtain $\sigma'_{\mathsf{ag}} : (2^X)^+ \to 2^Y$ by taking $\sigma'_{\mathsf{ag}}(x) \doteq \sigma_{\mathsf{ag}}(f(x))$, where $f(x)$ is the unique joint history of odd length consistent with σ_{ag} whose projection on environment moves is x, i.e., define f inductively by: $f(x_0) \doteq x_0$; and $f(x_{<i}) \doteq f(x_{<i-1}) \cdot \sigma_{\mathsf{ag}}(f(x_{<i-1})) \cdot x_{i-1}$.

For an agent (resp. env) strategy σ and a play/history ρ, say that σ *and* ρ *are consistent*, if for every proper prefix $\rho_{<i}$ of odd (resp. even) length we have that $\rho_i = \sigma(\rho_{<i})$. Let $\textsc{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$ be the unique play consistent with both σ_{ag} and σ_{env}. An agent strategy σ_{ag} *enforces* ψ, written $\sigma_{\mathsf{ag}} \rhd \psi$, iff $\textsc{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \psi$ for every environment strategy σ_{env}; if such a strategy exists we say the *agent can enforce* ψ. A symmetric definition holds for environment strategies.

We write $\Sigma^{\mathcal{E}}_{\mathsf{env}}$ for the set of all environment strategies that enforce $\mathcal{E}$. If $\Sigma^{\mathcal{E}}_{\mathsf{env}} \neq \emptyset$ we say that $\mathcal{E}$ is an *environment specification* (aka *assumption*). The idea of treating environment specifications as environment-enforceable formulas is justified in (Aminof et al. 2018). Write $\Sigma^{\mathcal{E},h}_{\mathsf{env}}$ for those environment strategies that enforce $\mathcal{E}$ and are consistent with the history h. For the rest of this paper, we use $\mathcal{E}$ to denote an environment specification. The *synthesis under assumption problem* asks, for an LTL *goal* φ and an LTL environment specification $\mathcal{E}$, to find (if there is one) an agent strategy σ_{ag} such that $\textsc{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$ satisfies φ for every σ_{env} that enforces $\mathcal{E}$, and in this case we say that σ_{ag} *enforces* φ *under* $\mathcal{E}$. This problem is 2EXPTIME-complete (Aminof et al. 2019a; Aminof et al. 2018). The case $\mathcal{E} = \mathsf{true}$, called the *synthesis problem*, was pioneered in (Pnueli and Rosner 1989).

For the constructions in this paper, we will also need refined notions that start at a history h. Let *the* $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ *extension of* h be the unique play π, extending h, satisfying for every $i \geq |h|$ that if i is odd then $\sigma_{\mathsf{ag}}(h_{<i}) = h_i$ and if i is even then $\sigma_{\mathsf{env}}(h_{<i}) = h_i$. Observe that the $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ extension of h is equal to $\textsc{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$ if h is consistent with both σ_{ag} and σ_{env}. An agent strategy σ_{ag} *enforces* ψ *from* h, written $\sigma_{\mathsf{ag}} \rhd_h \psi$, if the $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ extension of h satisfies ψ for every environment strategy σ_{env}; if such a σ_{ag} exists we say that *the agent can enforce* ψ *from* h.

Given agent (resp. env) strategies σ, σ', and a history h, let $\sigma'' \doteq \sigma[h \leftarrow \sigma']$ be the following strategy: for a history h' ending in an environment (resp. agent) move, let $\sigma''(h') \doteq \sigma'(h')$ if h is a prefix of h', and let $\sigma''(h') \doteq \sigma(h')$ otherwise. Informally, σ'' does what σ' does from h onwards, and elsewhere does what σ does. We say that σ'' is a *shift* of σ by σ' at h. The following is immediate:

Lemma 1. *If* $\sigma_h \in \Sigma^{\mathcal{E},h}_{\mathsf{env}}$ *and* $\sigma_{\mathsf{env}} \rhd_h \mathcal{E}$, *then* $\sigma_h[h \leftarrow \sigma_{\mathsf{env}}] \in \Sigma^{\mathcal{E},h}_{\mathsf{env}}$, *and it agrees with* σ_{env} *on* h *and its extensions.*

The next Lemma characterizes plays that are not consistent with all environment strategies enforcing a given $\mathcal{E}$.

Lemma 2. *Given an* LTL *formula* $\mathcal{E}$, *a play* π *is inconsistent with all environment strategies enforcing* $\mathcal{E}$ *iff: (i)* π *does not satisfy* $\mathcal{E}$*; or (ii) the environment can not enforce* $\mathcal{E}$ *(equivalently, the agent can enforce* $\neg\mathcal{E}$*) from some prefix* h *of* π.

Say that σ_{ag} *enforces* ψ *where possible* if σ_{ag} enforces ψ from h for every history h from which the agent can enforce ψ. Note that if the agent can enforce ψ then an agent strategy enforcing ψ where possible in particular enforces ψ. A similar definition holds for environment strategies.

Lemma 3. *For every* LTL *formula ψ, there is an environment (resp. agent) strategy that enforces ψ where possible.*

Proof. Define an environment strategy (the agent case is similar) σ by considering all histories h of even length, in length-lexicographic order. If σ is not yet defined on h then define it as follows: if there is σ^h_{env} s.t. $\sigma^h_{\mathsf{env}} \rhd_h \psi$, define σ to be equal to σ^h_{env} on h and all histories extending h that are consistent with σ^h_{env}; otherwise, define σ only on h, arbitrarily. The construction is correct since σ^h_{env} also witnesses the fact that the environment can enforce ψ from h' for every extension h' of h that is consistent with σ^h_{env}. □

3 Synthesizing for Multiple Environments

We ground our notion of "best-effort" on the game-theoretic notion of dominance (Apt and Grädel 2011), and in our context of reactive synthesis we draw on (Aminof et al. 2020). We first recall what it means for an agent strategy to dominate (i.e., be no worse than) another with respect to an agent goal φ and a single environment specification $\mathcal{E}$.

Definition 1. *Let φ be an agent goal, and let $\mathcal{E}$ be an environment specification. The* dominance *order $\geq_{\varphi|\mathcal{E}}$ on agent strategies is defined by $\sigma \geq_{\varphi|\mathcal{E}} \sigma'$ iff for every $\sigma_{\mathsf{env}} \rhd \mathcal{E}$, $\text{PLAY}(\sigma', \sigma_{\mathsf{env}}) \models \varphi$ implies $\text{PLAY}(\sigma, \sigma_{\mathsf{env}}) \models \varphi$.*

We write $\sigma >_{\varphi|\mathcal{E}} \sigma'$ if $\sigma \geq_{\varphi|\mathcal{E}} \sigma'$ and $\sigma' \not\geq_{\varphi|\mathcal{E}} \sigma$. In this case we say that *σ strictly dominates σ' (for goal φ under environment specification $\mathcal{E}$)*. Note that $\geq_{\varphi|\mathcal{E}}$ is a preorder, and $>_{\varphi|\mathcal{E}}$ is a strict partial order.

Definition 2. *Let φ be an agent goal, $\mathcal{E}$ be an environment specification, and Σ_{ag} be a set of agent strategies. We say that an agent strategy $\sigma_{\mathsf{ag}} \in \Sigma_{\mathsf{ag}}$ is* maximal *(also called* best-effort*), wrt $\Sigma_{\mathsf{ag}}, \varphi, \mathcal{E}$, if there is no $\sigma'_{\mathsf{ag}} \in \Sigma_{\mathsf{ag}}$ such that $\sigma'_{\mathsf{ag}} >_{\varphi|\mathcal{E}} \sigma_{\mathsf{ag}}$. Write $\text{MAX}_{\varphi|\mathcal{E}}(\Sigma_{\mathsf{ag}})$ for the set of strategies in Σ_{ag} that are maximal wrt $\Sigma_{\mathsf{ag}}, \varphi, \mathcal{E}$.*

In the definition above, in case $\Sigma_{\mathsf{ag}} = \Sigma^{all}_{\mathsf{ag}}$ (the set of all agent strategies), we talk about the *maximal strategies for φ under $\mathcal{E}$*. Intuitively, if $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$ then σ_1 does at least as well as σ_2 against every environment strategy enforcing $\mathcal{E}$, and strictly better against at least one such environment strategy. In particular, if σ_2 is not maximal and σ_1 strictly dominates it, then an agent that uses σ_2 is not doing its "best" to achieve the goal: if it used σ_1 instead, it could achieve the goal against a strictly larger set of environment strategies. Hence, within our framework, we consider maximal strategies as doing a "best-effort" to achieve the goal, and use the terms "maximal" and "best-effort" interchangeably.

Remark 1. *Enforcing strategies dominate all others, i.e., if φ is agent-enforceable under $\mathcal{E}$, then $\text{MAX}_{\varphi|\mathcal{E}}(\Sigma^{all}_{\mathsf{ag}})$ is the set of strategies enforcing φ under $\mathcal{E}$. On the other hand, in general, maximal strategies may not dominate all others.*

The Synthesis Problem Having recalled best-effort strategies we now define the novel problem of synthesizing them under multiple environment specifications. Our solution concept requires strategies to be best-effort for the goal under each of the environment specifications individually:

Definition 3 (Best-effort Synthesis under Multiple Environment Specifications)**.** *Given an* LTL *goal φ and environment specifications $\mathcal{E}_1, \ldots, \mathcal{E}_n$, return an agent strategy in $\bigcap \text{MAX}_{\varphi|\mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$, or return "no solution" if there is none. We call any such strategies* best-effort *(wrt. $\varphi, \mathcal{E}_1, \ldots, \mathcal{E}_n$).*

The case of a single environment specification ($n = 1$) was indirectly studied in (Aminof et al. 2020) who: a) proved a best-effort strategy always exists, and b) gave a non-optimal algorithm that finds such a strategy.

The problem may have no solution in the case $n > 1$. For instance, in a game reminiscent of matching pennies, in which the agent first chooses $x \in \{H, T\}$ and then the environment chooses $y \in \{H, T\}$, and the goal of the agent is to have $x = y$, then, for $y \in \{H, T\}$ writing $\mathcal{E}_y$ for the assumption that the environment will play y, a best-effort (in fact winning) strategy assuming $\mathcal{E}_y$ is for the agent is to play y; thus there is no agent strategy that is best-effort for both environments $\mathcal{E}_H$ and $\mathcal{E}_T$. We will see in Section 3 that the synthesis problem under one or multiple environment assumptions is 2EXPTIME-complete.

Chains of Environment Specifications We consider an important special case of our problem in which the assumptions form a chain $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$, i.e., every environment strategy that enforces $\mathcal{E}_i$ also enforces $\mathcal{E}_{i+1}$, for $i < n$. We show below that, remarkably, in this setting *there always exists a solution*. Moreover, the second example in Section 4 shows that *every assumption in the chain may be needed to identify the solutions*. In other words we cannot just take the least or the greatest element in the chain to find the solution (in particular, $\bigcap \text{MAX}_{\varphi|\mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$ is not always the same as $\text{MAX}_{\varphi|\bigwedge \mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$ or as $\text{MAX}_{\varphi|\bigvee \mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$). Moreover, the set of solutions are exactly the strategies formed as follows: *amongst* the best-effort agent strategies for φ under the environment specification $\mathcal{E}_1$, find those that do a best-effort for φ under the environment specification $\mathcal{E}_2$, and *amongst those* find those that do a best-effort for φ under the environment specification $\mathcal{E}_3$, etc. This is of practical significance, because, intuitively, we can consider the chain as encoding an order of environment specifications from the most to the least determinate. The special case of $n = 2$, where the agent can enforce the goal under $\mathcal{E}_1$, was studied in (Aminof et al. 2020) where it was shown that the best-effort synthesis problem is decidable (but their algorithm is not optimal).

Formally, consider a set of environment specifications $\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_n$, each expressed in LTL, such that, for every $i < n$, an environment strategy that enforces $\mathcal{E}_i$ also enforces $\mathcal{E}_{i+1}$, i.e., $\Sigma^{\mathcal{E}_i}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_{i+1}}_{\mathsf{env}}$. This setting will also be used for the examples discussed in Section 4.

The following theorem states that in this setting the synthesis problem in Definition 3 always has a solution:

Theorem 1. *Given* LTL *formulas $\varphi, \mathcal{E}_1, \ldots, \mathcal{E}_n$ satisfying $\Sigma^{\mathcal{E}_i}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_{i+1}}_{\mathsf{env}}$ for every $i < n$, the set of best-effort strategies $\bigcap_i \text{MAX}_{\varphi|\mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$ is non-empty.*

This theorem follows immediately from the following two semantic properties which are of interest in their own right. The first property states that the set of strategies that solve

the best-effort synthesis problem under a chain of environments can be formed by iteratively choosing amongst best-effort strategies for the goal under the current environment from those already selected from the previous environment:

Theorem 2. *Given* LTL *formulas* $\varphi, \mathcal{E}_1, \ldots, \mathcal{E}_n$ *satisfying* $\Sigma^{\mathcal{E}_i}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_{i+1}}_{\mathsf{env}}$ *for every* $i < n$*, and a set* Σ_{ag} *of agent strategies, the set* $\text{MAX}_{\varphi|\mathcal{E}_n}(\text{MAX}_{\varphi|\mathcal{E}_{n-1}\ldots}(\text{MAX}_{\varphi|\mathcal{E}_1}(\Sigma_{\mathsf{ag}}))\ldots)$ *is equal to* $\bigcap_{i=1}^{n} \text{MAX}_{\varphi|\mathcal{E}_i}(\Sigma_{\mathsf{ag}})$.

Proof sketch. The proof is by induction on n. The case $n = 1$ is trivial, the case $n = 2$ was essentially proved in (Aminof et al. 2020). For $n > 2$, let $\Sigma^k_{\mathsf{ag}} \doteq \text{MAX}_{\varphi|\mathcal{E}_k}(\text{MAX}_{\varphi|\mathcal{E}_{k-1}\ldots}(\text{MAX}_{\varphi|\mathcal{E}_1}(\Sigma_{\mathsf{ag}}))\ldots)$, for $1 \leq k \leq n$, and use the case $n = 2$ taking $\Sigma_{\mathsf{ag}} = \Sigma^{n-2}_{\mathsf{ag}}$ in the expression $\text{MAX}_{\varphi|\mathcal{E}_n}(\Sigma^{n-2}_{\mathsf{ag}}) \cap \text{MAX}_{\varphi|\mathcal{E}_{n-1}}(\Sigma^{n-2}_{\mathsf{ag}})$, then apply the inductive hypothesis to $\Sigma^{n-2}_{\mathsf{ag}}$. □

The second result shows that a solution to the previous process always produces at least one strategy:

Theorem 3. *Given* LTL *formulas* $\varphi, \mathcal{E}_1, \ldots, \mathcal{E}_n$ *satisfying* $\Sigma^{\mathcal{E}_i}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_{i+1}}_{\mathsf{env}}$ *for every* $i < n$*, the set* $\text{MAX}_{\varphi|\mathcal{E}_n}(\text{MAX}_{\varphi|\mathcal{E}_{n-1}\ldots}(\text{MAX}_{\varphi|\mathcal{E}_1}(\Sigma^{all}_{\mathsf{ag}}))\ldots)$ *is non-empty.*

At first glance this may seem obvious: by starting with a non-empty partially ordered set, and successively taking the maximal elements, are we not guaranteed to get a non-empty set? While this is clearly the case for finite sets, it is not the case in general for infinite sets (as here), where maximal elements may not exist. In fact, the proof of this theorem requires a sophisticated construction, which extends the results in (Berwanger 2007; Aminof et al. 2020).

Solving the Synthesis Problem Our main algorithmic result about synthesis is the following:

Theorem 4. *The best-effort synthesis under multiple environment specifications problem is* 2*EXPTIME-complete.*

The lower bound already holds for $n = 1, \mathcal{E} = \mathsf{true}$ (Aminof, De Giacomo, and Rubin 2021), i.e., a single environment specification and no assumption (and thus also for the case of chains where there is always a best-effort strategy). For the upper bound, we provide in section 5 an automata-theoretic approach that in fact shows how to build a tree-automaton whose language is the set of (encodings of) strategies that are best-effort for a given goal φ and environment specifications $\mathcal{E}_1, \cdots, \mathcal{E}_n$.

4 Examples

In this section we provide two planning examples.

Example 1 An agent's position is in the middle of an uphill snowy-route from Home to Destination, initially not wearing snow tires. The agent can move forward and backward and when at Home she can wear snow tires. Formally, we have the following fluents (i.e., variables under the control of the environment): $Pos \in \{1, \cdots, N\}$ and $SnowTiresOn$, and abbreviations $AtHome \doteq (Pos{=}0)$ and $AtDest \doteq (Pos{=}N)$. We also have the following actions (under the control of the agent): fwd for going forward one step; bwd for going backward one step; $wearSnowTires$ for wearing snow tires, which works only when $AtHome$. All these actions have no preconditions. The goal is $\varphi \doteq \mathsf{F}(AtDest)$.

Using these fluents and actions we consider three possible environments: $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ organized in a chain such that $\Sigma^{\mathcal{E}_1}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_2}_{\mathsf{env}} \subseteq \Sigma^{\mathcal{E}_3}_{\mathsf{env}}$. These environment share all the same initial state, which is the following (where k_{init} is some position in the middel of the route):

$$Pos{=}k_{init} \wedge \neg AtHome \wedge \neg AtDest \wedge \neg SnowTiresOn$$

Each of these environments reacts differently to the actions of the agent, i.e., specifies different effects, which become more and more indeterminate as we follow the chain.

Environment $\mathcal{E}_1$ is a simple deterministic planning domain, in which each action always works as expected, i.e., the action effects are:[1]

$\mathsf{G}(fwd \wedge \neg AtDest \wedge Pos{=}k \supset X(Pos{=}k{+}1))$
$\mathsf{G}(bwd \wedge \neg AtHome \wedge Pos{=}k \supset X(Pos{=}k{-}1))$
$\mathsf{G}(fwd \wedge AtDest \supset X(AtDest))$
$\mathsf{G}(bwd \wedge AtHome \supset X(AtHome))$
$\mathsf{G}(wearSnowTires \wedge AtHome \supset X(SnowTiresOn))$

In this environment there are several *winning* agent strategies for the goal. For example, one possible winning agent strategy is σ_1: "go forward till AtDest". Another possible winning strategy is σ_2 "go backward till AtHome, wear snow tires, go forward till AtDest". Yet another possible winning strategy is σ_3 "go forward as long as you don't slip" (a 'slip' is when the agent tries to move in some direction but its position does not change: $Pos{=}k \wedge (fwd \vee bwd) \wedge X(Pos{=}k)$) then follow σ_2. In fact σ_3 is one of the many other winning strategies that are also available, e.g., those that are a middle ground between the first two.

Environment $\mathcal{E}_2$ is analogous to $\mathcal{E}_1$, except that the actions fwd and bwd are now nondeterministic to capture that the road may be slippery because of snow.

$\mathsf{G}(fwd \wedge \neg AtDest \wedge \neg SnowTiresOn \wedge Pos{=}k \supset X(Pos{=}k{+}1 \vee Pos{=}k))$
$\mathsf{G}(fwd \wedge \neg AtDest \wedge SnowTiresOn \wedge Pos{=}k \supset X(Pos{=}k{+}1))$
$\mathsf{G}(fwd \wedge AtDest \supset X(AtDest))$
$\mathsf{G}(bwd \wedge \neg AtHome \wedge Pos{=}k \supset X(Pos{=}k{-}1 \vee Pos{=}k))$
$\mathsf{G}(bwd \wedge AtHome \supset X(AtHome))$
$\mathsf{G}(wearSnowTires \wedge AtHome \supset X(SnowTiresOn))$

Moreover $\mathcal{E}_2$ includes an additional stuck condition: "if forward and slip and forward again, then slip forever", i.e.,

$$\mathsf{G}((Pos{=}k \wedge fwd \wedge X((Pos{=}k) \wedge fwd)) \supset \mathsf{G}(Pos{=}k))$$

Finally, environment $\mathcal{E}_3$ is the same of $\mathcal{E}_2$, but without the "stuck condition" above, capturing not only that the road can become slippery, but also that the agent does not know if and when it may get stuck forever.

In $\mathcal{E}_2$ the agent has no winning strategies. However, σ_2 and σ_3 are now *best-effort*. Note, however, that σ_1 is not,

[1] For brevity we omit the formulas for the frame assumption, i.e., valuations of unmentioned fluents remain as before.

since after the agent learns that she slip while going forward, she then knows that going forward again will get her stuck. Also under $\mathcal{E}_3$ the agent has no winning strategies; however, all three strategies $\sigma_1, \sigma_2, \sigma_3$ are *best-effort*. In particular, σ_1 which was not best-effort under $\mathcal{E}_2$ is best-effort under $\mathcal{E}_3$, since we have more uncertainty of how the environment will react to the fwd action.

This example shows that, as expected, having a less determined environment specification (i.e., more environment strategies) makes it more difficult to have a winning strategy. Indeed, the agent has winning strategies only in $\mathcal{E}_1$, and only best-effort (but not winning) strategies in $\mathcal{E}_2$ and $\mathcal{E}_3$. Interestingly, it also shows that as the environment becomes less determinate, the agent can have more best-effort strategies. Indeed $\mathcal{E}_3$ admits more best-effort strategies than $\mathcal{E}_2$. Finally note that, our synthesis algorithm returns a strategy that is best-effort in all $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ (e.g., σ_2), and in fact winning in $\mathcal{E}_1$ where the goal is indeed enforceable.

Example 2 In this example every environment contributes towards the solution, and since the environment specifications form a chain one cannot consider only the first or last environment (i.e., equivalently, only the conjunction or the disjunction of all the environment specifications).

Consider a chemical plant that has three warehouses, primary, secondary, and tertiary (P, S, T, respectively). The same toxic chemical is stored in each wharehouses. The chemical is not flammable, except under certain rare conditions. The plant is equipped with a DRD (Disaster Response Drone). The DRD has a limited carrying capacity of 200 kg. One can load it with 200 kg of a chemical that can neutralizes up to 10 times its weight of the toxic chemical, or with both 100 kg of the neutralizer and 100 kg fire extinguisher. We will call these actions N and F, respectively.

The environment and the DRD interact as follows: the environment declares one of the warehouses P or S or T to have a chemical leak; the DRD is sent in response, either in configuration N or F; the environment decides how bad the situation has become by the time the DRD arrives; this consists of selecting how much leaked chemical has to be neutralized, and whether there is a fire or not; the DRD uses the chemicals it carries to (hopefully) handle the situation.

The goal of the DRD is to neutralize all the leaked chemicals and put out any fires. We now describe the three environment assumptions $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$.

$\mathcal{E}_1$: The first environment assumption is the most favourable. The warehouses have the regulation amounts of the toxic chemical: at most 2000 kg in primary, at most 1000 kg in secondary, at most 500 kg in tertiary. The circumstances needed for fire do not exist. Note that the only constraint on a best-effort strategy under $\mathcal{E}_1$ is for the agent to do N if the environment declares P.

$\mathcal{E}_2$: This is like assumption $\mathcal{E}_1$, but a fire (that can be handled with 100 kg of fire extinguisher) is also a possibility in primary or secondary (tertiary simply does not have the conditions needed for fire, e.g., no electricity is routed there). Note that the only constraint on a best-effort strategy under $\mathcal{E}_2$ is to do F if the environment declares S.

$\mathcal{E}_3$: This is like assumption $\mathcal{E}_2$, but somebody may have also broken the regulations and stored up to 2000 kg in the secondary or tertiary warehouses. Note that the only constraint on a best-effort strategy under this environment assumption is to do N if the environment declares T.

A best-effort strategy for this set of environment assumptions has to satisfy all of the stated constraints. Thus, the only best-effort strategy for the series of environment assumptions is that the agent does N if the environment declares P or T, and does F if the environment declares S.

5 Automata-theoretic Characterization

We provide an automata-theoretic characterization of the set of best-effort strategies for a single environment. We state the result for ease of reference (we will shortly formalize the terminology used in the statement):

Theorem 5. *One can, given* LTL *formulas φ and $\mathcal{E}$, compute in 2EXPTIME a nondeterministic parity tree-automaton (NPT) $\mathcal{A}_{\varphi,\mathcal{E}}$ of size 2EXP and index EXP whose language is* $\text{MAX}_{\varphi|\mathcal{E}}(\Sigma^{all}_{\mathsf{ag}})$.

In the rest of this section we show how to build the automaton from Theorem 5. It is worth noting that (Aminof et al. 2020) also shows how to build an automaton recognizing the same language. However, while that paper does not provide a complexity analysis, performing such an analysis shows that the approach in (Aminof et al. 2020) only achieves an automaton of size quadruple-exponential. We begin with a review of tree-automata (Section 5), then give a local characterizations of dominance and maximality and some useful lemmas (Section 5). We then give high-level sketch of the construction of the automaton (Section 5), its formal construction (Section 5) and, finally, a sketch of the correctness of this construction (Section 5).

Tree-Automata For sets Γ and D, a (full) D-branching Γ-labeled tree is a function $T : D^* \to \Gamma$. We use alternating tree-automata which are like classic finite-state automata that process infinite trees instead of finite words. Also, they incorporate both nondeterminism and its dual, universality, .

An *alternating parity tree automaton (APT)* $\mathcal{A}$ is a tuple $(D, \Gamma, Q, q_0, \delta, c)$ consisting of a finite set D of *directions*, a finite *input alphabet* Γ, a finite set Q of *states*, an *initial state* q_0, a *transition function* δ that maps elements in $Q \times \Gamma$ to positive Boolean formulas over $D \times Q$, and a *priority function* $c : Q \to \mathbb{N}$. A *run* of $\mathcal{A}$ on a D-branching Γ-labeled tree T is a $(D^* \times Q)$-labeled tree R whose domain is a prefix-free closed subset of $\mathbb{N}^*$ such that the root is labeled by (λ, q_0), and for each node v with $R(v) = (w, q)$ there is a satisfying assignment $C \subseteq D \times Q$ of $\delta(q, T(w))$ such that for every $(d', q') \in C$ there is a child v' of v such that $R(v') = (w \cdot d', q')$. Intuitively, an APT $\mathcal{A}$ takes a tree $T : D^* \to \Gamma$ as input and processes it starting at the root in the initial state; then, if $\mathcal{A}$ is at node $w \in D^*$ of T and the current state is $q \in Q$, it picks a satisfying assignment $C \subseteq D \times Q$ of $\delta(q, T(w))$ and, for each $(d', q') \in C$ sends a copy of itself in state q' in the direction d'. The run R is *accepting* if on every branch of R, the smallest priority seen infinitely often is even. The tree T is *accepted* by $\mathcal{A}$ if it

has an accepting run. The set of all trees accepted by $\mathcal{A}$ is called its *language*. A special case is the *co-Büchi* condition where certain states should not be seen infinitely often, i.e., the priorities are from the set $\{1, 2\}$.

Two automata are *equivalent* if their languages are equal. An APT is *nondeterministic* (resp. *universal*) if for every $(q, a) \in Q \times \Gamma$ there is a set S of $|D|$-tuples of states such that $\delta(q, a)$ is the disjunction (resp. conjunction) over all $(q_1, \cdots, q_{|D|}) \in S$ of the formulas $\bigwedge_{d \in D}(d, q_d)$. An APT in which every such S contains just a single tuple is called *deterministic*. These classes of automata are denoted NPT, UPT, and DPT, respectively.

APTs have robust closure properties (Muller and Schupp 1995; Löding 2021). We measure the cost of operations on an APT in terms of two parameters: its *size* is the cardinality of Q, and its *index* is the cardinality of $c(Q)$. APTs are closed under Boolean operations with no blowup; every APT can be converted into an equivalent NPT with an exponential blowup in size and polynomial blowup in index. If L is a language of D-branching $(\Gamma \times \Gamma')$-labeled trees, the *projection* of L onto Γ is the language of D-branching Γ-labeled trees defined as follows: the tree T is in the projection of L iff there exists a D-branching Γ'-labeled tree T' such that the tree $w \mapsto (T(w), T'(w))$ is in L. NPTs are closed under projection with a polynomial blowup in size and a polynomial blowup in index.

We also use deterministic (resp. universal) parity automata on words, denoted DPW (resp. UPW). We think of words as trees with $|D| = 1$, i.e., no branching.

Theorem 6. *(Vardi and Wolper 1994) Given an* LTL *formula φ, over the atomic propositions $AP \doteq 2^{X \cup Y}$, one can compute in EXPTIME a UCW, of size EXP, that accepts exactly the traces satisfying φ.*

A consequence of Theorem 5 is the upper bound in Theorem 4, i.e., a 2EXPTIME algorithm for synthesizing best-effort strategies under multiple environment specifications:

Proof of the upper bound in Theorem 4. Let l be the size of the largest input formula among $\varphi, \mathcal{E}1, \ldots \mathcal{E}_n$. For each $i \leq n$ construct, using Theorem 5, an NPT $\mathcal{A}_{\varphi, \mathcal{E}_i}$ for the language $\text{MAX}_{\varphi|\mathcal{E}_i}(\Sigma^{all}_{\mathsf{ag}})$ of size (resp. index) at most 2EXP (resp. EXP) in l. For two NPTs of sizes s_1, s_2 and indexes k_1, k_2, one can build an NPT for the intersection of their languages of size $s_1 s_2 \frac{(k_1+k_2)!}{k_1! k_2!}$ and index $(k_1 + k_2)$ (Chatterjee, Henzinger, and Piterman 2010). Thus, we can build an NPT $\mathcal{A}$ for the intersection $\bigcap_{i=1}^{n} \text{MAX}_{\varphi|\mathcal{E}_n}(\Sigma^{all}_{\mathsf{ag}})$ of size 2EXP in l and EXP in n, and index polynomial in n and EXP in l. An NPT can be tested for non-emptiness — and if non-empty a finite-state representation of a tree in its language can be extracted — in time polynomial in its size and EXP in its index. Thus, the whole procedure is 2EXP in l and EXP in n, and thus at most 2EXP in the size of the input. □

Characterizing Dominance and Maximality We use local characterizations of dominance and maximality based on assigning values to histories (Berwanger 2007; Aminof et al. 2020). The value of a history h, and an agent strategy σ_{ag} consistent with h, is defined iff h is a prefix of some play $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})$ where $\sigma_{\mathsf{env}} \in \Sigma^{\mathcal{E}}_{\mathsf{env}}$ is an environment strategy enforcing $\mathcal{E}$. The value is: 1 (aka "winning") if φ holds for all such plays; -1 (aka "losing") if $\neg\varphi$ holds for all such plays; and 0 (aka "pending") otherwise. Thus, the value captures how well a given agent strategy does (for achieving the goal φ), starting at a given history h, against environment strategies in $\Sigma^{\mathcal{E},h}_{\mathsf{env}}$.

More formally, let $H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$ be the set of histories h that end in an environment move, are consistent with the agent strategy σ_{ag}, and for which $\Sigma^{\mathcal{E},h}_{\mathsf{env}} \neq \emptyset$. Then:

Definition 4. *For a goal φ, an environment spec $\mathcal{E}$, agent strategy σ_{ag}, and history $h \in H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$, define $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h)$ as follows:*

- $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) := 1$ *(winning) if* $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \varphi$ *for every* $\sigma_{\mathsf{env}} \in \Sigma^{\mathcal{E},h}_{\mathsf{env}}$;
- $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) := -1$ *(losing) if* $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \neg\varphi$ *for every* $\sigma_{\mathsf{env}} \in \Sigma^{\mathcal{E},h}_{\mathsf{env}}$;
- $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) := 0$ *(pending) otherwise.*

We can compare two agent strategies by looking at those histories at which they make a different decision, i.e., "split". Formally, two agent strategies σ_1, σ_2 *split* at a history h if h ends in an environment move, is consistent with σ_1 and with σ_2, and $\sigma_1(h) \neq \sigma_2(h)$. The following proposition characterizes dominance by comparing the values of agent strategies at their split points.

Proposition 1 (Characterization of Dominance)**.** *Given agent strategies σ_1, σ_2, we have that $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$ iff for every history h, such that $\Sigma^{\mathcal{E},h}_{\mathsf{env}} \neq \emptyset$, at which σ_1, σ_2 split: (i) $val_{\varphi|\mathcal{E}}(\sigma_1, h) \geq val_{\varphi|\mathcal{E}}(\sigma_2, h)$, and (ii) it is not the case that $val_{\varphi|\mathcal{E}}(\sigma_1, h) = val_{\varphi|\mathcal{E}}(\sigma_2, h) = 0$. Furthermore, $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$ iff, in addition, (iii) for some h at which σ_1, σ_2 split, $val_{\varphi|\mathcal{E}}(\sigma_1, h) > val_{\varphi|\mathcal{E}}(\sigma_2, h)$.*

The next definition assigns values to histories alone:

Definition 5. *For a history h ending in an environment move, define $val_{\varphi|\mathcal{E}}(h)$ as the maximum of $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h)$, where σ_{ag} varies over all agent-strategies for which $h \in H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$; if there are no such agent strategies then write $val_{\varphi|\mathcal{E}}(h) = und$ (which stands for "undefined").*

The following proposition follows from Proposition 1, and characterizes maximal strategies.

Proposition 2. *A strategy σ_{ag} is best-effort for φ under $\mathcal{E}$ iff $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) = val_{\varphi|\mathcal{E}}(h)$ for every history $h \in H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$.*

Our automata will be guessing strategies that witness properties of many histories at once. For this we need some supporting lemmas.

Lemma 4. *Given* LTL *formulas $\varphi, \mathcal{E}$, and an agent strategy σ_{ag}, there is an environment strategy σ_{env} that enforces $\mathcal{E}$ where possible and, for every $h \in H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$ for which $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) = 0$, the $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ extension of h satisfies φ, and is consistent with some strategy in $\Sigma^{\mathcal{E}}_{\mathsf{env}}$.*

Proof. The proof is essentially the same as that of Lemma 3 by taking, for every $h \in H_{\mathcal{E}}(\sigma_{\mathsf{ag}})$ for which $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) = 0$, the strategy σ^h_{env} to be a strategy in $\Sigma^{\mathcal{E}}_{\mathsf{env}}$ satisfying $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma^h_{\mathsf{env}}) \models \varphi$. □

The next Lemma characterizes non-winning histories in terms of environment strategies.

Lemma 5. *Given* LTL *formulas* $\varphi, \mathcal{E}$ *and a history* h *such that* $\Sigma_{\mathsf{env}}^{\mathcal{E},h} \neq \emptyset$*:* $val_{\varphi|\mathcal{E}}(h) \neq 1$ *iff the environment can enforce* $\mathcal{E} \wedge \neg\varphi$ *from* h*.*

Proof. Given an agent strategy σ_{ag}, it is clear from the definitions that $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) \neq 1$ iff there is $\sigma_{\mathsf{env}} \in \Sigma_{\mathsf{env}}^{\mathcal{E},h}$ such that $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \neg\varphi$. The fact that a single σ_{env} can witness this for all σ_{ag} follows from the determinacy of the corresponding game.

More formally, let $\mathcal{E}' \doteq h_X \wedge \mathcal{E}$ where h_X is an LTL formula saying that the environment's first move is h_0, and, for odd $i < |h| - 1$, if the agent played h_i then the environment responds with h_{i+1}. Similarly, let $\varphi' \doteq h_Y \wedge \varphi$ where h_Y is an LTL formula expressing that, for even $i < |h| - 1$, if the environment plays h_i then the agent responds with h_{i+1}. Intuitively, $\mathcal{E}'$ and φ' add to $\mathcal{E}$ and φ the assurance that the environment/agent will follow h as long as the other does.

The lemma follows from the following chain of equivalences. The environment can enforce $\mathcal{E} \wedge \neg\varphi$ from h iff it can enforce $\mathcal{E}' \wedge \neg\varphi'$ (we will show this below), iff the agent cannot enforce $\mathcal{E}' \supset \varphi'$ (by determinacy of the corresponding game (Apt and Grädel 2011)), iff there is no agent strategy σ_{ag} for which $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \varphi'$ for all $\sigma_{\mathsf{env}} \in \Sigma_{\mathsf{env}}^{\mathcal{E}'}$ (by Theorem 9 in (Aminof et al. 2019a)), iff for every σ_{ag} consistent with h there is $\sigma_{\mathsf{env}} \in \Sigma_{\mathsf{env}}^{\mathcal{E},h}$ such that $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}) \models \neg\varphi$, in other words, iff $val_{\varphi|\mathcal{E}}(\sigma_{\mathsf{ag}}, h) \neq 1$.

We now prove the first equivalence in the chain above. Note that (†): $\mathcal{E}' \wedge \neg\varphi' = h_X \wedge \mathcal{E} \wedge (\neg h_Y \vee \neg\varphi)$. By the assumption of the Lemma, take some strategy $\sigma_h \in \Sigma_{\mathsf{env}}^{\mathcal{E},h}$. For the $\Rightarrow$ direction, suppose σ_{env} enforces $\mathcal{E} \wedge \neg\varphi$ from h. We will show that $\sigma'_{\mathsf{env}} \doteq \sigma_h[h \leftarrow \sigma_{\mathsf{env}}]$ enforces $\mathcal{E}' \wedge \neg\varphi'$. Take some play π consistent with σ'_{env}. By our choice of σ_h, we have that h is consistent with σ'_{env}. Thus, if the agent proceeds along h so does the environment. Hence, h_X holds on π. If π does not extend h then (i) $\neg h_Y$ holds and (ii) σ'_{env} and σ_h agree on π. Thus, by our choice of σ_h (i.e., that it enforces $\mathcal{E}$), we have that π satisfies $\mathcal{E}$. On the other hand, if π extends h then $\pi \models \mathcal{E} \wedge \neg\varphi$ since σ_{env} enforces $\mathcal{E} \wedge \neg\varphi$. Thus, in both cases, by (†), $\pi \models \mathcal{E}' \wedge \neg\varphi'$.

For the $\Leftarrow$ direction, suppose σ_{env} enforces $\mathcal{E}' \wedge \neg\varphi'$. We show that σ_{env} enforces $\mathcal{E} \wedge \neg\varphi$ from h. Note that σ_{env} in particular enforces h_X. Hence, h is consistent with σ_{env}, and for every σ_{ag}, the $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ extension of h, call it π, is consistent with σ_{env}. Hence, in particular, $\pi \models \mathcal{E}$. Since π extends h it also satisfies h_Y and thus, $\pi \models \neg\varphi$ by (†). □

High-level Description of the Automaton $\mathcal{A}_{\varphi,\mathcal{E}}$ The automaton $\mathcal{A}_{\varphi,\mathcal{E}}$ in Theorem 5 will accept an input strategy σ_{ag} iff σ_{ag} satisfies the characterization of maximality in Proposition 2. To achieve the size and index stated in Theorem 5, we use a sophisticated construction. In what follows, since $\varphi, \mathcal{E}$ are fixed, we may drop them and write $val(-)$ instead of $val_{\varphi|\mathcal{E}}(-)$ and $H(-)$ instead of $H_{\mathcal{E}}(-)$.

The automaton will guess, using its nondeterminism, a value $guess(h) \in \{-1, 0, 1\}$ for every history h of odd length consistent with σ_{ag} for which it believes $val(h) \neq und$; it will then verify that if $val(h) \neq und$ then $guess(h) = val(\sigma_{\mathsf{ag}}, h) = val(h)$. By Proposition 2, the input strategy σ_{ag} is maximal iff the automaton can successfully guess and verify these values. The automaton will make no attempt to correctly guess, nor verify, that $val(h) = und$. Indeed, if $val(h) = und$ then there are no requirements regarding h that σ_{ag} should satisfy. Thus, if it guessed (and "verified") by mistake a value $guess(h) \in \{-1, 0, 1\}$, it can only make the automaton do unnecessary work, but it cannot make it accept a non-maximal strategy.

To help it correctly verify the guessed values, the automaton will employ the characterization in Lemma 2. In particular, it will optionally guess for a history h a flag $\neg\mathcal{E}$ indicating that the agent can enforce $\neg\mathcal{E}$ from h. This guess can be verified by guessing a corresponding agent strategy $\sigma_h^{\neg\mathcal{E}}$ and checking that it enforces $\neg\mathcal{E}$ from h. By Lemma 3, the automaton will actually guess a single strategy $\sigma_{\mathsf{ag}}^{\neg\mathcal{E}}$ that will serve as $\sigma_h^{\neg\mathcal{E}}$ for all such h.

Observe that if $val(\sigma_{\mathsf{ag}}, h) = 1$ then for every extension h' of h, consistent with σ_{ag}, we have that $val(\sigma_{\mathsf{ag}}, h') = 1$, or is undefined; and if $val(h) = -1$ then $val(h') \in \{-1, und\}$ for every extension h' of h. Thus, once the automaton verifies that $val(\sigma_{\mathsf{ag}}, h) = val(h) \in \{-1, 1\}$ it does not need to verify anything for any extension h' of h in order to prove that σ_{ag} is maximal. Similarly, if the agent can enforce $\neg\mathcal{E}$ from h then [2], by Lemma 2, $val(h) = und$, and thus $val(h') = und$ for all extensions h' of h. Hence, the automaton does not have to guess or verify anything for histories extending ones for which it guessed $\neg\mathcal{E}$.

In order to verify that if $val(h) \neq und$ then $guess(h)$ is correct, the automaton will verify two things:

(a) $guess(h) \leq val(\sigma_{\mathsf{ag}}, h)$, i.e., the guessed value is no larger than the correct value of the input strategy at h;

(b) $val(h) \leq guess(h)$, i.e., no agent strategy consistent with h achieves a larger value than the guessed value.

Since obviously $val(\sigma_{\mathsf{ag}}, h) \leq val(h)$, this shows that $guess(h) = val(\sigma_{\mathsf{ag}}, h) = val(h)$, as claimed.

If the guessed value is 1*,* then (since it is the highest possible value) it is enough to verify (a). I.e., that every trace extending h that is consistent with σ_{ag} and some strategy in $\Sigma_{\mathsf{env}}^{\mathcal{E}}$ satisfies φ. By Lemma 2, this can be done by showing that every trace τ consistent with σ_{ag} extending h, but that does not extend any history h' from which the agent can enforce $\neg\mathcal{E}$, satisfies $\mathcal{E} \supset \varphi$. Thus, the automaton checks that every trace τ consistent with σ_{ag} extending h, but not extending an h' for which $\neg\mathcal{E}$ was guessed, satisfies $\mathcal{E} \supset \varphi$.

If the guessed value is -1*,* then (since this is the lowest possible value) it is enough to verify (b). I.e., that every trace extending h, that is consistent with some strategy in $\Sigma_{\mathsf{env}}^{\mathcal{E}}$, satisfies $\neg\varphi$. By Lemma 2, this is the case iff every trace extending h, but not extending any h' from which the agent can enforce $\neg\mathcal{E}$, satisfies $\mathcal{E} \supset \neg\varphi$. Thus, the automaton checks that every trace extending h, but not extending an h' for which $\neg\mathcal{E}$ was guessed, satisfies $\mathcal{E} \supset \neg\varphi$.

[2] One can actually show that for histories that are minimal in the prefix order this condition is also necessary, i.e., $val(h) \neq und$ iff the environment can enforce $\mathcal{E}$ from every prefix of h.

If the guessed value is 0*,* the automaton has to verify both (a) and (b). To verify (a), by definition, it has to find $\sigma_h \in \Sigma^{\mathcal{E},h}_{\mathsf{env}}$, such that $\text{PLAY}(\sigma_{\mathsf{ag}}, \sigma_h) \models \varphi$. By Lemma 1, instead of guessing σ_h, the automaton can guess a strategy σ^{env}_h that enforces $\mathcal{E}$ from h and for which the $\sigma_{\mathsf{ag}}, \sigma^h_{\mathsf{env}}$ extension of h satisfies φ. To do this for all such h, the automaton will actually guess, by Lemma 4, a single strategy σ_{env}. To verify (b), by definition, one has to show that no agent strategy consistent with h achieves the value 1 on h. By Lemma 5, this can be done by guessing a strategy $\sigma^{\mathcal{E}\wedge\neg\varphi}_h$ and checking that it enforces $\mathcal{E} \wedge \neg\varphi$ from h. By Lemma 3, a single strategy $\sigma^{\mathcal{E}\wedge\neg\varphi}_{\mathsf{env}}$ can serve as $\sigma^{\mathcal{E}\wedge\neg\varphi}_h$ for all such h.

Detailed Construction of the Automaton $\mathcal{A}_{\varphi,\mathcal{E}}$ The construction of the NPT $\mathcal{A}_{\varphi,\mathcal{E}}$ in Theorem 5 will proceed in four steps.

1. We build a UPT $U_{\varphi,\mathcal{E}}$ that instead of making all the guesses described above (in Section 5), expects these guesses to be written on its input trees. Thus, $U_{\varphi,\mathcal{E}}$ will only be concerned with the task of verifying that the guesses written on the input tree are correct.
2. We build an APT $\mathcal{A}$ that universally launches, at the root of an input tree, both $U_{\varphi,\mathcal{E}}$ and a constant size APT $\mathcal{A}_{legal}$ that is charged with the simple task of verifying that the annotation of the input tree with all these guesses is done in a legal way (e.g., that every strategy written on the tree specifies exactly one move for each of its input histories).
3. We convert the APT $\mathcal{A}$ to an equivalent NPT $\mathcal{A}'$.
4. We obtain $\mathcal{A}_{\varphi,\mathcal{E}}$ by "projecting out" all the extraneous annotations on the input trees. That is, unlike $\mathcal{A}'$ that expects a tree labeled with all the guesses of strategies, values, etc., $\mathcal{A}_{\varphi,\mathcal{E}}$ expects the tree labeled only by the agent strategy σ_{ag}, and guesses everything else on its own.

We give a brief analysis of size and index of the resulting NPT $\mathcal{A}_{\varphi,\mathcal{E}}$. We will see that the UPT $U_{\varphi,\mathcal{E}}$ has size EXP and constant index. Thus, so does the APT $\mathcal{A}$ (since it is the product of $U_{\varphi,\mathcal{E}}$ with the constant sized $\mathcal{A}_{legal}$). Thus, the NPT $\mathcal{A}'$ has size 2EXP and index EXP, and so does its projection $\mathcal{A}_{\varphi,\mathcal{E}}$ — as stated in Theorem 5.

We now focus on the encoding and annotations of the input trees and the construction of $U_{\varphi,\mathcal{E}}$.

We assume w.l.o.g. that the sets X, Y of agent and environment variables are of the same size n (otherwise we pad with dummy variables). Thus, the input trees will be $|2^n|$-branching, where the empty history corresponds to the root, a history $X_0 \cdot Y_0 \cdots, X_k \cdot Y_k$ corresponds to a node of depth $2k$, and a history $X_0 \cdot Y_0 \cdots X_k \cdot Y_k \cdot X_{k+1}$ corresponds to a node of depth $2k+1$. Nodes at even (resp. odd) depth are called *environment nodes* (resp. *agent nodes*) since it is that player's turn to move. From now on we will not distinguish between histories and nodes. We label such a tree by an agent (resp. environment) strategy σ, as follows: if h is an agent (resp. environment) node then mark with the symbol σ its child node $h \cdot \sigma(h)$. Note that agent strategies mark environment nodes, and vice versa.

The input trees of $U_{\varphi,\mathcal{E}}$ are labeled/marked with moves of two agent strategies σ_{ag} and $\sigma^{\neg\mathcal{E}}_{\mathsf{ag}}$, and two environment strategies σ_{env} and $\sigma^{\mathcal{E}\wedge\neg\varphi}_{\mathsf{env}}$. In addition, an agent node consistent with σ_{ag} (i.e., whose parent is the root or is marked by σ_{ag}) is optionally marked by a number in $\{-1, 0, 1\}$, and any node may be optionally marked by $\neg\mathcal{E}$. Finally, every environment node $h = X_0 \cdot Y_0 \cdots X_k \cdot Y_k$ is marked by the atomic proposition X_k, Y_k, i.e., the last move of the agent and of the environment leading up to h (these atomic propositions are read by the UCW's making up $U_{\varphi,\mathcal{E}}$ in order to verify that certain traces satisfy some LTL formulas).

A marking of a tree is *legal* iff it satisfies the following: for the agent strategies $\sigma \in \{\sigma_{\mathsf{ag}}, \sigma^{\neg\mathcal{E}}_{\mathsf{ag}}\}$ (resp. environment strategies $\sigma \in \{\sigma_{\mathsf{env}}, \sigma^{\mathcal{E}\wedge\neg\varphi}_{\mathsf{env}}\}$) marking the tree, agent (resp. environment) nodes are not labeled by σ, and every agent (resp. env) node has a unique child labeled by σ; the atomic propositions are marked on environment nodes and are equal to the last two directions leading to this node; a node is marked by a number iff it is an agent node consistent with σ_{ag} that is not marked by $\neg\mathcal{E}$ and none of its ancestors is marked by 1, -1 or $\neg\mathcal{E}$. A node can be marked by $\neg\mathcal{E}$ only if none of its ancestors is marked by $\neg\mathcal{E}$.

Observe that one can easily build an APT $\mathcal{A}_{legal}$ with a constant number of states that accepts a tree iff it is legal.

Intuitively, the UPT $U_{\varphi,\mathcal{E}}$ will be constructed in such a way that it accepts a (legally marked) tree iff $(\dagger)$:

$(\dagger i)$ $\sigma^{\neg\mathcal{E}}_{\mathsf{ag}}$ enforces $\neg\mathcal{E}$ from every node marked with $\neg\mathcal{E}$;

$(\dagger ii)$ σ_{env} enforces $\mathcal{E}$ from every node marked with 0;

$(\dagger iii)$ $\sigma^{\mathcal{E}\wedge\neg\varphi}_{\mathsf{env}}$ enforces $\mathcal{E} \wedge \neg\varphi$ from every node marked 0;

$(\dagger iv)$ if a node h is marked 0 then the $\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}}$ extension of h satisfies φ;

$(\dagger v)$ $\sigma_{\mathsf{ag}} \in \text{MAX}_{\varphi|\mathcal{E}}(\Sigma^{all}_{\mathsf{ag}})$.

The UPT $U_{\varphi,\mathcal{E}}$ is made up of 6 component UPW's, i.e., U_ψ for $\psi \in \{\varphi, \mathcal{E}, \neg\mathcal{E}, \mathcal{E} \supset \varphi, \mathcal{E} \supset \neg\varphi, \mathcal{E} \wedge \neg\varphi\}$. Informally, the component $U_{\neg\mathcal{E}}$ verifies that the agent can enforce $\neg\mathcal{E}$ from nodes marked $\neg\mathcal{E}$; the component $U_{\mathcal{E}\supset\varphi}$ is used in the verification of the correctness of the 1 markings; U_φ together with $U_{\mathcal{E}}$ verify that the value of a node h marked by 0 is not actually -1, and $U_{\mathcal{E}\wedge\neg\varphi}$ verifies that no other agent strategy achieves a higher value for h than σ_{ag}; finally, $U_{\mathcal{E}\supset\neg\varphi}$ is used in the verification of -1 markings.

For $\psi \in \{\mathcal{E} \supset \varphi, \mathcal{E} \supset \neg\varphi, \varphi, \mathcal{E} \wedge \neg\varphi, \mathcal{E}, \neg\mathcal{E}\}$, the UPW U_ψ is built by applying Theorem 6 to get a UCW $\mathcal{A}_\psi$ for ψ, and modifying this UCW by adding to every state a Boolean flag $\top, \bot$, designating 'active' or 'inactive', respectively. The transition relation of U_ψ is identical to that of $\mathcal{A}_\psi$ except that the flag may also be flipped when reading certain inputs, as will be described later. Finally, the (parity condition) priority function of U_ψ assigns the priority 0 to all states with the flag $\bot$, the priority 1 to the co-Büchi states of $\mathcal{A}_\psi$ with a $\top$ flag, and the priority 2 to the remaining states. Thus, a run of a copy of U_ψ rejects a trace iff from some point on the flag is $\top$ and the sequence of atomic propositions labeled on that trace does not satisfy ψ. At the beginning, the flag is $\bot$, it is changed according to the following rules:

1. $U_{\mathcal{E}\supset\varphi}$ sets the flag to $\top$ when reading a node marked 1;
2. $U_\varphi, U_{\mathcal{E}}$ set the flag to $\top$ when reading a node marked 0, and reset it to $\bot$ when reading an agent node unmarked

by σ_{env};

3. $U_{\mathcal{E}\wedge\neg\varphi}$ sets the flag to $\top$ when reading a node marked 0, and resets it to $\bot$ when reading an agent node unmarked by $\sigma_{\mathsf{env}}^{\mathcal{E}\wedge\neg\varphi}$;
4. $U_{\mathcal{E}\supset\neg\varphi}$ sets the flag to $\top$ when reading a node marked -1;
5. $U_{\neg\mathcal{E}}$ sets the flag to $\top$ when reading a node marked $\neg\mathcal{E}$.

The automaton $U_{\varphi,\mathcal{E}}$ launches all of its 6 component UPW's in a universal mode at the root. While reading a node h, a copy of each of its component UCWs sends copies to all or none of the sons of h, as follows:

1. $U_{\mathcal{E}}$, and $U_{\mathcal{E}\wedge\neg\varphi}$ always send copies to all sons;
2. $U_{\mathcal{E}\supset\varphi}$ sends no copies if h is inconsistent with σ_{ag}, or marked by $\neg\mathcal{E}$;
3. U_{φ} sends no copies if h is inconsistent with σ_{ag};
4. $U_{\mathcal{E}\supset\neg\varphi}$ sends no copies if h is marked by $\neg\mathcal{E}$;
5. $U_{\neg\mathcal{E}}$ sends no copies when active and h is an environment node unmarked by $\sigma_{\mathsf{ag}}^{\neg\mathcal{E}}$ (i.e., once active it follows $\sigma_{\mathsf{ag}}^{\neg\mathcal{E}}$).

This completes the description of $U_{\varphi,\mathcal{E}}$.

Note that each UCW $\mathcal{A}_{\psi}$ has size EXP and a constant index (by Theorem 6) and thus, so does the UCW U_{ψ}, as well as the UPW $U_{\varphi,\mathcal{E}}$ (since it is a product of a constant number of these UCW's).

Correctness of the Construction of the Automaton $\mathcal{A}_{\varphi,\mathcal{E}}$ The statement of correctness is as follows: an agent strategy σ_{ag} is in $\textsc{max}_{\varphi|\mathcal{E}}(\Sigma_{\mathsf{ag}}^{all})$ iff there exists a legally marked tree with σ_{ag} written on it which is accepted by $U_{\varphi,\mathcal{E}}$. This, in turn, is based on the following relatively easy observations about the run of $U_{\varphi,\mathcal{E}}$ on a tree. Assuming that the tree is legally marked, then:

1. All copies of $U_{\neg\mathcal{E}}$ accept iff $\dagger(i)$ holds.
2. All copies of $U_{\mathcal{E}}$ accept iff $\dagger(ii)$ holds.
3. All copies of $U_{\mathcal{E}\wedge\neg\varphi}$ accept iff $\dagger(iii)$ holds.
4. All copies of U_{φ} accept iff $\dagger(iv)$ holds.
5. All copies of $U_{\mathcal{E}\supset\varphi}$ accept iff $\forall h$ with $guess(h) = 1$, all traces consistent with σ_{ag} extending h (that do not also extend a node marked by $\neg\mathcal{E}$) satisfy $\mathcal{E}\supset\varphi$.
6. All copies of $U_{\mathcal{E}\supset\neg\varphi}$ accept iff $\forall h$ with $guess(h) = -1$, all traces extending it (that do not extend a history marked by $\neg\mathcal{E}$) satisfy $\mathcal{E}\supset\neg\varphi$.

6 Related Work

In classical game-theory, it is argued that a rational player would not choose a strictly dominated strategy. This idea was introduced to games on graphs in (Berwanger 2007) which provides the local characterisation of best-effort, and studies properties of the set of strategies that survive iterated deletion of dominated strategies, so called "iteratively admissible strategies"; finer algorithmic results in the context of model-checking were studied in (Brenguier, Raskin, and Sassolas 2014) for graph-games with ω-regular objectives.

(Aminof et al. 2020) studies the problem of best-effort synthesis under two environments (linearly ordered expected and exceptional with the assumption that the goal is enforceable under the expected environment), and show it to be decidable; an analysis of their algorithm shows it is in 4EXPTIME. Theorem 5 provides an optimal 2EXPTIME algorith also for this special case.

Best-effort strategies have been studied in the game-graphs literature. (Faella 2009) argues for best-effort strategies as reasonable responses in environments that may not be adversarial, characterizes goals admitting positional best-effort strategies, and shows how to compute such strategies with no environment specifications. (Aminof, De Giacomo, and Rubin 2021) provide an optimal and simple graph-based algorithm for computing a best-effort strategy for an agent under a single assumption. Other works assume that players always use best-effort strategies, e.g., (Brenguier, Raskin, and Sankur 2017) introduce assume-admissible (AA) synthesis which is a compositional approach to synthesis.

Synthesis of dominant (i.e., maximum, rather than maximal) strategies have been studied in the trace-based setting for game-graphs (Damm and Finkbeiner 2011) and for distributed reactive-synthesis (Damm and Finkbeiner 2014; Finkbeiner and Passing 2020).

Apart from (Aminof et al. 2020), multiple environments have been considered in a number of other synthesis/planning contexts. (Ciolek et al. 2020) studies planning in multiple environments consisting of nondeterministic fully observable planning domains (FOND). They only consider winning (not best-effort) policies, and propose that as the environment becomes more indeterminate also the goal should become less demanding. (Finkbeiner and Gölz 2018) introduces the synthesis problem in distributed environments, i.e., it is the environment (not the system) that consists of several components with different knowledge/information. They formulate the problem as a Petri-game with one system token and a bounded number of environment tokens, and show the problem to be EXPTIME-complete. Also generalized planning can be seen as a form of synthesis under multiple assumption (Srivastava 2011; Hu and De Giacomo 2011; De Giacomo et al. 2016). In the language of this paper, this is the problem of finding a strategy that enforces the goal in multiple environments which share the same observables and available agent actions but not the same internal structure. Such environments are typically specified as deterministic planning domains. In our case, the multiple environments are fully observable and share the same set of fluents and actions, although they are specified in LTL (hence may be non-Markovian) instead of planning domains (which are Markovian). Moreover, we allow for best-effort solutions.

Many complex synthesis problems can been expressed in a number of logics for strategic reasoning, notably in Strategy Logic and its variations (Mogavero et al. 2014; Berthon et al. 2021; Belardinelli et al. 2020; Aminof et al. 2019b). These allow one to quantify over strategies, and thus can naturally express complex strategic properties, including dominance and maximality. However, doing so does not seem to provide optimal algorithms, as we do here.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), and by the Austrian Science Fund (FWF): P 32021. Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies.

References

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2018. Synthesis under assumptions. In *KR*.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019a. Planning under LTL environment specifications. In *ICAPS*.

Aminof, B.; Kwiatkowska, M.; Maubert, B.; Murano, A.; and Rubin, S. 2019b. Probabilistic strategy logic. In *IJCAI*.

Aminof, B.; De Giacomo, G.; Lomuscio, A.; Murano, A.; and Rubin, S. 2020. Synthesizing strategies under expected and exceptional environment behaviors. In *IJCAI*.

Aminof, B.; De Giacomo, G.; and Rubin, S. 2021. Best-effort synthesis: Doing your best is not harder than giving up. In *IJCAI*.

Apt, K., and Grädel, E. 2011. *Lectures in game theory for computer scientists*. Cambridge.

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *AAAI*.

Belardinelli, F.; Lomuscio, A.; Murano, A.; and Rubin, S. 2020. Verification of multi-agent systems with public actions against strategy logic. *Artificial Intelligence* 285:103302.

Berthon, R.; Maubert, B.; Murano, A.; Rubin, S.; and Vardi, M. Y. 2021. Strategy logic with imperfect information. *ACM Transactions on Computational Logic (TOCL)* 22(1):1–51.

Berwanger, D. 2007. Admissibility in infinite games. In *STACS*.

Brenguier, R.; Raskin, J.; and Sankur, O. 2017. Assume-admissible synthesis. *Acta Inf.* 54(1).

Brenguier, R.; Raskin, J.; and Sassolas, M. 2014. The complexity of admissibility in omega-regular games. In *CSL-LICS*.

Camacho, A.; Bienvenu, M.; and McIlraith, S. 2018. Finite LTL synthesis with environment assumptions and quality measures. In *KR*.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*.

Chatterjee, K.; Henzinger, T. A.; and Piterman, N. 2010. Strategy logic. *Information and Computation* 208(6):677–693.

Ciolek, D. A.; D'Ippolito, N.; Pozanco, A.; and Sardiña, S. 2020. Multi-tier automated planning for adaptive behavior. In *ICAPS*.

Damm, W., and Finkbeiner, B. 2011. Does it pay to extend the perimeter of a world model? In *FM*.

Damm, W., and Finkbeiner, B. 2014. Automatic compositional synthesis of distributed systems. In *FM*.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDLf goals. In *IJCAI*.

De Giacomo, G.; Di Stasio, A.; Murano, A.; and Rubin, S. 2016. Imperfect-information games and generalized planning. In *IJCAI*.

D'Ippolito, N.; Rodríguez, N.; and Sardiña, S. 2018. Fully observable non-deterministic planning as assumption-based reactive synthesis. *J. Artif. Intell. Res.* 61.

Faella, M. 2009. Admissible strategies in infinite games over graphs. In *MFCS*.

Finkbeiner, B., and Gölz, P. 2018. Synthesis in distributed environments. In *FSTTCS*.

Finkbeiner, B., and Passing, N. 2020. Dependency-based compositional synthesis. In *ATVA*.

Hu, Y., and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*.

Löding, C. 2021. Automata on infinite trees. In Pin, J.-É., ed., *Handbook of Automata Theory*. EMS Press, Berlin. To appear.

Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2014. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.* 15(4):34:1–34:47.

Muller, D. E., and Schupp, P. E. 1995. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science* 141(1-2):69–107.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *POPL*.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*.

Srivastava, S. 2011. Foundations and applications of generalized planning. *AI Commun.* 24(4):349–351.

Vardi, M. Y., and Wolper, P. 1994. Reasoning about infinite computations. *Inf. Comput.* 115(1):1–37.