Effective Approach to LTL_f Best-Effort Synthesis in Multi-Tier Environments

Benjamin Aminof[1] , **Giuseppe De Giacomo**[1,2] , **Gianmarco Parretti**[1] and **Sasha Rubin**[3]
[1]Università degli Studi di Roma "La Sapienza"
[2]University of Oxford
[3]University of Sydney
aminof@forsyte.at, parretti@diag.uniroma1.it, giuseppe.degiacomo@cs.ox.ac.uk, sasha.rubin@sydney.edu.au

Abstract

We consider an agent acting in a complex environment modeled through a multi-tiered specification, in which each tier adds nondeterminism in the environment response to the agent actions. In this setting, we devise an effective approach to best-effort synthesis, i.e., synthesizing agent strategies that win against a maximal set of possible environment responses in each tier. We do this in a setting where both the multi-tier environment and agent goal are specified in the linear temporal logic on finite traces (LTL_f). While theoretical solution techniques based on automata on infinite trees have been developed previously, we completely side-step them here and focus on a DFA-based game-theoretic technique, which can be effectively implemented symbolically. Specifically, we present a provably correct algorithm that is based on solving separately DFA-based games for each tier and then combining the obtained solutions on-the-fly. This algorithm is linear, as opposed to being exponential, in the number of tiers, and thus, it can graciously handle multi-tier environments formed of several tiers.

1 Introduction

There is a growing interest in Reasoning about Actions, Planning, and Sequential Decision Making on developing autonomous AI systems that can operate effectively in complex and dynamic environments where the level of nondeterminism is high. We typically assume that the AI system, i.e., the *agent*, has a single or *flat* model of the environment (specified, e.g., in the Situation Calculus [Reiter, 2001], or in PDDL [Haslum *et al.*, 2019], or in Temporal Logics [Aminof *et al.*, 2018; Camacho *et al.*, 2019; Aminof *et al.*, 2019]), which the agent uses to deliberate how to achieve its goals. However, accurately modeling such environments can be challenging, particularly when there is a high degree of uncertainty. Hence, the scientific community has been exploring the concept of *multi-tier* models of environment behavior, i.e., having simultaneously several models, or *tiers*, of the environment such that, in each tier the environment is more nondeterministic than in the previous one [Aminof *et al.*, 2020; Ciolek *et al.*, 2020; Aminof *et al.*, 2021a]. For example, an agent may have a tier that represents the expected environment behavior, but also other tiers that represent increasingly nondeterministic deviations from that behavior, due to deteriorated or exceptional responses.

Given a multi-tier environment model, the agent simultaneously reasons on the effects of its actions in all tiers when deliberating what to do. This increases the robustness and adaptability of its operations when deployed in complex and uncertain environments. However, while the agent may have winning strategies (plans) to achieve its goals in the most deterministic tier, it may be impossible to have winning strategies also for the most nondeterministic tiers. This calls for notions of strategies that are less stringent than the usual ones used in Formal Methods [Finkbeiner, 2016], or in Planning [Geffner and Bonet, 2013].

One option is to introduce stochastic/quantitative aspects in the models and base reasoning on optimization with probabilistic guarantees [Geffner and Bonet, 2013]. But also in the non-quantitative setting there are interesting solutions, in particular that of *best-effort strategies* [Aminof *et al.*, 2020; Aminof *et al.*, 2021a; Aminof *et al.*, 2021b]: if a strategy to win the goal against all possible environment responses does not exist, instead of giving up, we return a strategy that wins against a *maximal set* (though not all) of possible environment responses. Best-effort strategies are based on the game-theoretic rationality principle that a player (the agent) should not use a strategy that is "dominated" by another one (i.e., if another strategy fulfills the goal against more environment responses, then the player should adopt that strategy). Best-effort strategies have some notable properties: (*i*) they always exist, (*ii*) if a winning strategy exists, then best-effort strategies are exactly the winning strategies, (*iii*) for Linear Temporal Logic specifications both on infinite traces (LTL) [Pnueli, 1977] and on finite traces (LTL_f) [De Giacomo and Vardi, 2013], they can be computed in worst case 2EXPTIME, just as for winning strategies (best-effort synthesis is indeed 2EXPTIME-complete, just as is standard synthesis) [Aminof *et al.*, 2021b].

These results extend to multi-tier environments. In particular, a strategy that is best-effort in all tiers of the multi-tier environment model always exists, i.e., there exists a strategy that in every tier wins against a maximal set of environment strategies. Moreover, such a strategy can be computed in 2EXPTIME for LTL/LTL$_f$ specifications. The latter result has been proved constructively in [Aminof *et al.*, 2021a] by providing a solution technique based on automata on infinite trees. However that automata-based technique is not particularly promising for implementation.

Interestingly, in the case of flat environment models, one can resort to an alternative synthesis technique [Aminof *et al.*, 2021b], which is a game-theoretic construction that can be implemented effectively, especially for LTL$_f$. This technique is based on solving an adversarial and a cooperative game over an arena provided by the environment specification (and the agent goal) and then combining the two solutions.

In this paper, we focus on LTL$_f$[1]. Using LTL$_f$ allows to specify every LTL *guarantee* specification for the goal and every LTL *safety* specification for the environment [Manna and Pnueli, 1990]. Notably, safety environment specifications are a generalization of nondeterministic planning domain specifications (for example, they allow for non-Markovian properties [Gabaldon, 2011]), e.g., written in PDDL making use of *oneof* (dropping preconditions in favor of conditional effects) [Haslum *et al.*, 2019; Aminof *et al.*, 2018; De Giacomo *et al.*, 2023a].

Our first contribution is to show that the game-theoretic approach in [Aminof *et al.*, 2021b] can be extended to handle multi-tier environments. Specifically, we show that we can combine adversarial and cooperative games for each tier of the environment and generate a strategy that is best-effort for all of them (Algorithm 1). However, by adopting such a basic technique, we obtain a game arena that is exponential in the number of tiers, limiting the applicability to only few tiers.

Our second contribution is to show that this exponential blowup in the number of tiers can be avoided. We present a refinement (Algorithm 2) of the basic technique, which is based on solving the games corresponding to an environment specification separately and combining the solutions on-the-fly (in linear time) to obtain a strategy that step-wise returns the next action to be performed. The result is an algorithm that is linear in the number of environment specifications, and worst-case doubly exponential only in the size of the formulas specifying the goal and the environments (Theorem 6).

Our third contribution is to analyze two notable cases of multi-tier environments specified in LTL$_f$: (*i*) the case in which all tiers share a (large) common base component, i.e., have the form $\mathcal{E}_i = \mathcal{E}_c \wedge \mathcal{E}'_i$, and (*ii*) the case in which, each tier is obtained by conjoining some further conditions to the previous one, i.e., $\mathcal{E}_{i-1} = \mathcal{E}_i \wedge \mathcal{E}'_{i-1}$. We exploit this additional structure, getting a construction that is even more scalable.

To show the practicality of the proposed approach, we provide symbolic implementations by leveraging the framework of [Zhu *et al.*, 2017] and, using such implementations, perform an empirical evaluation on some scalable benchmarks.

2 LTL$_f$ Synthesis

A *trace* over an alphabet of symbols Σ is a finite or infinite sequence of elements from Σ. The empty trace is denoted λ. Traces are indexed starting at zero, and we write $\pi = \pi_0\pi_1\cdots$. For a finite trace π, let $\mathsf{lst}(\pi)$ denote the index of the last element of π, i.e., $\mathsf{lst}(\pi) = |\pi| - 1$.

Linear Temporal Logic on finite traces (LTL$_f$) is a specification language for expressing temporal properties on finite traces [De Giacomo and Vardi, 2013]. LTL$_f$ has the same syntax as LTL (which is instead interpreted over infinite traces [Pnueli, 1977]). Given a set AP of atomic propositions (aka atoms), the LTL$_f$ formulas over AP are generated by the following grammar: $\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \circ\varphi \mid \varphi\,\mathcal{U}\,\varphi$, where $a \in AP$. Here $\circ$ (*Next*) and $\mathcal{U}$ (*Until*) are temporal operators. We use standard Boolean abbreviations such as $\vee$ (or), $\supset$ (implies), $true$ and $false$. Moreover, we define the following abbreviations: $\bullet\varphi \equiv \neg\circ\neg\varphi$ (*Weak Next*), $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ (*Eventually*), and $\Box\varphi \equiv \neg\Diamond\neg\varphi$ (*Always*). The size of φ, written $|\varphi|$, is the number of its subformulas. Formulas are interpreted over finite traces π over the alphabet $\Sigma = 2^{AP}$, i.e., the alphabet consisting of the propositional interpretations of the atoms. Thus, for $0 \leq i \leq \mathsf{lst}(\pi)$, $\pi_i \in 2^{AP}$ is the i-th interpretation of π. That an LTL$_f$ formula φ *holds* at instant $i \leq \mathsf{lst}(\pi)$, written $\pi, i \models \varphi$, is defined inductively: 1. $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in AP$); 2. $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$; 3. $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$; 4. $\pi, i \models \circ\varphi$ iff $i < \mathsf{lst}(\pi)$ and $\pi, i+1 \models \varphi$; and 5. $\pi, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff $\exists j$ such that $i \leq j \leq \mathsf{lst}(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k, i \leq k < j$ we have that $\pi, k \models \varphi_1$. We say that π *satisfies* φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

LTL$_f$ *(reactive) synthesis* [De Giacomo and Vardi, 2015] concerns finding a strategy to satisfy an LTL$_f$ *goal* specification. Goals are expressed as LTL$_f$ formulas over $AP = \mathcal{Y} \cup \mathcal{X}$, where $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of variables. Intuitively, $\mathcal{Y}$ (resp. $\mathcal{X}$) is under the agent's (resp. environment's) control. Traces over $\Sigma = 2^{\mathcal{Y}\cup\mathcal{X}}$ will be denoted $\pi = (Y_0 \cup X_0)(Y_1 \cup X_1)\ldots$ where $X_i \subseteq \mathcal{X}$ and $Y_i \subseteq \mathcal{Y}$ for every i. Such infinite traces are called *plays*, and finite traces are called *histories* and represent a sequence of moves of the players ending in an environment move since we assume that the agent moves first.

An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ mapping sequences of environment moves to an agent move. Similarly, an *environment strategy* is a function $\sigma_{env} : (2^{\mathcal{Y}})^+ \to 2^{\mathcal{X}}$ mapping non-empty sequences of agent moves to an environment move. The domain of σ_{ag} includes the empty sequence λ as we assumed that the agent moves first. A trace π is σ_{ag}-consistent if $Y_0 = \sigma_{ag}(\lambda)$ and $Y_{j+1} = \sigma_{ag}(X_0 \cdots X_j)$ for every $j \geq 0$. Analogously, π is σ_{env}-consistent if $X_j = \sigma_{env}(Y_0 \cdots Y_j)$ for every $j \geq 0$. We define PLAY$(\sigma_{ag}, \sigma_{env})$ to be the unique (infinite) trace that is consistent with both σ_{ag} and σ_{env}.

Let φ be an LTL$_f$ formula over $\mathcal{Y} \cup \mathcal{X}$. An agent strategy σ_{ag} is *winning* for (aka *enforces*) φ if, for every environment strategy σ_{env}, some finite prefix of PLAY$(\sigma_{ag}, \sigma_{env})$ satisfies φ. An agent strategy is *cooperatively winning* for φ if there exists an environment strategy σ_{env} such that some finite prefix of PLAY$(\sigma_{ag}, \sigma_{env})$ satisfies φ. LTL$_f$ synthesis is the problem of finding an agent strategy σ_{ag} that enforces φ, if one exists [De Giacomo and Vardi, 2015].

In this paper, we are interested in LTL$_f$ synthesis under environment specifications [Aminof *et al.*, 2018]. Environment

[1] All techniques reported here also apply to LDL$_f$ [De Giacomo and Vardi, 2013] and pure-past LTL [De Giacomo *et al.*, 2020a].

specifications describe some knowledge about how the environment works and are expressed as LTL_f formulas $\mathcal{E}$ over $\mathcal{Y} \cup \mathcal{X}$. An environment strategy σ_{env} is *winning* for (aka *enforces*) $\mathcal{E}$ if, for every agent strategy σ_{ag}, *every* finite prefix of $\text{PLAY}(\sigma_{ag}, \sigma_{env})$ satisfies $\mathcal{E}$. An *environment specification* is an LTL_f formula $\mathcal{E}$ that is enforceable by some environment strategy. We denote by $\Sigma_{\mathcal{E}}$ the set of environment strategies that enforce $\mathcal{E}$.

Definition 1. *[Aminof* et al.*, 2018] Let φ (resp. $\mathcal{E}$) be an LTL_f formula over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal (resp. env spec).* LTL_f Synthesis under environment specifications *is the problem of finding an agent strategy σ_{ag} such that, for every environment strategy $\sigma_{env} \in \Sigma_{\mathcal{E}}$, some finite prefix of $PLAY(\sigma_{ag}, \sigma_{env})$ satisfies φ, if one exists. Such a strategy is winning for φ in $\mathcal{E}$ (aka enforces φ in $\mathcal{E}$)*

An agent strategy σ_{ag} is *cooperatively winning* for φ in $\mathcal{E}$ if there exists an environment strategy $\sigma_{env} \in \Sigma_{\mathcal{E}}$ such that $PLAY(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ.

Synthesis, both with LTL_f environment specifications or not, is 2EXPTIME-complete [De Giacomo and Vardi, 2015; Aminof *et al.*, 2018]. Synthesis under environment specifications is a generalization of synthesis which is obtained by taking $\mathcal{E} = true$.

3 Best-Effort Strategies

We start by recalling basic notions on best-effort strategies [Aminof *et al.*, 2020; Aminof *et al.*, 2021b; Aminof *et al.*, 2021a].

Definition 2. *Let φ and $\mathcal{E}$ be LTL_f formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively, and let σ_1 and σ_2 be agent strategies. We say that σ_1* dominates *σ_2, written $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$, if, for every $\sigma_{env} \in \Sigma_{\mathcal{E}}$, if some finite prefix of PLAY(σ_2, σ_{env}) satisfies φ then some finite prefix of PLAY(σ_1, σ_{env}) satisfies φ. Furthermore, σ_1* strictly dominates *σ_2, written $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$, if $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$ and $\sigma_2 \not\geq_{\varphi|\mathcal{E}} \sigma_1$.*

Intuitively, $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$ means that 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 strategy. An agent using σ_2 is not doing its best, since it could achieve its goal against a strictly larger set of environment strategies using σ_1. In this framework, a best-effort strategy is one which is not strictly dominated by any other strategy.

Definition 3. *An agent strategy σ is* best-effort*, or* maximal*, for φ in $\mathcal{E}$, written $\sigma \in \mathsf{Max}_{\varphi|\mathcal{E}}$, if there does not exist another agent strategy σ' such that $\sigma' >_{\varphi|\mathcal{E}} \sigma$.*

Best-effort strategies also admit a *local characterization* that uses the notion of *value* of a history [Aminof *et al.*, 2021b]. Intuitively, the value of a history h is: "winning", if the agent can enforce φ in $\mathcal{E}$ from h; otherwise, "pending", if the agent has a cooperatively winning strategy for φ in $\mathcal{E}$ from h; otherwise, "losing". With this notion, best-effort strategies are those that witness the maximum value of each history h consistent with them.

For a history h and an agent strategy σ_{ag}, we denote by $\Sigma_{\mathcal{E}}(h, \sigma_{ag})$ the set of environment strategies σ_{env} enforcing $\mathcal{E}$ such that h is consistent with σ_{ag} and σ_{env}. For an agent strategy σ_{ag}, we denote by $\mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ the set of all histories h such that $\Sigma_{\mathcal{E}}(h, \sigma_{ag})$ is non-empty, i.e., $\mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ is the set of all histories that are consistent with σ_{ag} and some environment strategy enforcing $\mathcal{E}$. For $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ define:

1. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = +1$ ("winning"), if for every $\sigma_{env} \in \Sigma_{\mathcal{E}}(h, \sigma_{ag})$, PLAY$(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ; otherwise,
2. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = 0$ ("pending"), if for some $\sigma_{env} \in \Sigma_{\mathcal{E}}(h, \sigma_{ag})$, PLAY$(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ; otherwise,
3. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = -1$ ("losing").

Finally, we denote by $val_{\varphi|\mathcal{E}}(h)$ the maximum of $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h)$ over all σ_{ag} such that $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ (we define $val_{\varphi|\mathcal{E}}(h)$ only in case $h \in \mathcal{H}_{\mathcal{E}}(\sigma)$ for some σ). Here is the local characterization of best-effort strategies:

Theorem 1. *[Aminof* et al.*, 2021b] A strategy σ_{ag} is best-effort for φ in $\mathcal{E}$ (i.e., $\sigma_{ag} \in \mathsf{Max}_{\varphi|\mathcal{E}}$) iff $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = val_{\varphi|\mathcal{E}}(h)$ for every $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$.*

4 Best-Effort Synthesis in Multi-Tier Environments

We now introduce best-effort synthesis in multi-tier environments, i.e., environment models consisting of several *tiers*, each allowing more nondeterminism than the previous one. Formally, an LTL_f *multi-tier environment specification* (aka *multi-tier environment model*) is a tuple $\mathcal{E} = (\mathcal{E}_1, \cdots, \mathcal{E}_n)$ of LTL_f environment tiers such that $\Sigma_{\mathcal{E}_i} \subseteq \Sigma_{\mathcal{E}_{i+1}}$ for every i $(i \leq i < n)$. In this framework, a best-effort strategy is one that is simultaneously best-effort for every tier.

Definition 4. *Given an LTL_f goal φ and an LTL_f multi-tier environment specification $\mathcal{E} = (\mathcal{E}_1, \cdots, \mathcal{E}_n)$, best-effort synthesis is the problem of finding an agent strategy that is best-effort for φ in $\mathcal{E}$, i.e. such that $\sigma_{ag} \in \bigcap_i \mathsf{Max}_{\varphi|\mathcal{E}_i}$.*

Unlike classic LTL_f synthesis [De Giacomo and Vardi, 2015; Pnueli and Rosner, 1989], a best-effort strategy for an LTL_f goal φ in a LTL_f multi-tier environment specification $\mathcal{E}$ always exists, though computing it requires 2EXPTIME as in classic synthesis [Aminof *et al.*, 2021a].

Theorem 2. *[Aminof* et al.*, 2021a] Let φ be an LTL_f goal and $\mathcal{E} = (\mathcal{E}_1, \cdots, \mathcal{E}_n)$ an LTL_f multi-tier environment specification. There exists $\sigma_{ag} \in \bigcap_i \mathsf{Max}_{\varphi|\mathcal{E}_i}$, and it can be computed in 2EXPTIME in the size of $\varphi, \mathcal{E}_1, \cdots, \mathcal{E}_n$.*

We now illustrate such notions in a simple Robot Navigation (in FOND domains) scenario [Cimatti *et al.*, 2003; Alford *et al.*, 2014] where an agent has to plan in spite of increasing nondeterminism.

Example 1. *An autonomous agent is assigned the goal of delivering packages in a building by moving across rooms. Assume that there is a kid in the building who has keys to close some doors. It is easy to see that the agent goal may not be realizable as, e.g., the kid might lock the robot in a room. Hence, the agent could use a best-effort strategy. In this scenario, an LTL_f environment describes the initial state, the transitions of the planning domain, and that the kid might* *close doors for which he has a key. Assume that the designer has no knowledge of which keys the kid holds. Then, the agent could be provided with several specifications describing the possible environment responses, each of which assumes that the kid holds some keys. In a multi-tier environment model, each tier assumes that the kid has more and more keys. A strategy* σ_{ag} *that is best-effort in such a multi-tier environment will intuitively behave as follows: at every point in time, and for each environment specification, if the goal is enforceable (e.g., because the kid cannot prevent this) then* σ_{ag} *will enforce it, and if the goal requires cooperation to achieve (e.g., because the kid has the keys to the relevant rooms) then* σ_{ag} *will achieve the goal if the kid chooses to cooperate. The fact that such a strategy exists is non-trivial in general, and follows from Theorem 2.*

5 Solving Best-Effort Synthesis in Multi-Tier Environments

While [Aminof *et al.*, 2021a] provide a solution technique for best-effort synthesis in multi-tier domains, their technique is based on automata on infinite trees and is not well suited for efficient implementation. Here, we provide a different technique based on DFA games, as that for best-effort synthesis in a flat environment from [Aminof *et al.*, 2021b], but extended to handle multi-tier environments.

Deterministic Finite Automata. For convenience, we separate the acceptance condition of automata from their structure. We define a *deterministic transition system* (aka *transition system*) as a tuple $\mathcal{D} = (\Sigma, S, s_0, \delta)$, where: Σ is a finite input alphabet (usually $\Sigma = 2^{AP}$); S is a finite set of states; $s_0 \in S$ is the initial state; and $\delta : S \times \Sigma \to S$ is the transition function. The *size* of $\mathcal{D}$ is the cardinality of S. Let $\alpha = \alpha_0\alpha_1 \dots \alpha_n$ be a finite trace over the alphabet Σ. The *run* of α in $\mathcal{D}$ is the finite sequence of states $\rho = s_0 s_1 \dots s_{n+1}$ such that s_0 is the initial state of $\mathcal{D}$ and $s_{i+1} = \delta(s_i, \alpha_i)$ for every $i \leq \mathsf{lst}(\alpha)$. We extend δ to be a function $\delta : S \times \Sigma^* \to S$ as follows: $\delta(s, \lambda) = s$, and if $s_n = \delta(s, \alpha_0 \dots \alpha_{n-1})$ then $\delta(s, \alpha_0 \dots \alpha_n) = \delta(s_n, \alpha_n)$.

Definition 5. *The synchronous product of two transition systems* $\mathcal{D}_i = (\Sigma, S_i, s_{(0,i)}, \delta_i)$ *(for* $i = 1, 2$*) over the same alphabet is the transition system* $\textsc{Product}(\mathcal{D}_1, \mathcal{D}_2) = \mathcal{D}_1 \times \mathcal{D}_2 = (\Sigma, S, s_0, \delta)$ *with:* $S = S_1 \times S_2$*;* $s_0 = (s_{(0,1)}, s_{(0,2)})$*; and* $\delta((s_1, s_2), x) = (\delta(s_1, x), \delta(s_2, x))$*. The product* $\textsc{Product}(\mathcal{D}_1, \cdots, \mathcal{D}_n) = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ *is defined analogously for any finite sequence* $\mathcal{D}_1, \cdots, \mathcal{D}_n$ *of transition systems over the same alphabet.*

A *deterministic finite automaton* (DFA) is a pair $\mathcal{A} = (\mathcal{D}, F)$, where $\mathcal{D} = (\Sigma, S, s_0, \delta)$ is a deterministic transition system and $F \subseteq S$ is the set of *final states* of the system. A trace α is *accepted* if $\delta(s_0, \alpha) \in F$. The *language* of $\mathcal{A}$ is the set of traces that the automaton accepts.

Theorem 3. *[De Giacomo and Vardi, 2013] Given an* LTL$_f$ *formula* φ *over AP, we can build a* DFA*, denoted* TODFA(φ)*, whose size is at most 2EXP in* $|\varphi|$ *and whose language is the set of finite traces that satisfy* φ*.*

DFA Games. A DFA $(\mathcal{D}, F)$ in which $\Sigma = 2^{\mathcal{Y} \cup \mathcal{X}}$ is called a DFA game. Here, $\mathcal{D}$ is called the *game arena*, and F is called the *goal*. The notions of plays, histories, and strategies from the Preliminaries Section apply also in this setting. A play is *winning* if it contains a finite prefix that is accepted by the DFA. Intuitively, winning a DFA game requires that F is visited at least once. An agent strategy σ_{ag} is *winning* if, for every σ_{env}, $\mathrm{PLAY}(\sigma_{ag}, \sigma_{env})$ is winning. Furthermore, an agent strategy is *cooperatively winning* if there exists σ_{env} such that $\mathrm{PLAY}(\sigma_{ag}, \sigma_{env})$ is winning. Finally, an environment strategy σ_{env} is *winning* if, for every σ_{ag}, $\mathrm{PLAY}(\sigma_{ag}, \sigma_{env})$ is not winning. The *winning region* (resp. *cooperatively winning region*) is the set of states $s \in S$ for which the agent has a winning (resp. cooperatively winning) strategy in the game $\mathcal{G}' = (\mathcal{D}', F)$, where $\mathcal{D}' = (2^{\mathcal{Y} \cup \mathcal{X}}, S, s, \delta)$, i.e., the same game as G, but with initial state s. The *environment winning region* is defined analogously. An agent strategy that is winning from every state in the agent winning region (resp. cooperatively winning region) is called *uniform winning* (resp. *uniform cooperatively winning*). Of special interest is the case where the agent strategy can be derived from a function $\kappa_{ag} : S \to 2^{\mathcal{Y}}$, called a *positional strategy*, mapping states of the game to agent moves. While a positional strategy is not formally an agent strategy (i.e., a function from sequences of environment moves to agent moves), it induces one as follows: $\textsc{Strategy}(\mathcal{D}, \kappa_{ag})(h) = \kappa_{ag}(\delta(s_0, h))$. The pair $(\mathcal{D}, \kappa_{ag})$ is sometimes called a *transducer*, i.e., a deterministic transition-system with output. *Solving* a DFA game is the problem of computing the agent winning (resp. cooperatively winning) region and determining a positional winning strategy (resp. positional cooperatively winning) strategy, written $(W, \kappa_{ag}) = \textsc{SolveAdv}(\mathcal{D}, F)$ (resp. $(W', \gamma_{ag}) = \textsc{SolveCoop}(\mathcal{D}, F)$). Games played over DFAs are determined, meaning that the agent winning region and the environment winning region partition the state space [Gale and Stewart, 1953]. DFA games can be solved in linear time in the size of the game arena through a least-fixpoint computation [Apt and Grädel, 2011]. The environment winning region is denoted $\textsc{EnvWin}(\mathcal{D}, F)$.

Sometimes, transitions must be constrained to those that do not allow the game to leave a set of states:

Definition 6. *Let* $\mathcal{D} = (\Sigma, S, s_0, \delta)$ *be a transition system and* $S' \subseteq S$ *a non-empty set of states. The restriction of* $\mathcal{D}$ *to* S' *is the transition system* $\textsc{Restrict}(\mathcal{D}, S') = (\Sigma, S' \cup \{sink\}, s_0, \delta')$ *where, for every* $a \in \Sigma$*,* $\delta'(s, a) = sink$ *if* $s = sink$ *or* $\delta(s, a) \notin S'$*, and* $\delta'(s, a) = \delta(s, a)$ *otherwise.*

Solution Technique – Basic Version. As a first step towards developing our solution, we first review the core step in [Aminof *et al.*, 2021b; De Giacomo *et al.*, 2023b][2] for the

[2]We observe that in [Aminof *et al.*, 2021b; Aminof *et al.*, 2023] the environment moves first. This causes a mismatch between the automata constructed in the algorithms of those papers and the local characterization. While the local characterization talks about histories ending in environment moves, finite runs in automata correspond to histories ending in agent moves. This mismatch causes a bug in the algorithms of those papers. This bug can be fixed by having the agent move first, as we do in this paper. If one wishes for the

Algorithm 0 $\textsc{SynthPos}(\varphi, \mathcal{E})$

Input: LTL_f goal φ and an env. specification $\mathcal{E}$
Output: goal DFA $\mathcal{A}_\varphi$; env. DFA $\mathcal{A}_\mathcal{E}$; winning region W; cooperatively winning region W'; positional winning strategy κ; positional cooperatively winning strategy γ

1: $\mathcal{A}_\varphi = \textsc{toDFA}(\varphi)$; $\mathcal{A}_\mathcal{E} = \textsc{toDFA}(\mathcal{E})$
 Say $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$ and $\mathcal{A}_\mathcal{E} = (\mathcal{D}_\mathcal{E}, F_\mathcal{E})$
2: $\mathcal{D} = \textsc{Product}(\mathcal{D}_\mathcal{E}, \mathcal{D}_\varphi)$
3: Let:
 - $F_{\mathcal{E}\supset\varphi} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \in F_\mathcal{E} \supset s_\varphi \in F_\varphi\}$
 - $F_{\neg\mathcal{E}} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \notin F_\mathcal{E}\}$
 - $F_{\mathcal{E}\wedge\varphi} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \in F_\mathcal{E} \wedge s_\varphi \in F_\varphi\}$

4: $(W, \kappa) = \textsc{SolveAdv}(\mathcal{D}, F_{\mathcal{E}\supset\varphi})$
5: $V = \textsc{EnvWin}(\mathcal{D}, F_{\neg\mathcal{E}})$
6: $\mathcal{D}' = \textsc{Restrict}(\mathcal{D}, V)$
7: $(W', \gamma) = \textsc{SolveCoop}(\mathcal{D}', F_{\mathcal{E}\wedge\varphi})$
8: **Return** $(\mathcal{A}_\varphi, \mathcal{A}_\mathcal{E}, W, W', \kappa, \gamma)$

Algorithm 1 $\textsc{MultiEnvBESynth}(\varphi, \mathcal{E}_1, \cdots, \mathcal{E}_n)$

Input: LTL_f goal φ and a multi-tier env. $\mathcal{E} = (\mathcal{E}_1 \cdots \mathcal{E}_n)$
Output: Agent strategy σ that is best-effort for φ in $\mathcal{E}$

1: For $i = 1 \dots n$:
 $(\mathcal{A}_\varphi, \mathcal{A}_{\mathcal{E}_i}, W_i, W'_i, \kappa_i, \gamma_i) = \textsc{SynthPos}(\varphi, \mathcal{E}_i)$[3]
 Say $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$, $\mathcal{D}_\varphi = (2^{\mathcal{Y}\cup\mathcal{X}}, S_\varphi, s_\varphi, \delta_\varphi)$
 Say $\mathcal{A}_{\mathcal{E}_i} = (\mathcal{D}_{\mathcal{E}_i}, F_{\mathcal{E}_i})$, $\mathcal{D}_{\mathcal{E}_i} = (2^{\mathcal{Y}\cup\mathcal{X}}, S_{\mathcal{E}_i}, s_{\mathcal{E}_i}, \delta_{\mathcal{E}_i})$
2: $\mathcal{D} = \textsc{Product}(\mathcal{D}_{\mathcal{E}_1}, \cdots, \mathcal{D}_{\mathcal{E}_n}, \mathcal{D}_\varphi)$
 Say $S = S_{\mathcal{E}_1} \times \cdots \times S_{\mathcal{E}_n} \times S_\varphi$
3: Define a positional strategy ν on S as follows.
 For $s = (s_1, \cdots, s_n, t) \in S$:
 1. $j = \mathit{max}\{i : (s_i, t) \in W_i\}$
 2. $\ell = \mathit{min}\{i : (s_i, t) \in W'_i\}$
 3. **if** j exists then define $\nu(s) = \kappa_j(s_j, t)$[4]
 else if ℓ exists then define $\nu(s) = \gamma_\ell(s_\ell, t)$
 else define $\nu(s) = \mathcal{Y}$ (i.e., arbitrarily) **endif**

4: **Return** $\textsc{Strategy}(\mathcal{D}, \nu)$

single environment LTL_f best-effort synthesis problem, encapsulated in Algorithm 0. That allows one to compute a positional strategy as follows: it maps a state s in $\mathcal{D} = \mathcal{D}_\mathcal{E} \times \mathcal{D}_\varphi$ to $\kappa(s)$ if $s \in W$, to $\gamma(s)$ if $s \in W' \setminus W$, and is arbitrary otherwise. Intuitively, the histories whose induced runs in $\mathcal{D}$ that pass or end in a state in W have value $+1$ (as witnessed by κ), those ending in a state in $W' \setminus W$ have value 0 (as witnessed by γ), and the rest have value -1. The correctness is a consequence of the local characterization (Theorem 1).

With the auxiliary procedure Algorithm 0 in place, we are ready to present our solution technique. For we make no effort to gain maximal efficiency, which we will do in the next section, this solution should be thought of as a conceptual solution and not yet a blueprint for implementation. The technique is detailed in Algorithm 1 and returns a strategy obtained by combining the solutions of simple (adversarial and cooperative) DFA games, two for each environment specification $\mathcal{E}_i$, computed in Step 1 by calling Algorithm 0. These strategies are combined into a positional strategy over the Cartesian product of all games computed in Step 3. Algorithm 1 exploits the fact the environment is given in the form of a multi-tier environment, i.e., $\Sigma_{\mathcal{E}_1} \subseteq \cdots \subseteq \Sigma_{\mathcal{E}_n}$, as follows. Suppose $k < i$. Then, an agent strategy that wins for φ in $\mathcal{E}_i$ also wins for φ in $\mathcal{E}_k$ since winning against all the strategies in $\Sigma_{\mathcal{E}_i}$ also wins against all the strategies in the subset $\Sigma_{\mathcal{E}_k}$. Similarly, an agent strategy that cooperatively wins for φ in $\mathcal{E}_k$ also cooperatively wins for φ in $\mathcal{E}_i$ since a cooperating environment strategy in $\Sigma_{\mathcal{E}_k}$ is also in $\Sigma_{\mathcal{E}_i}$. Intuitively, histories whose induced runs in the product $\mathcal{D}$ that pass or end in a state whose j-th coordinate is in W_j (where j is computed in Step 3) have value $+1$ for each of the environment specifications $\mathcal{E}_1$ up to $\mathcal{E}_j$, and have value 0 for each of the environment specifications $\mathcal{E}_{j+1}$ up to $\mathcal{E}_n$; of the remaining histories, those ending in a state whose ℓ-th coordinate is in W'_ℓ have value 0 for each of the environment specifications $\mathcal{E}_\ell$ up to $\mathcal{E}_n$, and otherwise have value -1. The following theorem shows the correctness of the solution technique above:

Theorem 4. *Algorithm 1 returns a strategy in* $\bigcap_{i \leq n} \mathsf{Max}_{\varphi|\mathcal{E}_i}$.

6 Advanced Solution Technique

Although Algorithm 1 is correct, its runtime grows *exponentially* in n, the number of tiers in the multi-tier environment. Indeed, Algorithm 1 returns an agent strategy represented as a transducer with state space $S_{\mathcal{E}_1} \times \cdots \times S_{\mathcal{E}_n} \times S_\varphi$, where S_φ is the state space of $\mathcal{A}_\varphi$ and, for every i, $S_{\mathcal{E}_i}$ is the state space of $\mathcal{A}_{\mathcal{E}_i}$. The size of this state space grows exponentially in n. Constructing the positional strategy ν requires searching the whole state space (Step 3), and hence exponential time in n. Such exponential dependency limits the applicability of Algorithm 1 to best-effort synthesis problems with just few tiers. However, the ideas at the base of Algorithm 1 can be refined to avoid the exponential blow-up.

To do so, we substitute Algorithm 1 with Algorithm 2. The key difference is that we avoid the construction of the Cartesian product and instead return a strategy that determines on-the-fly, at each instant, the next action to perform by scanning in *linear* time the regions W_i and W'_i (for $1 \leq i \leq n$) and choosing a suitable output from the strategies κ_i and γ_i. With this technique, the cost of computing the output best-effort strategy is just *linear* in n (while remaining double exponential in the size of the LTL_f formulas $\mathcal{E}_1, \cdots, \mathcal{E}_n, \varphi$), as well as the time cost for executing, at any instant, the output strategy. Since it is easy to see that the strategy returned by Algorithm 2 is equivalent to the strategy returned by Algorithm 1, from Theorem 4 we get the correctness of Algorithm 2.

Theorem 5. *Algorithm 2 returns a strategy in* $\bigcap_{i \leq n} \mathsf{Max}_{\varphi|\mathcal{E}_i}$.

Complexity. By analyzing Algorithm 2 we see that, while its complexity is 2EXPTIME in the LTL_f formulas (as classic synthesis), it depends linearly on the number of tiers:

environment to move first, we can easily change the specification so that it ignores the first agent move. Alternatively, we can modify the automata construction by adding intermediate states in each transition that correspond to half time-steps after the environment move but before the corresponding agent move.

[2]In fact, we only need to call $\textsc{toDFA}(\varphi)$ once.

[4]Recall that by Alg 0, the domains of κ_i and γ_i are $S_{\mathcal{E}_i} \times S_\varphi$.

Algorithm 2 ONTHEFLYBESYNTH($\varphi, \mathcal{E}_1, \cdots, \mathcal{E}_n$)

Input: LTL_f goal φ and a multi-tier env. $\mathcal{E} = (\mathcal{E}_1 \cdots \mathcal{E}_n)$
Output: Agent strategy σ that is best-effort for φ in $\mathcal{E}$

1: For $i = 1 \ldots n$:
   $(\mathcal{A}_\varphi, \mathcal{A}_{\mathcal{E}_i}, W_i, W_i', \kappa_i, \gamma_i) = \text{SYNTHPOS}(\varphi, \mathcal{E}_i)$[2]
   Say $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi), \mathcal{D}_\varphi = (2^{\mathcal{Y} \cup \mathcal{X}}, S_\varphi, s_\varphi, \delta_\varphi)$
   Say $\mathcal{A}_{\mathcal{E}_i} = (\mathcal{D}_{\mathcal{E}_i}, F_{\mathcal{E}_i}), \mathcal{D}_{\mathcal{E}_i} = (2^{\mathcal{X} \cup \mathcal{Y}}, S_{\mathcal{E}_i}, s_{\mathcal{E}_i}, \delta_{\mathcal{E}_i})$
2: **Return** the following best-effort strategy:
   While *true*:
   1. $j = max\{i : (s_{\mathcal{E}_i}, s_\varphi) \in W_i\}$
   2. $\ell = min\{i : (s_{\mathcal{E}_i}, s_\varphi) \in W_i'\}$
   3. **if** j exists, **output** $Y = \kappa_j(s_{\mathcal{E}_j}, s_\varphi)$
      else if ℓ exists, **output** $Y = \gamma_\ell(s_{\mathcal{E}_\ell}, s_\varphi)$
      else output $Y = \mathcal{Y}$ **endif**
   4. On environment's choice $X \subseteq \mathcal{X}$:
      - Update $s_\varphi = \delta_\varphi(s_\varphi, Y \cup X)$
      - For $i = 1 \ldots n$: update $s_{\mathcal{E}_i} = \delta_{\mathcal{E}_i}(s_{\mathcal{E}_i}, Y \cup X)$

Theorem 6. *Let φ be an LTL_f goal and $\mathcal{E} = (\mathcal{E}_1, \cdots, \mathcal{E}_n)$ a multi-tier environment specification. Then Algorithm 2 computes a strategy $\sigma \in \bigcap_i \mathsf{Max}_{\varphi|\mathcal{E}_i}$ in 2EPXTIME in the size of* $\varphi, \mathcal{E}_1, \cdots, \mathcal{E}_n$ and *in* linear time *in n, the number of tiers in the multi-tier environment.*

Specifically, Algorithm 2 finds, at each instant (history), the next agent move (assignment of the $\mathcal{Y}$) in *linear time* in the number of tiers, i.e., Algorithm 2 (differently form Algorithm 1) scales graciously as the number of tiers grows.

Interestingly, the computations in Step 1 of Algorithms 1 and 2 can be done in parallel. The $n+1$ steps for computing the DFAs of the goal and the n environment specifications can be done in parallel; the n steps for computing the regions W_i and the strategies κ_i can be done in parallel; the n steps for computing the regions W_i' and the strategies γ_i can be done in parallel. As a result, if $n+1$ processors are available, handling multi-tier environments is virtually for free, i.e., costs the same as handling the most computationally expensive tier.

These features suggest that Algorithm 2 is suited for efficient implementation, as confirmed empirically in Section 8.

7 Notable Cases

Before turning to implementation and experimental evaluation, we consider two notable cases of multi-tier environment models, for which we can offer further optimizations.

Multi-Tier Environments with a Common Base. In this case we have a (large) *common base* $\mathcal{E}_c$ that is common to all tiers. That is, each tier $\mathcal{E}_i$ is specified as conjunction of $\mathcal{E}_c$ with some additional LTL_f specification $\mathcal{E}_i'$. Formally, multi-tier environments with a common base have the form: for every i s.t. $1 \leq i \leq n$, $\mathcal{E}_i = \mathcal{E}_c \wedge \mathcal{E}_i'$, where $\Sigma_{\mathcal{E}_1'} \subseteq \cdots \subseteq \Sigma_{\mathcal{E}_n'}$.

Multi-Tier Environments with Conjunctive Refinements. Next, we consider multi-tier environments consisting of tiers that conjoin further constraints to the previous tier, becoming more determined. That is, the *base environment* is $\mathcal{E}_n$, and each tier $\mathcal{E}_i$ refines $\mathcal{E}_{i+1}$ with some conjunct $\mathcal{E}_i'$. Formally, multi-tier environments with conjunctive refinements have the form: for every i s.t. $1 \leq i < n$, $\mathcal{E}_i = \mathcal{E}_{i+1} \wedge \mathcal{E}_i'$.

Exploiting Structure in Notable Cases. By taking advantage of the syntactic structure of these notable cases, we can devise optimized variants of Algorithm 2 that construct more efficiently the DFAs of the tiers. To do this, we exploit the following composition technique, whose correctness follows immediately by the notion of product of transition systems:

Theorem 7. *Given n LTL_f formulas ψ_i, let $\psi = \bigwedge_{1 \leq i \leq n} \psi_i$. If $\mathcal{A}_{\psi_i} = (\mathcal{D}_{\psi_i}, F_{\psi_i})$ is a* DFA *recognizing ψ_i (for $1 \leq i \leq n$), then the* DFA *$\mathcal{A}_\psi = (\mathcal{D}_\psi, F_\psi)$ recognizes ψ: $\mathcal{D}_\psi =$* PRODUCT$(\mathcal{D}_{\psi_1}, \cdots, \mathcal{D}_{\psi_n})$ *and $F_\psi = F_{\psi_1} \times \cdots \times F_{\psi_n}$*

With Theorem 7, we can construct the DFAs of the tiers as follows: (*i*) we construct the DFA of the base conjunct, i.e., $\mathcal{E}_c$ and $\mathcal{E}_n$, respectively; (*ii*) we construct the DFAs of the conjuncts $\mathcal{E}_i'$; (*iii*) we compose the obtained DFAs to construct the DFAs of the environment tiers. Constructing and composing the DFAs of the various conjuncts takes less time than transforming every tier into a DFA as a whole, especially if the size of the least refined tier is large and dominates that of the other conjuncts. This is confirmed empirically in Section 8.

8 Implementation and Evaluation

We implemented Algorithm 2 in a tool called *MtSyft*[5], leveraging the symbolic LTL_f synthesis framework [Zhu *et al.*, 2017], at the base of state-of-the-art LTL_f synthesis tools [Bansal *et al.*, 2020; Favorito and Zhu, 2023]. We also developed variants of *MtSyft* customized for the two notable cases above, called *cb-MtSyft* and *conj-MtSyft*. In *MtSyft*, we build the minimized explicit-state DFAs of LTL_f formulas with LYDIA [De Giacomo and Favorito, 2021], which is among the best performing tools publicly available for LTL_f-to-DFA conversion. We code Boolean functions representing transitions and final states of symbolic DFAs by BDDs [Bryant, 1992] with the BDD library CUDD 3.0.0 [Somenzi, 2016]. We compute the positional strategies for the DFA-games through Boolean synthesis [Fried *et al.*, 2016].

Setup. Experiments were run on a laptop with an operating system 64-bit Ubuntu 20.04, 3.6 GHz CPU, and 12 GB of memory. Timeout was set to 300 seconds.

Benchmark. To evaluate the performance of our implementations, we devised an extension of the counter game benchmarks presented in [De Giacomo *et al.*, 2020b; Zhu *et al.*, 2020] to construct multi-tier environment specifications. The counter game involves a k-bit counter as follows: (*i*) at each round, the environment chooses whether to request an increment of the counter (*add*), and the agent chooses whether to grant such a request or not; (*ii*), the counter is initialized with all bits set to 0, and the agent goal is for the counter to have all bits set to 1; (*iii*) multi-tier environment specifications define possible policies according to which the environment issues increment requests. Specifically, environment specifications are LTL_f formulas $\mathcal{E}_1 = add$, and for $m \geq 2$, $\mathcal{E}_m = \mathcal{E}_{m-1} \wedge \bullet\bullet \cdots \bullet add$, where there are $m-1$ occurrences of $\bullet$ in $\mathcal{E}_m$. In our experiments, the environment issues between 1 and 100 increment requests ($1 \leq m \leq 100$). Given a k-bits counter and an environment specification $\mathcal{E}_m$,

[5]https://github.com/GianmarcoDIAG/MtSyft

Bits	Coverage *MtSyft*	Coverage *cb-MtSyft*	Avg. RT (secs) *MtSyft*	Avg. RT (secs) *cb-MtSyft*
1	82	82	50.98	54.92
2	82	82	55.85	51.46
3	81	83	52.04	54.95
4	80	82	49.55	53.40
5	82	84	55.75	57.99
6	82	82	58.84	55.53
7	81	80	64.05	57.64
8	76	77	73.42	74.94
9	0	0	-	-
10	0	0	-	-
Total	646	652		

Table 1: Coverage (solved instances out of 100) and average runtime (Avg. RT) achieved by *MtSyft* and *cb-MtSyft* in counter games with base conjunct $\mathcal{E}_1$ and number of tiers $1 \leq n \leq 100$.

the realizability of the agent goal (existence of a winning strategy for the goal) depends on k and m: if $m \geq 2^k - 1$, the goal is realizable. Regardless of the realizability of the agent goal, a best-effort (possibly winning) strategy for the agent is to accept all environment increment requests.

Our benchmark consists of counter games with at most 10-bits. For each game, we constructed multi-tier environments with n tiers as follows: (*i*) we fixed a base conjunct $\mathcal{E}_\ell$; (*ii*) stacked the tiers $\mathcal{E}_\ell, \cdots, \mathcal{E}_{\ell+n-1}$ in increments of 1. As base conjucts, we considered $\mathcal{E}_1$, and $\mathcal{E}_{10}$ to $\mathcal{E}_{90}$ in increments of 10. In total, our benchmark consists of about 5600 instances.

Empirical Results. We performed experiments to assess: (*i*) the practical feasibility of best-effort synthesis in multi-tier environments as the number of tiers grows; and (*ii*) further scalability improvement obtainable exploiting the special structure of the two notable cases.

Table 1 shows the performance of *MtSyft* in counter game instances with number of tiers between 1 and 100 ($1 \leq n \leq 100$) where $\mathcal{E}_1$ is the base conjuct. We can see that *MtSyft*, run on a laptop, solves at most 8-bits counter games up to 76 tiers within the 300 secs timeout. For 8-bits counter games (or lower) the computational bottleneck is converting LTL$_f$ tier specifications into DFAs. Instead, solving the single games and composing the synthesized positional strategies into a the best-effort strategy (expressed as the while-program returned by Algorithm 2) brings only a *minor computational overhead*. For 9-bits and 10-bits counter games, *MtSyft* reaches the timeout while converting the LTL$_f$ goal itself (i.e., the counter specification) into a DFA (as opposed to the tiers). This is an excellent scalability result with respect to the number of tiers and confirms the practical feasibility of the synthesis in multi-tier environments. Table 1 includes the performance of *cb-MtSyft* as well, but the size of the base conjunct is too small to observe a significant scalability improvement.

Figure 1 shows the performance comparison of *MtSyft* and *cb-MtSyft* in 8-bits counter games with base conjunct $\mathcal{E}_{80}$ (i.e., it has 80 successive increment request) and number of tiers between 1 and 20 (hence, going from $\mathcal{E}_{80}$ to $\mathcal{E}_{100}$). The results show that *cb-MtSyft* successfully solves all the considered instances, while *MtSyft* reaches the timeout when $n = 14$. In the solved instances, *cb-MtSyft* scales much

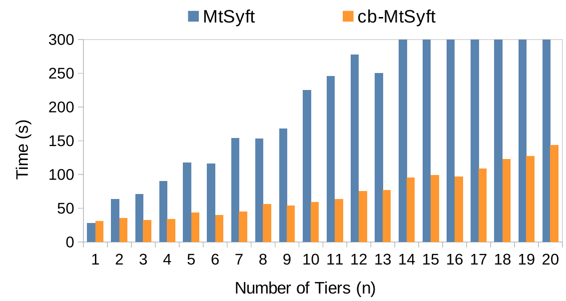

Figure 1: *MtSyft* and *cb-MtSyft* comparison on 8-bits counter games with base conjunct $\mathcal{E}_{80}$ and number of tiers $1 \leq n \leq 20$.

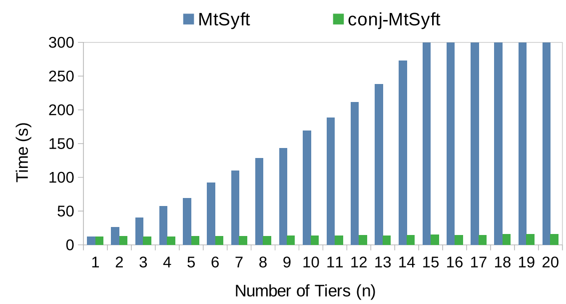

Figure 2: *MtSyft* and *conj-MtSyft* comparison on 1-bit counter games with base conjunct $\mathcal{E}_{80}$ and number of tiers $1 \leq n \leq 20$.

better than *MtSyft* by exploiting the structure of the multi-tier environment to more efficiently construct the DFAs of the tiers. Figure 2 shows an analogous result on the performance comparison of *MtSyft* and *conj-MtSyft* in 1-bit counter games with base conjunct $\mathcal{E}_{80}$ and number of tiers between 1 and 20. We get the same performance up to 3-bits counters, then for 4-bits on it goes in time out. The reason is that our implementation does the conjunctions symbolically without minimizing the result, i.e., the product DFA grows exponentially in the number of conjuncts. To avoid the exponential blowup, one should adopt a more sophisticated way of handling conjunctions, as in [Bansal *et al.*, 2020; Bansal *et al.*, 2022].

9 Conclusion

We developed an effective technique to solve LTL$_f$ best-effort synthesis in multi-tier environments which allow for increasing nondeterminism. In our framework, we have considered a single goal for all tiers. However, it is also of interest to consider the case in which, as tiers become more undetermined, also the goal is weakened. For instance, the agent may have a primary goal that requires a certain type of environment behavior, but also secondary goals that can be achieved even if the environment does not behave as expected. This was studied for PDDL planning in [Ciolek *et al.*, 2020]. We believe that the general approach presented here can be extended to handle this case as well. We leave the details for future work.

Acknowledgments

This work is supported in part by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013). Gianmarco Parretti is supported by the Italian National Ph.D. on Artificial Intelligence at Sapienza.

References

[Alford *et al.*, 2014] Ronald Alford, Ugur Kuter, Dana S. Nau, and Robert P. Goldman. Plan aggregation for strong cyclic planning in nondeterministic domains. *Artificial Intelligence*, 216:206–232, 2014.

[Aminof *et al.*, 2018] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning and synthesis under assumptions. *arXiv*, 2018.

[Aminof *et al.*, 2019] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning under LTL environment specifications. In *ICAPS*, pages 31–39, 2019.

[Aminof *et al.*, 2020] Benjamin Aminof, Giuseppe De Giacomo, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. Synthesizing strategies under expected and exceptional environment behaviors. In *IJCAI*, 2020.

[Aminof *et al.*, 2021a] Benjamin Aminof, Giuseppe De Giacomo, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. Synthesizing best-effort strategies under multiple environment specifications. In *KR*, pages 42–51, 2021.

[Aminof *et al.*, 2021b] Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Best-effort synthesis: Doing your best is not harder than giving up. In *IJCAI*, pages 1766–1772, 2021.

[Aminof *et al.*, 2023] Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Reactive synthesis of dominant strategies. In *AAAI*, pages 6228–6235. AAAI Press, 2023.

[Apt and Grädel, 2011] Krzysztof R. Apt and Erich Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

[Bansal *et al.*, 2020] Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020.

[Bansal *et al.*, 2022] Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi, and Shufang Zhu. Compositional safety LTL synthesis. In *VSTTE*, volume 13800 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2022.

[Bryant, 1992] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[Camacho *et al.*, 2019] Alberto Camacho, Meghyn Bienvenu, and Sheila A McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, pages 58–67, 2019.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147:35–84, 2003.

[Ciolek *et al.*, 2020] Daniel Alfredo Ciolek, Nicolás D'Ippolito, Alberto Pozanco, and Sebastian Sardiña. Multi-tier automated planning for adaptive behavior. In *ICAPS*, pages 66–74. AAAI Press, 2020.

[De Giacomo and Favorito, 2021] Giuseppe De Giacomo and Marco Favorito. Compositional approach to translate LTL_f/LDL_f into deterministic finite automata. In *ICAPS*, pages 122–130, 2021.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.

[De Giacomo *et al.*, 2020a] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI*, pages 4959–4965. ijcai.org, 2020.

[De Giacomo *et al.*, 2020b] Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y Vardi, and Shufang Zhu. Two-stage technique for LTL_f synthesis under LTL assumptions. In *KR*, volume 17, pages 304–314, 2020.

[De Giacomo *et al.*, 2023a] Giuseppe De Giacomo, Gianmarco Parretti, and Shufang Zhu. LTL_f best-effort synthesis in nondeterministic planning domains. In *ECAI*, pages 533–540, 2023.

[De Giacomo *et al.*, 2023b] Giuseppe De Giacomo, Gianmarco Parretti, and Shufang Zhu. Symbolic LTL_f best-effort synthesis. In *EUMAS*, pages 228–243, 2023.

[Favorito and Zhu, 2023] Marco Favorito and Shufang Zhu. LydiaSyft: A compositional symbolic synthesizer for LTL_f specifications. 2023.

[Finkbeiner, 2016] Bernd Finkbeiner. Synthesis of reactive systems. *Dependable Software Systems Eng.*, 45:72–98, 2016.

[Fried *et al.*, 2016] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. BDD-based Boolean functional synthesis. In *CAV*, pages 402–421, 2016.

[Gabaldon, 2011] Alfredo Gabaldon. Non-Markovian control in the situation calculus. *Artificial Intelligence*, 175:25–48, 2011.

[Gale and Stewart, 1953] David Gale and Frank M Stewart. Infinite games with perfect information. *Contributions to the Theory of Games*, 2(245-266):2–16, 1953.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool, 2013.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction* *to the Planning Domain Definition Language*. M&C, 2019.

[Manna and Pnueli, 1990] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *PODC*, pages 377–410, 1990.

[Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, page 179–190, 1989.

[Pnueli, 1977] Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57, 1977.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Somenzi, 2016] Fabio Somenzi. CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder. 2016.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTL$_f$ synthesis. In *IJCAI*, pages 1362–1369, 2017.

[Zhu *et al.*, 2020] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. LTL$_f$ synthesis with fairness and stability assumptions. In *AAAI*, pages 3088–3095, 2020.