Compositional Safety LTL Synthesis

Suguman Bansal[1], Giuseppe De Giacomo[2], Antonio Di Stasio[2],
Yong Li[3], Moshe Y. Vardi[4], and Shufang Zhu[2]

[1] University of Pennsylvania, PA, USA
[2] Sapienza University of Rome, Italy
[3] SKLCS, Institute of Software, CAS, China
[4] Rice University, TX, USA

Abstract. Reactive synthesis holds the promise of generating automatically a verifiably correct program from a high-level specification. A popular such specification language is Linear Temporal Logic (LTL). Unfortunately, synthesizing programs from *general* LTL formulas, which relies on first constructing a game arena and then solving the game, does not scale to large instances. The specifications from practical applications are usually large conjunctions of smaller LTL formulas, which inspires existing compositional synthesis approaches to take advantage of this structural information. The main challenge here is that they solve the game only after obtaining the game arena, the most computationally expensive part in the procedure. In this work, we propose a compositional synthesis technique to tackle this difficulty by synthesizing a program for each small conjunct separately and composing them one by one. While this approach does not work for general LTL formulas, we show here that it does work for *Safety* LTL formulas, a popular and important fragment of LTL. While we have to compose all the programs of small conjuncts in the worst case, we can prune the intermediate programs to make later compositions easier and immediately conclude unrealizable as soon as some part of the specification is found unrealizable. By comparing our compositional approach with a portfolio of all other approaches, we observed that our approach was able to solve a notable number of instances not solved by others. In particular, experiments on scalable conjunctive benchmarks showed that our approach scale well and significantly outperform current Safety LTL synthesis techniques. We conclude that our compositional approach is an important contribution to the algorithmic portfolio of Safety LTL synthesis.

1 Introduction

Reactive synthesis is the automated construction, from a high-level description of its desired behavior, of a reactive system that continuously interacts with an uncontrollable external environment [7]. By describing a system in terms of what it should do, instead of how it should do it, this declarative paradigm holds the promise of correct-by-construction philosophy of program design [26, 32]. We believe that reactive synthesis will be a viable way to create verified software. A popular language for specifying properties that systems should satisfy is Linear Temporal Logic (LTL) [25].

In the last decade, there have been extensive breakthroughs in the study of LTL synthesis [4, 22, 33]. A natural next step is to consider large scale synthesis instances. Many

practical specifications, by and large, are conjunctions of complex but smaller (shorter) *inner* temporal specifications. While the development of techniques for reactive synthesis for these inner formulas remains an active area of research [5, 11, 13, 22, 23], It is fair to combat large-scale practical specifications starting with developing synthesis algorithms for large conjunctions of (inner) temporal formulas.

Previously, large conjunctions, such as strong fairness properties, have been handled successfully in the context of model-checking [1]. One of the cornerstones of scalable model-checking is to represent the model by a *partitioned transition relation*, i.e., the transition relation of the model is represented as a product of smaller transition relations. In model-checking, this representation has been a boon to scale to very large systems. In reactive synthesis, however, this representation has been shown to be a bane to scalability. More specifically, [31] attempts to solve synthesis of large conjunctions by representing the state-space of the final game automaton as a product of the state-space of the game automaton of every inner formula. The issue is that by doing so, the state space of the final game may grow very large, since the algorithm loses the ability to perform fast minimization of the game automaton [35].

On the other hand, compositional approaches have shown promise in synthesis of large conjunctions. Theoretical compositional approaches are well known [12, 19] and implementations that handle large conjunction have been emerging [2, 5, 9, 22]. For example, Lisa [2] successfully scales synthesis to large conjunctions of LTL formulas over finite traces or LTL$_f$ [17] for short. This approach has been further extended to handle large disjunctions in Lydia [9]. Yet, a challenge in these approaches is that the inner formulas cannot be synthesized one after another separately to generate a program for the large conjunction [19]. This is because having correct programs for all inner formulas does not necessarily indicate the existence of a correct program of the large conjunction. To this end, compositional approaches have been deployed to generate the game automaton only, and not to solve the game. The game is solved only after the generation of the complete game arena, which is the main difficulty of synthesis for formulas with large lengths [26, 34].

In this work, we tackle this difficulty by looking into *specialized* compositional synthesis techniques for *Safety* LTL formulas, which is a popular and important fragment of LTL [28, 20, 24]. The key observation is that, for a Safety LTL formula, instead of utilizing its exact game arena when being conjuncted with other formulas, we only need to approximate the partial game arena to ensure the satisfaction of it under all circumstances, hence reducing the state space for subsequent operations. We note that recently, another safety fragment of LTL called *extended bounded response* (EBR) LTL [8] has been shown to be expressively equivalent to Safety LTL, but differs in the syntax of Safety LTL. The conversion from Safety LTL to EBR-LTL may incur blow-up of formula lengths [8], so we only consider Safety LTL here.

The synthesis instances we consider are Safety LTL formulas given in the form of $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$. The Safety LTL fragment and the conjunctive instances together form a special structure, which naturally enables us to develop a more advanced compositional synthesis approach. Indeed, our compositional synthesis technique can apply at two decomposition levels. To begin with, the specification-level decomposition breaks φ into the set of conjuncts $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ and constructs the *deterministic*

safety automaton (DSA) of each conjunct $\varphi_i, 1 \leqslant i \leqslant n$. Meanwhile, inspired by [33], we observe that one can directly consider the negation of Safety LTL φ_i in *negation normal form* (NNF) as an LTL_f formula, a finite-trace variant of LTL that has the same expressiveness power as first-order logic over finite traces [17]. This allows us to utilize LTL_f-to-DFA construction tools integrated with compositional techniques [2, 9], which have been proven outperforming MONA, to obtain the DFA of the bad prefixes of each φ_i, which is simply the dual of the DSA of φ_i. Furthermore, instead of utilizing the partitioned transition relation, which nullifies the benefits of automata minimization, we keep the explicit-state symbolic-transition representation of each DSA to take the maximal advantage of automata minimization, as in [2, 9]. As a result, our compositional approach avoids the straightforward DSA construction from the whole formula φ and performs the DSA construction for each conjunct φ_i separately.

Beyond that, before composing the DSAs to construct the ultimate one, the game-level decomposition splits each DSA into winning part and losing part by conducting a safety game. More specifically, we propose two decomposition versions, that are state-based game-level decomposition and strategy-based game-level decomposition. The state-based decomposition considers the winning part as the set of winning states. It thus trims the DSA by clustering all losing states into a single one and minimizes the resulting DSA. The strategy-based decomposition, instead, considers the winning part as the maximally permissive strategy of the safety game, e.g., a finite-state transducer, that encompasses all the necessary information to ensure the satisfaction of the conjunct under all circumstances [3]. Thereby, it trims the DSA by clustering all states and also *transitions* that do not belong to this strategy. The trimmed DSA is also minimized for subsequent computation. In addition, minimization is applied during every round of composing two DSAs into a product automaton.

We have implemented our compositional synthesis algorithms in a prototype tool called Gelato. To demonstrate the efficiency of our algorithms, we perform an empirical evaluation by comparing Gelato with the monolithic approach, i.e., not leveraging the proposed compositional synthesis technique, and Strix [22], the state-of-the-art LTL synthesis tool. By comparing our compositional approach with a portfolio of other approaches, we observed that our approach was able to solve a notable number of instances that were not solved by others. In particular, experiments on scalable conjunctive benchmarks showed that our approach scale well and significantly outperform current Safety LTL synthesis techniques. We are convinced that our compositional approach is a valuable and important contribution to the current portfolio of Safety LTL synthesis algorithms.

Related Works. There have been several theoretical compositional synthesis approaches and implementations proposed for LTL formulas of the form $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$. In [19], a Safraless compositional approach, inspired by [21], uses generalized co-Büchi *tree* automata to avoid the determinization of Büchi automata and parity condition for obtaining the game arena. This compositional approach checks the realizability of $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$ by first checking the realizability of each sub-formula φ_i with the structure of tree automata rather than DSAs that we use in this work; they try to reuse the result of each conjunct φ_i when checking $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$. To the best of our knowledge, there is no implementation for this approach. This may partially be because tree automata are

not as easy and well studied as word automata, especially in terms of tool support. We note that current practical synthesis tools [5, 11, 13, 23] are all based on *word* automata, just as our algorithm is here.

To make use of word automata, in [14], the authors proposed an algorithm that treats the tree automaton for each conjunct φ_i as a universal co-Büchi word automaton, the game on which can then be solved by a reduction to solving a safety game, based on a given bound of the length of words. When composing the synthesized programs for the conjuncts to obtain a program for the whole formula φ, this algorithm also relies on the computation of the maximally permissive strategy for each safety game as we do in this work; they have implemented the algorithm in the tool Acacia+ [5]. In fact, our strategy-based decomposition variant is inspired by this approach. The difference is that we do not need a given bound for building the safety game, since we focus on Safety LTL formulas, while their algorithm can be incomplete if the given bound is not large enough. Another key difference is that we construct the safety game based on the construction of automata on finite words, while their algorithm builds a universal co-Büchi automaton for each conjunct. This allows us to leverage advanced compositional DFA construction in literature [2, 9], a key to make our algorithm outperform the state of the arts (cf. Section 4).

Another compositional synthesis approach, presented in [29], constructs compositionally a parity game from an LTL formula of the form $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$ based on a variant of Safra's determinization. In addition, this approach tries to detect local parity games that are equivalent to safety games to improve efficiency. As aforementioned, we construct automata on finite words, which is different from the algorithm in [29].

The compositional approach proposed in [15] is based on decomposing the LTL formula into sub-formulas that are independent, such that completely separate synthesis tasks can be performed for them. The approach from [16] first splits the system into components and then proceeds in an incremental fashion such that each component can already assume a particular strategy for the synthesized components. The implementations of both approaches above are not, however, publicly available. We remark that there is a compositional construction of the game arena from LTL formulas [12], which is not involved with the synthesis task.

2 Preliminaries

2.1 LTL/LTL$_f$

Linear Temporal Logic (LTL) [25] is one of the most popular logics for temporal properties. Given a set $\mathcal{P}$ of propositions, the syntax of LTL formulas is defined as:

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid p \mid (\neg p) \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid (\bigcirc \varphi) \mid (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

$$\mid (\varphi_1 \,\mathcal{W}\, \varphi_2) \mid (\varphi_1 \,\mathcal{M}\, \varphi_2) \mid (\varphi_1 \,\mathcal{R}\, \varphi_2).$$

where $p \in \mathcal{P}$ is an *atom*. $\bigcirc$ (Next), $\mathcal{U}$ (Until), $\mathcal{W}$ (Weak Until), $\mathcal{M}$ (Release) and $\mathcal{R}$ (Weak Release) are temporal connectives. We use the abbreviations $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \mathit{false}\,\mathcal{M}\,\varphi$, for temporal connectives $\Diamond$ (Eventually) and $\Box$ (Always).

A *trace* $\pi = \pi_0\pi_1\ldots$ is a sequence of propositional interpretations (sets), where $\pi_m \in 2^{\mathcal{P}}$ ($m \geqslant 0$) is the m-th interpretation of π, and $|\pi|$ represents the length of π. Trace π is an *infinite* trace if $|\pi| = \infty$, which is formally denoted as $\pi \in (2^{\mathcal{P}})^{\omega}$. Otherwise π is a *finite* trace, denoted as $\pi \in (2^{\mathcal{P}})^*$. LTL formulas are interpreted over infinite traces. Given an infinite trace π and an LTL formula φ, we inductively define when φ is $true$ in π at instant i ($i \geqslant 0$), written $\pi, i \models \varphi$, as follows:

- $\pi, i \models true$ and $\pi, i \not\models false$;
- $\pi, i \models a$ iff $a \in \pi_i$ and $\pi, i \models \neg a$ iff $a \notin \pi_i$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$, iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \vee \varphi_2$, iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models \bigcirc\varphi$, iff $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$, iff $\exists k.k \geqslant i$ such that $\pi, k \models \varphi_2$, and $\forall j.i \leqslant j < k$, $\pi, j \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{W}\, \varphi_2$, iff either $\exists k.k \geqslant i$ such that $\pi, k \models \varphi_2$, and $\forall j.i \leqslant j < k$, we have $\pi, j \models \varphi_1$, or $\forall k.k \geqslant i$ we have $\pi, k \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{M}\, \varphi_2$ iff $\exists k.k \geqslant i$ such that $\pi, k \models \varphi_1$, and $\forall j.i \leqslant j \leqslant k$, $\pi, j \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{R}\, \varphi_2$, iff $\exists k.k \geqslant i$ such that $\pi, k \models \varphi_1$, and $\forall j.i \leqslant j \leqslant k$, $\pi, j \models \varphi_2$, or $\forall k.k \geqslant i$ we have $\pi, k \models \varphi_2$.

LTL$_f$ is a variant of LTL interpreted over *finite traces* instead of infinite traces [17]. The syntax of LTL$_f$ is exactly the same to the syntax of LTL. We define $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators:

- $\pi, i \models \bigcirc\varphi$ iff $i < last(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff $\exists j.i \leqslant j \leqslant last(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k.i \leqslant k < j$ we have $\pi, k \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{W}\, \varphi_2$, iff either $\exists k.i \leqslant k \leqslant last(\pi)$ such that $\pi, k \models \varphi_2$, and $\forall j.i \leqslant j < k$, we have $\pi, j \models \varphi_1$, or $\forall k.i \leqslant k \leqslant last(\pi)$ we have $\pi, k \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{M}\, \varphi_2$ iff $\exists k.i \leqslant k \leqslant last(\pi)$ such that $\pi, k \models \varphi_1$, and $\forall j.i \leqslant j \leqslant k$, $\pi, j \models \varphi_1$.
- $\pi, i \models \varphi_1 \,\mathcal{R}\, \varphi_2$, iff $\exists k.i \leqslant k \leqslant last(\pi)$ such that $\pi, k \models \varphi_1$, and $\forall j.i \leqslant j \leqslant k$, $\pi, j \models \varphi_2$, or $\forall k.i \leqslant k \leqslant last(\pi)$ we have $\pi, k \models \varphi_2$.

where we denote the last position in the finite trace π by $last(\pi)$. In addition we define the *weak next* operator $\bullet$ as abbreviation of $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$. Note that, over finite traces, $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$, instead $\neg\bigcirc\varphi \equiv \bullet\neg\varphi$. We say that a trace *satisfies* an LTL$_f$ formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

Without loss of generality, assume the input LTL formulas in Negation Normal Form (NNF), which requires negations only occurring in front of atomic propositions.

2.2 Safety/Co-Safety LTL

Intuitively, a *safety* formula rejects traces whose "badness" follows from a finite prefix. Dually, a *co-safety* formula accepts traces whose "goodness" follows from a finite prefix. We formally refer to these prefixes as bad/good prefixes accordingly. Consider a language $\mathcal{L} \subseteq (2^{\mathcal{P}})^{\omega}$ of infinite traces $\mathcal{P}$. A finite trace $h \in (2^{\mathcal{P}})^*$ is a *bad (resp., good)* prefix for $\mathcal{L}$ iff for all infinite traces $\pi \in (2^{\mathcal{P}})^{\omega}$, we have that $h \cdot \pi \notin \mathcal{L}$ (resp., $h \cdot \pi \in \mathcal{L}$). A language $\mathcal{L}$ is a *safety language* iff every trace that violates (resp., satisfies) φ has a bad (resp., good) prefix. We say that an LTL formula is a safety (co-safety) formula iff $||\varphi||$, i.e., the set of infinite traces that satisfy φ, is a safety (co-safety) language.

We now introduce a fragment of LTL, where safety (resp., for co-safety) is expressed as a syntactical feature, by restricting the occurrences of temporal connectives.

Definition 1 ([6, 27]). *Safety LTL (resp. Co-Safety LTL) formulas are LTL formulas in* NNF *containing only temporal operators such as* $\bigcirc$, $\mathcal{R}$, *and* $\mathcal{W}$ *(resp.,* $\bigcirc$, $\mathcal{M}$, *and* $\mathcal{U}$*).*

Theorem 1 ([6, 27]). *Every safety (resp., co-safety) formula is equivalent to a formula in Safety LTL (resp., Co-Safety LTL).*

Note that, the syntactic fragment of Safety (resp. Co-Safety) LTL in Definition 1 is equivalent to the one defined in [33], which requires that the $\mathcal{U}$ (resp. $\mathcal{R}$) connective does not occur. Specifically, since $\varphi_1 \,\mathcal{W}\, \varphi_2 \equiv ((\bigcirc\varphi_2)\,\mathcal{R}\,\varphi_1) \vee \varphi_2$, and $\varphi_1 \,\mathcal{M}\, \varphi_2 \equiv \varphi_2\,\mathcal{U}(\varphi_1 \wedge \varphi_2)$, every occurrence of $\mathcal{W}$ can be replaced by $\mathcal{R}$ and $\bigcirc$, and every occurrence of $\mathcal{M}$ can be replaced by $\mathcal{U}$, without introducing any extra negations. Thus, every safe (resp., co-safe) formula is equivalent to a Safety (resp., Co-Safety) LTL formula φ.

2.3 Safety LTL Synthesis

Reactive synthesis concerns constructing the behaviors of an *agent* that satisfy a given property while interacting with its *environment* [26]. Formally, a reactive synthesis problem is described as a tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, where $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint sets of variables controlled by the *environment* and the *agent*, respectively, and φ is a linear temporal formula over $\mathcal{X} \cup \mathcal{Y}$ expressing desired properties. A *deterministic* agent strategy is a function $\sigma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$. A *trace* is an infinite sequence $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1) \ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ over the alphabet $2^{\mathcal{X}\cup\mathcal{Y}}$. A trace π is *compatible* with an agent strategy σ, if $\sigma(\epsilon) = Y_0$ and $\sigma(X_0X_1 \ldots X_i) = Y_{i+1}$ for every $i \geqslant 0$, where ϵ denotes empty trace. Analogously, finite prefix $\pi^k = (X_0 \cup Y_0)(X_1 \cup Y_1) \ldots (X_k \cup Y_k)$ is *compatible* with σ if $\sigma(X_0X_1 \ldots X_i) = Y_{i+1}$ for every $0 \leqslant i < k$. Given a synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, an agent strategy σ *realizes* φ if every trace π that is compatible with σ satisfies φ. There are two versions of reactive synthesis, depending on the first player. Here we consider the case where the *agent* moves first; the variant where the environment moves first can be obtained with a minor modification.

In this paper, we focus on the problem of *Safety LTL Synthesis*.

Definition 2 (Safety LTL Synthesis). *The problem of Safety LTL synthesis is described as a tuple* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$*, where* φ *is a Safety LTL formula over* $\mathcal{X} \cup \mathcal{Y}$*. Computing an agent strategy* σ *that realizes* φ *if one exists, is called the Safety LTL synthesis problem.*

The problem of Safety LTL synthesis can be solved by a reduction to *safety games*, which is a two-player game over a so-called *deterministic safety automaton* [33].

Deterministic Safety Automata. A *deterministic safety automaton* (DSA) is a tuple $\mathcal{D} = (2^{\mathcal{P}}, S, s_0, \delta)$, where $2^{\mathcal{P}}$ is the alphabet, S is a finite set of states with s_0 as the initial state, and $\delta : S \times 2^{\mathcal{P}} \rightharpoonup S$ is a partial transition function. Given an infinite trace $\pi \in (2^{\mathcal{P}})^\omega$, the run r of $\mathcal{D}$ on π, denoted by $r = \mathsf{Run}(\mathcal{D}, \pi)$, is a sequence of states $r = s_0s_1s_2 \ldots$ such that $s_{i+1} = \delta(s_i, \pi_i)$ for every $i \geqslant 0$. π is accepted by $\mathcal{D}$ if $r = \mathsf{Run}(\mathcal{D}, \pi)$ is well defined. Note that, δ is a partial function, meaning that given $s \in S$ and $a \in 2^{\mathcal{P}}$, $\delta(s, a)$ can either return a state $s' \in S$ or be undefined. Thus, $r = \mathsf{Run}(\mathcal{D}, \pi)$ may not be an infinite sequence due to the possibility of $\delta(s_i, \pi_i)$ being undefined for some $(s_i, \pi_i) \in S \times 2^{\mathcal{P}}$.

Symbolic DSA. The *symbolic-state* representation of a DSA $\mathcal{D} = (2^{\mathcal{P}}, S, s_0, \delta)$ is a tuple $\mathcal{A} = (\mathcal{S}(\mathcal{Z}), \mathcal{K}(\mathcal{Z}, \mathcal{P}, \mathcal{Z}'))$, where $\mathcal{Z} = \{z_1, \ldots z_n\}$ are propositions encoding the state space S, with $n = \lceil \log |S| \rceil$, and their primed counterparts $\mathcal{Z}' = \{z'_1, \ldots z'_n\}$ encode the next state. Each state $s \in S$ corresponds to an interpretation $Z \in 2^{\mathcal{Z}}$ over propositions $\mathcal{Z}$. When representing the next state of the transition function, the same encoding is used for an interpretation Z' over $\mathcal{Z}'$. Then, $\mathcal{S}$ and $\mathcal{K}$ are Boolean formulas representing s_0 and δ, respectively. $\mathcal{S}(\mathcal{Z})$ is satisfied only by the interpretation of the initial state s_0 over $\mathcal{Z}$. $\mathcal{K}(\mathcal{Z}, \mathcal{P}, \mathcal{Z}')$ is satisfied by interpretations $Z \in 2^{\mathcal{Z}}$, $P \in 2^{\mathcal{P}}$ and $Z' \in 2^{\mathcal{Z}'}$ iff $\delta(s, P) = s'$, where s and s' are the states corresponding to Z and Z'.

Safety Games. A safety game is defined as a tuple $\mathcal{G} = (\mathcal{X}, \mathcal{Y}, \mathcal{D})$, where $\mathcal{D} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \delta)$ is a DSA, and $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint sets of variables, controlled by the environment, and the agent, respectively. A trace $\pi \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ is *winning* for the agent if $r = \mathsf{Run}(\mathcal{D}, \pi)$ is accepted by $\mathcal{D}$. An agent strategy σ is winning if every trace π that is compatible with σ is a winning play. *Solving a DSA game* aims to computing an agent winning strategy if one exists. A state $s \in S$ is *winning* for the agent if there exists an agent strategy such that all traces beginning in s are winning for the agent. The winning set of a DSA is the set of all winning states of the agent. To compute the winning set of $\mathcal{G}$ the following fixed-point computation is performed:

$$\mathsf{Win}_0 = S;$$
$$\mathsf{Win}_{i+1} = \mathsf{Win}_i \cap \{s \in S \mid \exists Y \forall X . \delta(s, X \cup Y) \in \mathsf{Win}_i\}.$$

Clearly, a safety game $\mathcal{G}$ can be analyzed by checking whether the initial state s_0 is a winning state, in which case we say that $\mathcal{G}$ is *realizable*. Next, we see that for safety games there exists *maximally permissive strategies* [3].

Maximally Permissive Strategies. Different definitions of maximally permissive strategies exist. In this work we refer to the definition in [3], where strategies are compared by looking at inclusion of the behaviors/outcomes they allow.

Definition 3 (Non-Deterministic Strategy). *A* non-deterministic *strategy for the agent is defined as a function $\alpha : (2^{\mathcal{X}})^* \rightarrow 2^{2^{\mathcal{Y}}}$. The set of deterministic strategies induced by a non-deterministic strategy α is the set*

$$[[\alpha]] = \{\sigma : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}} | \sigma(h) \in \alpha(h), \textit{ for } h \in (2^{\mathcal{X}})^*\}.$$

Definition 4 (Maximally Permissive Strategy). *A non-deterministic strategy α is at least as permissive as α' if $[[\alpha']] \subseteq [[\alpha]]$. A non-deterministic strategy α is a maximally permissive strategy if $[[\alpha']] \subseteq [[\alpha]]$, for every non-deterministic strategy α'.*

Theorem 2 ([3]). *Let $\mathcal{G}$ be a safety game. We have that if $\mathcal{G}$ is realizable, then $\mathcal{G}$ has a maximal permissive strategy that is* memoryless*, i.e., $\alpha : S \rightarrow 2^{2^{\mathcal{Y}}}$.*

3 Compositional Approaches for Safety LTL Synthesis

3.1 From Safety LTL to DSA

Consider a Safety LTL formula φ, since every trace rejected by its corresponding DSA $\mathcal{D}^s_\varphi$ can be rejected in a finite number of steps, we can alternatively define the language

accepted by $\mathcal{D}^s_\varphi$ by the finite prefixes that it rejects [20]. Therefore, the DSA construction can be achieved by first obtaining the DFA $\mathcal{D}^f_\varphi$ that accepts all the bad prefixes of φ, and then complementing it, which gives us the DSA of φ [33]. The construction shown in [33] is processed as follows: given a Safety LTL formula φ, first negate it to obtain a Co-Safety LTL formula $\neg\varphi$, then translate it into a first-order logic formula $fol(\neg\varphi)$. The DFA of $fol(\neg\varphi)$ is able to accept exactly the set of bad prefixes for φ, and can be constructed using MONA [18], a DFA construction tool from logic specifications.

Note that the key step here is to leverage the technique and tools developed for constructing $\mathcal{D}^f_\varphi$. To do so, we make use of LYDIA [9], which has shown better performance than MONA. In particular, this change does not require the explicit translation to first-order logic. Instead, we can directly consider the Co-Safety formula $\neg\varphi$ as an LTL$_f$ formula, and give it to LYDIA as input. The returned automaton is the DFA that accepts all the good prefixes of $\neg\varphi$, e.g., the bad prefixes of φ.

Theorem 3. *Let ψ be a Co-Safety LTL formula in* NNF*, φ the same formula as ψ, but in LTL$_f$, and π a finite trace. Then π is good prefix of ψ iff $\pi \models \varphi$.*

Proof. We prove it by induction over the structure of φ.

- Base case, if $\psi = p$ is an atom, π is a good prefix for ψ iff $p \in \pi_0$. By definition of φ, we have that $\pi \models \varphi$. If $\psi = \neg p$, π is a good prefix of ψ iff $p \notin \pi_0$, then $\pi \not\models \varphi$.
- If $\psi = \psi_1 \wedge \psi_2$, π is a good prefix for ψ implies π is a good prefix for both ψ_1 and ψ_2. By induction hypothesis, $\pi \models \varphi_1$ and $\pi \models \varphi_2$, where φ_1 and φ_2 are defined as ψ_1 and ψ_2, respectively, in LTL$_f$. Then, we have that $\pi \models \varphi_1 \wedge \varphi_2$.
- If $\psi = \psi_1 \vee \psi_2$, π is a good prefix for ψ implies π is a good prefix for either ψ_1 or ψ_2. Without loss of generality, suppose π is a good prefix for ψ_1. By induction hypothesis, $\pi \models \varphi_1$ where φ_1 is the LTL$_f$ formula defined as ψ_1. Then, $\pi \models \varphi_1 \vee \varphi_2$, with φ_2 defined in LTL$_f$ as ψ_2.
- If $\psi = \bigcirc\psi_1$, π is a good prefix for ψ iff suffix $\pi' = \pi_1\pi_2 \ldots, \pi_{|\pi|-1}$ of π is a good prefix for ψ_1. By induction hypothesis, $\pi', 1 \models \varphi_1$ where φ_1 is defined as ψ_1 in LTL$_f$. Then, we have that $\pi \models \bigcirc\varphi_1$.
- If $\psi = \psi_1 \,\mathcal{U}\, \psi_2$, π is a good prefix for ψ iff there exists $0 \leqslant i \leqslant |\pi| - 1$ such that suffix $\pi' = \pi_i\pi_{i+1}, \ldots, \pi_{|\pi-1|}$ of π is a good prefix for ψ_2, and for all $0 \leqslant j < i$, $\pi'' = \pi_j\pi_{j+1}, \ldots, \pi_{i-1}$ is a good prefix for ψ_1. By induction hypothesis, $\pi', i \models \varphi_2$ where φ_2 is defined as ψ_2 in LTL$_f$, and $\pi'', j \models \varphi_1$ with φ_1 defined as ψ_2 in LTL$_f$. Therefore, $\pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$.
- The cases for $\mathcal{W}$, $\mathcal{M}$, and $\mathcal{R}$ are derived from the above.

3.2 Compositional Safety LTL Synthesis

The crux of our compositional approach is to avoid the DSA construction of the complete Safety LTL formula φ by performing the DSA construction for each conjunct φ_i, and, most importantly, solving the safety game over the DSA before composing it with the other DSAs. We first propose a compositional approach based on the computation of the agent winning states of safety games. In particular, inspired by the compositional automata construction technique presented in [2], we also employ here the explicit-DSA to symbolic-DSA switch heuristics to achieve promising practical benefits.

State-Based Compositional Approach. After checking realizability of each corresponding safety game of φ_i, we prune the safety game with respect to the winning states and then minimize the game; the algorithm then goes through a phase of combination of two DSAs, minimization of the combined DSA, solving the safety game over the DSA, and pruning the game again, until a switch to a symbolic representation occurs. When we have switched to using the symbolic representation for DSAs, we will not perform minimization operation on the symbolic DSA since it is time-consuming because of large DSA state space; instead, in each round we only combine the DSAs and solve the safety game over the corresponding DSA. Specifically, given a Safety LTL formula in the form of $\varphi = \bigwedge_{1 \leqslant i \leqslant n} \varphi_i$, and *switch-over threshold values* $t_1, t_2 > 0$ that represent the thresholds for the numbers of states in an individual DSA and in the product of two DSAs, respectively, to trigger the symbolic representation, the algorithm proceeds as follows.

1. **Decomposition.** Construct minimal DSA $\mathcal{D}_i$ for each sub-formula φ_i of φ in explicit-state representation as described in Section 3.1 and let $H_1 = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$. Then, for all $i \in \{1, \ldots, n\}$,
   (a) compute the winning set W_i of the agent in the safety game $\mathcal{G}_i = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_i \rangle$. Return φ is unrealizable if $\mathcal{G}_i$ is unrealizable.
   (b) Prune $\mathcal{D}_i$ such that only the states in W_i are retained. Formally, let $\mathcal{D}_i = (\Sigma, S, s_0, \delta)$. Then prune $\mathcal{D}_i$ with respect to W_i obtaining $\mathcal{D}_i^w = (\Sigma, W_i, s_0, \delta^w)$ where the transition function δ^w is defined as follows:
   $$\delta^w(s, \sigma) = \begin{cases} \delta(s, \sigma) & \text{if } \delta(s, \sigma) \in W_i, \\ \text{undefined} & \text{if } \delta(s, \sigma) \notin W_i. \end{cases}$$
   (c) Minimize $\mathcal{D}_i^w$. Note that, since DSAs are represented as DFAs, the pruning step is performed on DFAs, and therefore we can apply minimization techniques on DFAs to obtain the minimal DSA.
2. **Explicit-state composition.** For $j \in \{1, \ldots, n-1\}$, let $H_j = \{\mathcal{D}_1 \ldots \mathcal{D}_{n-j+1}\}$ be the set of DSAs in the j-th iteration. If H_j has only one DSA $\mathcal{D}_1$, then return a winning strategy for the agent in $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_1 \rangle$. Otherwise, pick from H_j two DSAs, $\mathcal{D}_1$ and $\mathcal{D}_2$, chosen by the dynamic smallest-first heuristic [2] which always returns two DSAs in H_j with the smallest number of states. This allows to find an order that can optimize time and space in the composition phase. Indeed, if the algorithm would fail on the composition of the smallest two DFAs in that iteration, then it would probably fail on the composition of all other pairs of DFAs as well. Let $|\mathcal{D}|$ be the number of states in a DSA $\mathcal{D}$ represented in explicit-state form. If $|\mathcal{D}_1| > t_1$ or $|\mathcal{D}_2| > t_1$, or $(|\mathcal{D}_1| \cdot |D_2|) > t_2$, then change state representation moving to Step 3 and let k be the iteration in which this occurs, i.e., take $k = j$. If not, continue with the explicit-state representation and perform the following steps.
   (a) Construct $\mathcal{D}_{1,2} = \mathcal{D}_1 \cap \mathcal{D}_2$, and minimize $\mathcal{D}_{1,2}$ to generate $\mathcal{D}$.
   (b) Compute the winning set W of the agent in safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D} \rangle$. Return φ is unrealizable if $\mathcal{D}$ is unrealizable.
   (c) Prune $\mathcal{D}$ such that only the states in W are retained (see Step 1(b)), and minimize it. Then, create $H_{j+1} = \{\mathcal{D}, \mathcal{D}_3 \ldots \mathcal{D}_{n-j+1}\}$.
   (d) Go to Step 2.

3. **Change state representation.** Convert all DSAs in $H_k = \{\mathcal{D}_1, \ldots, \mathcal{D}_{n-k+1}\}$ from explicit-state to symbolic-state representation, and proceed to Step 4. Note that the state space of each DSA $\mathcal{D}_i$ is encoded symbolically using a different set of state variables $\mathcal{Z}_i$, where all $\mathcal{Z}_i$ are disjoint. Since no more minimization occurs after this point, the total set of state variables $\mathcal{Z} = \mathcal{Z}_1 \cup \ldots \cup \mathcal{Z}_{n-k+1}$ defines the state space of the final DSA.
4. **Symbolic-state composition.** For $j \in \{k, \ldots, n\}$, let $H_j = \{\mathcal{D}_1, \ldots, \mathcal{D}_{n-j+1}\}$ be the set of DSAs in the j-th iteration. If H_j has only one DSA, return a winning strategy for the agent, otherwise return φ is unrealizable. Otherwise, assume w.l.o.g. that $\mathcal{D}_1$ and $\mathcal{D}_2$ are the two DSAs chosen by the DSF heuristic and perform the following steps:
   (a) Construct $\mathcal{D} = \mathcal{D}_1^w \cap \mathcal{D}_2^w$. Recall that, since $\mathcal{D}_1$ and $\mathcal{D}_2$ are in symbolic form, we do not perform DSA minimization of $\mathcal{D}_{1,2}$.
   (b) Compute the winning set W of the agent in the safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D} \rangle$. Return φ is unrealizable if any of the two $\mathcal{G}$ is unrealizable. Then, create $H_{j+1} = \{\mathcal{D}, \mathcal{D}_3 \ldots \mathcal{D}_{n-j+1}\}$.
   (c) Go to Step 4.

To prove the correctness of the algorithm described above, i.e., to prove that the algorithm correctly evaluates realizability of the input safety formula φ and synthesizes a valid winning strategy (if realizable), we make use of the following result.

Lemma 1. *Let $\mathcal{D}$ be a DSA with winning set W for the agent player in the safety game played over $\mathcal{D}$. Let $\mathcal{D}^w$ be the pruning of $\mathcal{D}$ w.r.t. W, as described above. Then, every winning strategy in the safety game over $\mathcal{D}$ is a winning strategy in the safety game over $\mathcal{D}^w$, and vice-versa.*

Proof. We begin by showing that every winning strategy in the safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D} \rangle$ is also a winning strategy in the safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}^w \rangle$.

Let $\sigma : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ be a strategy. Let $\pi_\sigma = (X_0, \sigma(\epsilon)), (X_1, \sigma(X_0)), \ldots, (X_n, \sigma(X_0, X_1, \ldots X_{n-1}))$ be a play of finite-length induced by σ. Given a DSA $\mathcal{D}' = (\mathcal{X} \cup \mathcal{Y}, S, s_0, \delta)$, let s_σ be the unique state in which the run of π_σ beginning in s_0 in DSA $\mathcal{D}'$ terminates. We will show that when σ is a winning strategy for the agent, then the terminal state s_f of the run of all finite plays π_σ is such that $s_\sigma \in W$.

By means of contradiction, suppose σ is a winning strategy such that there exists a finite play π_σ such that the terminal state of its run in DSA $\mathcal{D}$ is $s_\sigma \in S \backslash W$. Then, since DSAs are determined games and both players have memoryless winning strategies, the environment can begin executing a memoryless environment winning strategy from s_σ. Then, by definition of winning strategies of the environment, this ensures that every resulting play is winning for the environment. Thus, we have a contradiction.

This shows that every winning strategy σ of the agent in the safety game over $\mathcal{D}$ can be executed in a game over $\mathcal{D}^w$ since $\mathcal{D}^w$ is defined over the winning set of $\mathcal{D}$. Finally, since $\delta^*(\pi_\sigma) \in W$ in DSA $\mathcal{D}$ for all π_σ, we get that $(\delta^w)^*(\pi_\sigma) \in W$ in DSA $\mathcal{D}^w$ for all π_σ, where δ^* and $(\delta^w)^*$ are the transitive closures of δ and δ^w. Thus, σ is also a winning strategy in safety game played over $\mathcal{D}^w$ as it never encounters an undefined transition in $\mathcal{D}^w$.

Next, we show that a strategy that is not winning for the agent in a safety game over $\mathcal{D}$ is also not a winning strategy for the agent in the safety game over $\mathcal{D}^w$. The proof

for this is the dual of the earlier case. Here we will show that for strategies that are not winning for the agent, the terminal state of the run of every finite-play in $\mathcal{D}$ lies in $S \backslash W$. Then, it is easy to see that these strategies will encounter an undefined transition in the game over $\mathcal{D}^w$. Meaning, that the strategy is not winning for the agent in the safety game over $\mathcal{D}^w$.

Theorem 4. *The state-based compositional approach is sound and complete for Safety LTL synthesis.*

Proof. Clearly, σ is a winning strategy for the agent for the input formula φ iff σ is a winning strategy in every DSA in H_1. Suppose $\mathcal{D}_1$ and $\mathcal{D}_2$ are chosen in the first iteration of the algorithm. Then, by Lemma 1, since winning strategies are preserved via pruning, we get that σ is a winning strategy in every DSA in $H_1 \backslash \{\mathcal{D}_1, \mathcal{D}_2\} \cup \{\mathcal{D}_1^w, \mathcal{D}_2^w\}$. Since σ is a winning strategy in both $\mathcal{D}_1^w$and $\mathcal{D}_2^w$, σ is a winning strategy for $\mathcal{D}_1^w \cap \mathcal{D}_2^w$. Since the language of $\mathcal{D}_1^w \cap \mathcal{D}_2^w$ is equivalent to that of its minimal DSA $\mathcal{D}_{1,2}$, we get that σ is also a winning strategy in $\mathcal{D}_{1,2}$. Thus, σ is a winning strategy for the input formula iff σ is a winning strategy in every DSA in H_2.

By repeated application of this argument, we show that σ is a winning strategy for the input formula iff it is a winning strategy over the single DSA in H_n.

It should be noted that when pruning each DSA, the state-based decomposition approach focuses only on winning states and therefore trims the DSAs by clustering all losing states into a single one and minimizes the resulting DSA. Nevertheless, certain transitions, though leading to winning states, do not contribute to the realizability of the conjunct since such transitions do not belong to the maximally permissive strategy of the safety game, e.g., a finite-state transducer that encompasses all the necessary information to ensure the satisfaction of the conjunct under all circumstances [3]. Furthermore, trimming also these transitions might result in an even smaller DSA. We now give a compositional approach based on the computation of the maximally permissive strategy of safety games over DSAs.

Strategy-Based Compositional Approach. Unlike the state-based approach, in each round, it trims from the DSAs not only all states but also *transitions* that do not belong to the maximally permissive strategy. The algorithm proceeds as follows.

1. **Decomposition.** Let $\mathcal{D}_1 \dots \mathcal{D}_n$ be the minimal DSAs for each sub-formula φ_i of the input formula φ in the explicit-state representation as described in Section 3.1. Then, for all $i \in \{1, \dots, n\}$, proceed as follows.
   (a) Compute the set of winning states W_i in the safety game $\mathcal{G}_i = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_i \rangle$. Return φ unrealizable if $\mathcal{G}_i$ is unrealizable.
   (b) Compute the maximally permissive strategy α_i based on the set of winning states W_i. To do so, we define a strategy generator, which is a nondeterministic transducer $\mathcal{T} = (2^{\mathcal{X} \cup \mathcal{Y}}, W_i, s_0, \varrho, \tau)$, where
      – $W_i \subseteq S$ is the set of winning states;
      – $\tau : W_i \to 2^{(2^{\mathcal{Y}})}$ is the output function such that
      $$\tau(s) = \begin{cases} \{Y \mid \forall X. \delta(s, X \cup Y) \in W_i\} & \text{if } s \in W_i, \\ \varnothing & \text{otherwise.} \end{cases}$$

– $\varrho : W_i \times 2^{\mathcal{X}} \to 2^{W_i}$ is the transition function such that $\varrho(s, X) = \{s' \mid s' = \delta(s, X \cup Y)\ and\ Y \in \tau(s)\}$;

This transducer represents the maximally permissive strategy $\alpha : (2^{\mathcal{X}})^* \to 2^{2^{\mathcal{Y}}}$ in the following way: $\alpha(\epsilon) = \tau(s_0)$, and $\alpha(\xi^k) = \tau(s_{k+1})$ for every $\xi^k \in (2^{\mathcal{X}})^+$, where s_{k+1} is the ending state of $\mathsf{Run}(A, \pi^k) = s_0 s_1 s_2 \dots s_k$, $\pi^k = (X_0 \cup Y_0)(X_1 \cup Y_1) \dots (X_k \cup Y_k)$ and $Y_k \in \alpha(\xi^{k-1})$.

(c) Prune $\mathcal{D}_i$ according to α_i. Intuitively, this pruning trims all states and transition that do not belong to α_i, unlike the state-based approach which only cuts states. Let $\mathcal{D} = (\Sigma, S, s_0, \delta)$ be a DSA. We prune $\mathcal{D}$ with respect to $T = (\Sigma, W, s_0, \varrho, \tau)$ such that obtaining $\mathcal{D}^t = (\Sigma, W, s_0, \delta^t)$, where transition function δ^t is defined as follows:

$$\delta^t(s, X \cup Y) = \begin{cases} \delta(s, X \cup Y) & \text{if } Y \in \tau(s), \\ \text{undefined} & \text{if } Y \notin \tau(s). \end{cases}$$

(d) Minimize $\mathcal{D}_i^t$, and create $R = \{\mathcal{D}_1^t, \dots, \mathcal{D}_i^t\}$.

2. **Explicit-state composition.** For $j \in \{1, \dots, n-1\}$, let $R_j = \{\mathcal{D}_1 \dots \mathcal{D}_{n-j+1}\}$ be the set of DSAs in the j-th iteration. If R_j has only one $\mathcal{D}_1$, then return a deterministic strategy for the agent. Otherwise, pick from R_j two DSAs, $\mathcal{D}_1$ and $\mathcal{D}_2$, chosen by the DSF heuristic. If $|\mathcal{D}_1| > t_1$ or $|\mathcal{D}_2| > t_1$, or $(|\mathcal{D}_1| \cdot |D_2|) > t_2$, then change state representation moving to Step 3 and let k be the iteration in which this occurs, i.e., take $k = j$.

   If not, continue with explicit-state representation as follows.

   (a) Compute Step 2(a) and 2(b) as for the state-based approach, obtaining the DSA $\mathcal{D}$, which is the minimal DSA of $D_{1,2}^t = \mathcal{D}_1 \cap \mathcal{D}_2$, and the winning set W.

   (b) Compute maximally permissive strategy α based on W (see Step 1(b)).

   (c) Prune $\mathcal{D}$ in according to α (see Step 1(c)), obtaining D^t, and minimize it. Then, create $R_{j+1} = \{\mathcal{D}, \mathcal{D}_3 \dots \mathcal{D}_{n-j+1}\}$.

   (d) Go to Step 2.

3. Step 3 and 4 are performed as Step 3 and 4 of the state-based approach.

Lemma 2. *Let $\mathcal{D}$, W and $\mathcal{D}^t$ be as above, then the agent has a winning strategy in the safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D} \rangle$ iff the agent has a winning strategy in the safety game $\mathcal{G}^t = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}^t \rangle$.*

Proof. The proof follows Lemma 1.

Lemma 3. *The agent has a winning strategy in safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_{i,j}^t \rangle$ iff $\varphi_i \wedge \varphi_j$ is realizable.*

Proof. We first obverse that $\varphi_i \wedge \varphi_j$ is realizable iff there exists an agent strategy σ that is winning in both safety games $\mathcal{G}_i = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_i \rangle$ and $\mathcal{G}_j = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}_j \rangle$, where $\mathcal{D}_i$ and $\mathcal{D}_j$ are the DSAs for φ_i and φ_j, respectively. By Lemma 2, we know that the σ is also winning in both safety games over $\mathcal{D}_i^t$ and $\mathcal{D}_j^t$, and then it is also a winning strategy for $\mathcal{D}_{i,j}^t = \mathcal{D}_i^t \cap \mathcal{D}_j^t$, as required.

Theorem 5. *The Safety LTL synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, where $\varphi = \bigwedge_{1 \leqslant i \leqslant n} \varphi_i$, is realizable iff the agent has a winning strategy in safety game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \mathcal{D}^t \rangle$, where $\mathcal{D}^t$ is the last DSA obtained executing the strategy-based compositional approach.*

Proof. We can prove it by repeatedly applying Lemma 3, then we have that σ is an agent winning strategy for the input formula φ iff it is a winning strategy in the safety game over the single DSA $\mathcal{D}^t$.

4 Experimental Evaluation

4.1 Implementation

We implemented our two compositional synthesis approaches described in Section 3.2 in a prototype tool Gelato, on top of the Safety LTL synthesis tool SSyft [33]. We first use SPOT [10] to parse the input Safety LTL formula φ in the form of $\varphi_1 \wedge \cdots \wedge \varphi_k, k \geqslant 1$ and then call LYDIA [9] to obtain the DSAs for the smaller Safety LTL conjuncts $\varphi_i, 1 \leqslant i \leqslant k$. Note that all the explicit-state DSAs are, in fact, stored with their corresponding bad-prefixes DFAs. In this way, we can exploit the advanced compositional approach in LYDIA for constructing the DFAs of bad prefixes from small Safety LTL conjuncts. We then employ MONA for the minimization, state-pruning/strategy-prunning and product operations for explicit-state DSAs by operating on their bad-prefixes DFAs. Note that Gelato needs to take switch-over thresholds t_1, t_2 from explicit-states to symbolic-states representations and then performs synthesis on symbolic-state DSAs [34], where CUDD 3.0.0 [30] is used as the BDD library. The thresholds t_1 and t_2 are empirically set to 700 and 1500, respectively, in all experiments. We use native support of SSyft for solving safety game over symbolic DSAs and extracting the winning strategies if φ is realizable; we refer to [33] for more details.

4.2 Experimental Methodology

We compare our tool Gelato with two state of the art tools, namely SSyft, the synthesis tool dedicated for *Safety* LTL [33], and Strix (version 21.0.0) [22], the state-of-the-art synthesis tool for *general* LTL. In particular, we optimize SSyft by using LYDIA rather than MONA to construct the DSA, which highly speeds up the performance of SSyft used in [33]. Experiments were run on a computer cluster, where each instance took exclusive access to a computing node with Intel-Xeon processor running at 2.6 GHz, with 8GB of memory and 30 minutes of time limit.

We consider large-scale Safety LTL synthesis instances in the form of $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$. We collected in total 2,500 Safety LTL synthesis instances, consisting of 1,250 instances from the *Conjunction* benchmark family and 1,250 instances from the *Random-Conjunction* benchmark family. Since Strix only supports the synthesis setting where the environment acts first, the instances taken by them had to be modified slightly to add a $\bigcirc$ (Next) operator in front of all environment variables. The *Conjunction* benchmark family has 1,250 instances that are constructed from basic cases taken from Safety LTL synthesis datasets [33]. In particular, these basic cases are Safety LTL formulas splitting into 5 categories. Every category i $(1 \leqslant i \leqslant 5)$ consists of a set of Safety LTL formulas with i nesting $\bigcirc$ (Next) operators. Indeed, the more nesting $\bigcirc$ operators are, the more difficult the basic case is. In order to evaluate the performance on scalability of handling conjunction formulas, for every category of Safety LTL formulas, we obtain 50 scalable conjunction instances by increasing the number of conjuncts

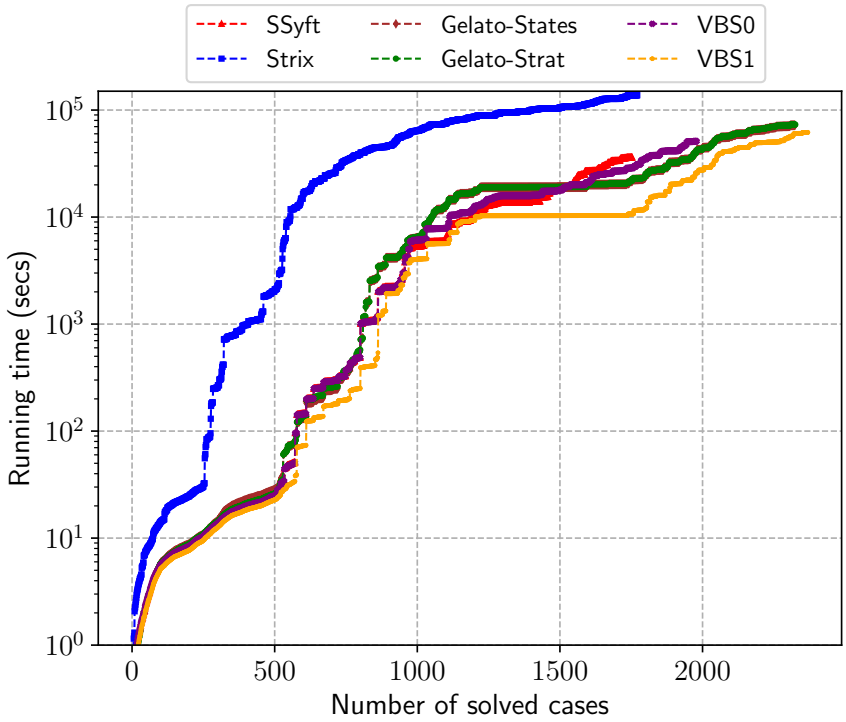

Fig. 1: Cactus plot indicating number of benchmarks solved by each tool over time.

from 1 to 5. The *Random-Conjunction* benchmark family also has 1,250 instances that are constructed in the similar way as the *Conjunction* instances. The key difference is that, all the variables in the randomly conjuncted formula are chosen randomly from a set of 20 candidate variables. Moreover, if a variable v is an environment-variable in the basic case, then the replacement variable v' of v is also an environment-variable in the randomly conjuncted formula. The same applies to the agent-variables.

We have evaluated the results from Gelato with those from Strix and SSyft, and we only find consistent results for the commonly solved cases.

4.3 Results

We denote the winning states-based variant and the winning strategy-based variant of our algorithm in Section 3.2 by Gelato-States and Gelato-Strat, respectively. We compare both Gelato-States and Gelato-Strat against SSyft and Strix in terms of the number of solved cases and the running time. Additionally, we also consider two *virtual best solvers*, VBS0 (two existing tools, *without* Gelato) and VBS1 (*all* implementations).

The cactus plot in Figure 1 reports how many benchmarks solved by each tool over time; we do not show the part where the running time is below 1 second for clarity. We can see that Gelato-Strat only has a slight advantage comparing to Gelato-States, with

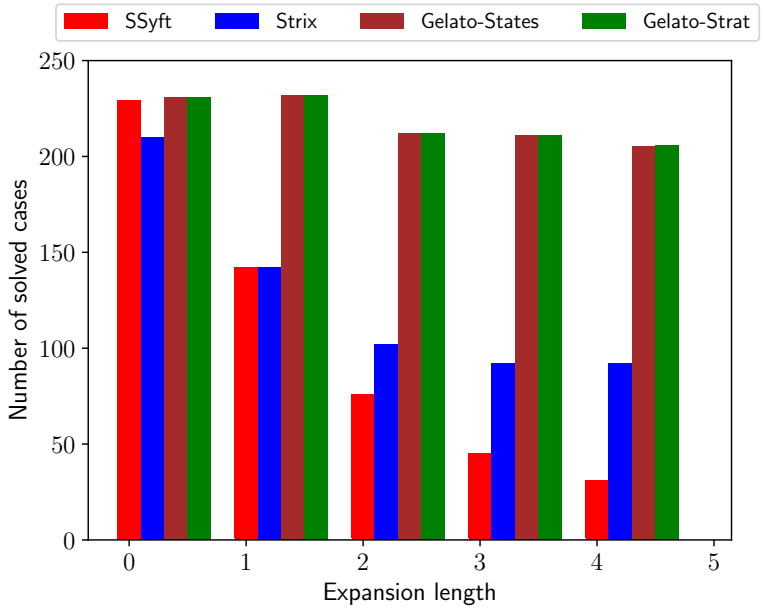

Fig. 2: Number of solved cases for different number of conjunctions in Conjunction benchmarks

Gelato-Strat solving 2,325 cases and Gelato-States 2,324 cases, out of a total 2,500 cases. Regarding the number of solved cases, the performance of Gelato-Strat is significantly better than SSyft and Strix, since they only manage to solve 1,753 and 1,771 cases, respectively. In particular, Strix solved 554 cases less than Gelato-Strat did while taking more time to solve as many instances as both implementations in Gelato. This is reasonable since Strix considers the whole set of LTL while Gelato is carefully designed for big conjunctions of Safety LTL formulas. It is clear to see that our Gelato-Strat has the *best* performance regarding the number of solved cases within the same time limit. Between two virtual best solvers, VBS1 is significantly better than VBS0, with 2,369 cases solved by VBS1 and 1,979 cases by VBS0. It is worth mentioning that both our implementations Gelato-Strat and Gelato-States perform even better than VBS0.

The experimental results showed that our approach can solve a notable number of instances that cannot be managed by existing tools. Therefore, we believe that our compositional algorithm is a *valuable* and *important* contribution to the current portfolio of Safety LTL synthesis approaches.

On a closer inspection, we observe that Gelato-Strat and Gelato-States have a bigger advantage over SSyft and Strix for Conjunction benchmarks than they do for Random-Conjunction benchmarks. For Random-Conjunction benchmarks, Gelato-Strat and Gelato-States solve 3 cases more than SSyft and 100 more than Strix; while for Conjunction benchmarks, Gelato-Strat and Gelato-States solve 1,092 and 1,091 cases, respectively, which are approximately twice as many as those of SSyft and Strix. This may due to the fact that our pruning operation in the synthesis procedure reduces more

state space of the intermediate programs from the Conjunction benchmarks than those from the Random-Conjunction cases.

Figure 2 shows the number of solved cases of all tools for different numbers of conjuncts in Conjunction benchmarks. From Figure 2, we can see that the advantage of Gelato-Strat and Gelato-States over SSyft and Strix gets larger as the expansion length (i.e., the number of conjunctions) grows. This is because constructing DSAs by LYDIA for each small conjunct in Gelato does not get much harder as the length increases; it is more dependent on the size of the small conjunct formulas than the number of conjuncts. In contrast, the performance of SSyft, which relies on LYDIA to construct the DSA for the whole formula, decreases greatly when the expansion length grows. Moreover, we observe that the performance of Gelato does not vary too much when the expansion length grows. This confirms that our compositional synthesis approaches indeed can mitigate the difficulty encountered by other approaches that solve the game only after obtaining the game arena. We also observe similar performance trend of each tool when the expansion length grows for Random-Conjunction benchmarks, except that the advantage of our Gelato-Strat and Gelato-States over SSyft and Strix is not as significant as depicted in Figure 2.

Finally, we compared the running time of Gelato-States and Gelato-Strat on all benchmarks. It is surprising to see that Gelato-States is competitive with Gelato-Strat in general, although, Gelato-Strat solves one more case than Gelato-States and performs better than Gelato-States in hard cases. Gelato-Strat in particular, was expected to benefit from the fact that the transducer of the maximally permissive strategy is supposed to more compact than the one of the winning states. Indeed, both transducers have the same number of states, thus leaving no state space to prune for Gelato-Strat. Nevertheless, the transducer of the maximally permissive strategy should contain fewer propositional evaluations on the transitions. However, this does not lead to a more compact transducer when the transducers are in an explicit-state symbolic-transition representation. On the one hand, the transition conditions are represented symbolically in BDDs, it is possible that removing evaluations that do not belong to the maximally permissive strategy generate larger BDDs. On the other hand, removing evaluations even bring an extra cost. Thereby, we can not expect significant advantage of applying the strategy-based compositional approach.

5 Conclusion

We presented a novel compositional synthesis technique *specialized* for Safety LTL formulas in the form of $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_2$. In contrast to extant compositional synthesis approaches that solve the game after obtaining the game arena, our algorithm synthesizes a program for each smaller conjunct $\varphi_i, 1 \leqslant i \leqslant n$ separately and then composes them one by one. A big advantage of our algorithm is that the intermediate programs will be made smaller with pruning techniques, mitigating the possibility of blow-up of program state space. Empirical evaluation shows that our proposed algorithm outperforms the state of the arts in terms of the number of solved cases and running time. We believe that our compositional approach is a valuable contribution to the portfolio of Safety LTL synthesis algorithms. As future work, we plan to study how

to further improve the construction of DSAs for each conjunct, which is the current bottleneck of our approach. Alternatively, we can investigate how to decompose the specification better to obtain smaller conjunct formulas. It is also interesting to see how our approach performs on practical benchmarks. We leave this to future work as well.

Acknowledgements 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), the National Natural Science Foundation of China (Grant Nos. 62102407 and 61836005), CAS grant QYZDB-SSW-SYS019, NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CNS-2016656, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

1. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
2. Bansal, S., Li, Y., Tabajara, L.M., Vardi, M.Y.: Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications. In: AAAI. pp. 9766–9774 (2020)
3. Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. RAIRO Theor. Informatics Appl. 36(3), 261–275 (2002)
4. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. 78(3), 911–938 (2012)
5. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: CAV. pp. 652–657 (2012)
6. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: ICALP. pp. 474–486 (1992)
7. Church, A.: Application of recursive arithmetic to the problem of circuit synthesis. Journal of Symbolic Logic 28(4), 289–290 (1963)
8. Cimatti, A., Geatti, L., Gigante, N., Montanari, A., Tonetta, S.: Expressiveness of extended bounded response LTL. In: GandALF 2021. pp. 152–165 (2021)
9. De Giacomo, G., Favorito, M.: Compositional approach to translate LTL_f/LDL_f into deterministic finite automata. In: ICAPS. pp. 122–130 (2021)
10. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — A Framework for LTL and ω-automata Manipulation. In: ATVA. pp. 122–129 (2016)
11. Ehlers, R., Raman, V.: Slugs: Extensible GR(1) synthesis. In: CAV. pp. 333–339 (2016)
12. Esparza, J., Kretínský, J., Sickert, S.: From LTL to deterministic automata - A Safraless compositional approach. Formal Methods Syst. Des. 49(3), 219–271 (2016)
13. Faymonville, P., Finkbeiner, B., Tentrup, L.: Bosy: An experimentation framework for bounded synthesis. In: CAV. pp. 325–332 (2017)
14. Filiot, E., Jin, N., Raskin, J.: Antichains and compositional algorithms for LTL synthesis. Formal Methods Syst. Des. 39(3), 261–296 (2011)
15. Finkbeiner, B., Geier, G., Passing, N.: Specification decomposition for reactive synthesis. In: NFM. pp. 113–130 (2021)
16. Finkbeiner, B., Passing, N.: Dependency-based compositional synthesis. In: ATVA. pp. 447–463 (2020)
17. Giacomo, G.D., Vardi, M.Y.: Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In: IJCAI. pp. 854–860 (2013)
18. Henriksen, J.G., Jensen, J.L., Jørgensen, M.E., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: Mona: Monadic Second-order Logic in Practice. In: TACAS. pp. 89–110 (1995)

19. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: CAV 2006. pp. 31–44 (2006)
20. Kupferman, O., Vardi, M.Y.: Model Checking of Safety Properties. Formal Methods in System Design 19(3), 291–314 (2001)
21. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS. pp. 531–542 (2005)
22. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: CAV. pp. 578–586 (2018)
23. Michaud, T., Colange, M.: Reactive synthesis from LTL specification with spot. In: SYNT@CAV (2018)
24. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Falsification of LTL safety properties in hybrid systems. Int. J. Softw. Tools Technol. Transf. 15(4), 305–320 (2013)
25. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57 (1977)
26. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: POPL. pp. 179–190 (1989)
27. Sickert, S., Esparza, J.: An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In: LICS. pp. 831–844 (2020)
28. Sistla, A.P.: Safety, Liveness and Fairness in Temporal Logic. Formal Asp. Comput. 6(5), 495–512 (1994)
29. Sohail, S., Somenzi, F.: Safety first: a two-stage algorithm for the synthesis of reactive systems. Int. J. Softw. Tools Technol. Transf. 15(5-6), 433–454 (2013)
30. Somenzi, F.: Cudd: Cu decision diagram package 3.0.0. university of colorado at boulder
31. Tabajara, L.M., Vardi, M.Y.: Partitioning techniques in LTL_f synthesis. In: IJCAI. pp. 5599–5606 (2019)
32. Vardi, M.Y.: From verification to synthesis. In: VSTTE. Lecture Notes in Computer Science, vol. 5295, p. 2 (2008)
33. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A Symbolic Approach to Safety LTL Synthesis. In: HVC. pp. 147–162 (2017)
34. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTL_f Synthesis. In: IJCAI. pp. 1362–1369 (2017)
35. Zhu, S., Tabajara, L.M., Pu, G., Vardi, M.Y.: On the power of automata minimization in temporal synthesis. In: GandALF. pp. 117–134 (2021)