Compositional High-Quality Synthesis

Rafael Dewes and Rayna Dimitrova

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract Over the last years, there has been growing interest in synthesizing reactive systems from quantitative specifications, with the goal of constructing correct and high-quality systems. Considering quantitative requirements in systems consisting of multiple components is challenging not only because of scalability limitations but also due to the intricate interplay between the different possibilities of satisfying a specification and the required cooperation between components. Compositional synthesis holds the promise of addressing these challenges.
We study the compositional synthesis of reactive systems consisting of multiple components, from requirements specified in a fragment of the logic LTL[F], which extends LTL with quality operators. We consider specifications that are combinations of local and shared quantitative requirements. We present a sound decomposition rule that allows for synthesizing one component at a time. The decomposition requires assume-guarantee contracts between the components, and we provide a method for iteratively refining the assumptions and guarantees. We evaluate our approach with a prototype implementation, demonstrating its advantages over monolithic synthesis and ability to generate decompositions.

1 Introduction

Reactive synthesis can be used to automatically construct correct-by-design reactive systems from high-level specifications. The correctness requirements are typically specified using temporal logics such as Linear Temporal Logic (LTL). However, logics like LTL have limited ability to capture preferences on the quality of the synthesized system, such as resource requirements, efficiency, or prioritization of tasks. Over the last decades, the study of quantitative specification formalisms and the development of synthesis techniques for constructing high-quality implementations has attracted significant attention. Specification formalisms include weighted automata [8] and temporal logics equipped with propositional quality operators and discounting operators [2,1].

In the development of synthesis algorithms for quantitative specifications, the focus has been predominantly on single-component synthesis. Extensions have considered quantitative specifications in assume-guarantee form, for instance expressed in the GR(1)[$\mathcal{F}$] fragment [4], as well as relaxations of the synthesis problem, such as good-enough synthesis [3], where the system is required to produce an output satisfying the specification with a certain value only if the environment provides an input sequence for which such an output exists. While such

extensions focus on the explicit or implicit assumptions made about the environment, they do not study the interaction between cooperating components within the synthesized system in the presence of quantitative specifications. It has been widely recognized that compositional approaches are necessary in order to make reactive synthesis applicable to complex systems of multiple components.

In this paper we propose a *compositional method* for good-enough synthesis [3] for specifications expressed in a fragment of the logic LTL[$\mathcal{F}$] [2]. LTL[$\mathcal{F}$] extends LTL with quality operators, and instead of true or false, the satisfaction values of formulas are real values in $[0, 1]$. A higher value of satisfaction of a formula by an execution trace corresponds to a higher quality. Good-enough synthesis, introduced in [3], is a relaxation of the classical synthesis problem. For specifications in LTL[$\mathcal{F}$] this means that for each input sequence the synthesized system is required to ensure the highest value possible for this input sequence.

We study the problem of decomposing the good-enough synthesis task over multiple components. To this end, we consider LTL[$\mathcal{F}$] specifications that are a *combination of local specifications* for the individual components *and a shared specification*. The local specification φ^c_{local} for a component c captures requirements that are local to c, while the shared specification Ψ_{shared} captures requirements pertaining to multiple components. We furthermore make some natural assumptions about the specifications, which we make use of in order to provide a sound decomposition of the synthesis problem. We assume for each local specification that it does not refer to the output variables of other components, and that the shared specification is a safety property. These assumptions are meaningful in a setting where components have their own individual objectives, but must in addition jointly guarantee that some safety requirement is fulfilled.

The task we study is to synthesize in a compositional manner a system of *cooperating* components that ensures for each input sequence the *highest possible value for the combined specification*. Our goal is to decompose this task into synthesis tasks for the individual components, which for each component c consider only φ^c_{local} and Ψ_{shared}, and in addition, assumptions on the other components and additional guarantees that c provides to the other components when necessary. We illustrate the problem on an example.

Example. Suppose that our goal is to synthesize a system consisting of two components, where i is a Boolean variable input from the external environment, o_1 is a Boolean output of component 1, and o_2 is a Boolean output of component 2. We assume that each component has full information, that is, observes all environment inputs and outputs of other components. We consider implementations in the form of Moore machines, i.e., the output of each component at a given step can only depend on past inputs and outputs of other components.

The system must satisfy the specification $\Phi = \varphi^1_{local} \wedge \varphi^2_{local} \wedge \Psi_{shared}$, where $\varphi^1_{local} = (\square\diamondsuit o_1) \oplus_{\frac{1}{2}} (\square\diamondsuit(i \wedge \bigcirc \neg o_1))$ is the specification of component 1, and $\varphi^2_{local} = \square\diamondsuit o_2$ is the local specification of component 2, and $\Psi_{shared} = \square(o_1 \rightarrow \bigcirc \neg o_2)$ is the shared specification.

Here, the local specification φ^2_{local}, which requires that o_2 is true infinitely often, and the shared specification Ψ_{shared}, which requires that every time when

o_1 is true then o_2 must be false in the next step, are both qualitative LTL specifications. Hence, each has two possible values, 0 and 1.

The local specification φ^1_{local} of component 1 uses the weighted sum operator $\oplus_{\frac{1}{2}}$ with weight $\frac{1}{2}$ for each of the subformulas. Hence, an execution of the system where o_1 is true infinitely often will result in value of at least $\frac{1}{2}$ for φ^1_{local}. If additionally $i \wedge \bigcirc \neg o_1$ is true infinitely often on that execution then the value of φ^1_{local} will be 1. If both are false, the value of φ_1 is 0. Since Φ is the conjunction of the three specifications, its value is the minimum of their values.

The task is then to synthesize a system $\mathcal{S}$ that satisfies the following: for any value v and infinite sequence σ_I of values of i provided by the environment, if there is a sequence of outputs σ_O, such that the value of Φ on $\sigma_I \parallel \sigma_O$ is v, then, for the output $\mathcal{S}(\sigma_I)$ of $\mathcal{S}$ on σ_I, the value of Φ on $\sigma_I \parallel \mathcal{S}(\sigma_I)$ must be at least v.

Consider the monolithic system that has three (global) states s_1, s_2, s_3, and such that in s_1 the output is $(o_1 := \mathsf{true}, o_2 := \mathsf{true})$, and in the other two states it is $(o_1 := \mathsf{false}, o_2 := \mathsf{false})$. From s_1 the system always transitions to s_2 and from s_3 it always transitions to s_1. From s_2 it goes to s_1 if i is false and to s_3 if i is true. Thus, if i is infinitely often true, then this system implementation ensures value 1, since it also guarantees that each of o_1 and o_2 is true infinitely often and Ψ_{shared} holds. If i is *not* true infinitely often, then the maximal value of Φ that is possible is $\frac{1}{2}$, which is also what the implementation ensures.

Such a system can be synthesized by applying to Φ the algorithm presented in [3] for good-enough synthesis for LTL[$\mathcal{F}$] specifications.

Our goal is to decompose the synthesis problem, such that we *consider the local specifications of the two components in isolation.* That is, to synthesize an implementation for component 1 considering φ^1_{local} and Ψ_{shared}, and similarly for component 2. Each component cannot on its own enforce the maximal possible values for both the shared and its local specification. Thus, it needs to make assumptions on the behavior of the other component. Here, component 1 needs to make an assumption, A_1, on the behavior of component 2, which in this case is Ψ_{shared} itself. Component 2 also needs to make an assumption, A_2, that requires that o_1 is false infinitely often, to be able to satisfy both φ^2_{local} and Ψ_{shared}. Component 1 can guarantee the assumption A_2, while ensuring the maximal possible value for φ^1_{local} and Ψ_{shared} (under assumption A_1). The components obtained by projection from the system above satisfy these requirements.

An *assume-guarantee contract* between the components, like the one above, allows for the decomposition of the synthesis task for Φ into local synthesis tasks for the individual components. We study the problem of establishing such a decomposition and automatically deriving assume guarantee contracts.

Contributions We provide a decomposition rule that identifies conditions on the LTL[$\mathcal{F}$] specifications of the above form, which guarantee that an assume-guarantee decomposition is sound. In particular, we define the notion of *good-enough* tuples of assumptions, and describe a method for automatically deriving assume-guarantee contracts for quantitative specifications. We have implemented the proposed approach in a prototype and demonstrate on a set of examples that the compositional synthesis technique outperforms the monolithic one.

Related Work Compositional approaches for synthesis from qualitative specifications have been studied extensively. In assume-guarantee synthesis [11,9,21,18] the synthesis problem is decomposed using an assume-guarantee contract to capture the interface and dependencies between components. Other techniques are based on a decomposition of the given specification into independent specifications by analysis of the dependencies between components [17], or by semantic analysis of the language descried by the specification [16]. None of these approaches consider good-enough synthesis for quantitative specifications.

The notion of good-enough synthesis [3], where the system must only ensure the satisfaction value made possible by the environment, is closely related to the notions of dominant strategies [13], that is, strategies that perform as good as the best alternative, as well as admissible strategies [10], which are strategies that are not dominated by another strategy. Dominant strategies have been used for compositional synthesis [13,17], by making use of the fact that implementations must be dominant strategies and reducing the synthesis problem to a sequence of synthesis tasks treating the processes in the system one at a time. These compositional techniques are only studied in the qualitative setting and when the components have a common objective, while in our case we consider quantitative specifications and components have both shared and local specifications. In rational synthesis [20], the environment of the system being synthesized consists of rational agents with their own objectives. In contrast, we consider cooperating components as part of the system and aim to synthesize them compositionally.

The automatic generation of assumptions for qualitative specifications has been studied extensively. [12] presents a game-based method for deriving environment assumptions that turn an unrealizable specification into a realizable one. [5,6] propose techniques for correcting unrealizable specifications in the form of implications between assumptions and guarantees. In [21] the authors present an iterative procedure, called negotiation, for deriving assumption-guarantee pairs. Their assumption-generation method, similarly to ours, is based on [12]. Our iterative strengthening of assumptions follows the idea of their negotiation process. Neither of these works considers quantitative specifications.

A number of techniques have been developed for compositional synthesis for conjunctions of multiple specifications. In [15] this is done for solving games compositionally in the context of bounded synthesis, and [7] presents a technique for compositional synthesis for conjunctions of Safety LTL specifications. The method we use to treat the conjunctions of multiple combinations of values in our good-enough synthesis procedure is similar to these techniques, but in the context of the construction of the safety games in bounded synthesis.

2 Preliminaries

2.1 Languages and Automata Over Infinite Words

Let Σ be a finite alphabet. The set of finite (infinite) words over Σ is denoted by Σ^* (respectively Σ^ω). For a word $\sigma = \sigma_0, \sigma_1, \ldots \in \Sigma^\omega$, we denote with $\sigma[i] = \sigma_i$

the letter at position i, and with $\sigma[i,\infty) = \sigma_i, \sigma_{i+1}, \ldots$ the suffix of σ starting at position i. For a finite word $\sigma' \in \Sigma^*$ and a word $\sigma'' \in \Sigma^* \cup \Sigma^\omega$, we denote with $\sigma' \cdot \sigma''$ the concatenation of the prefix σ' and the suffix σ''. A language $L \subseteq \Sigma^\omega$ is a safety language if and only if for every $\sigma \in \Sigma^\omega \setminus L$ there exists a finite prefix σ' of σ such that for every $\sigma'' \in \Sigma^\omega$ it holds that $\sigma' \cdot \sigma'' \notin L$.

For a set X, we denote with 2^X the powerset of X. For a word σ over alphabet 2^X and a subset $Y \subseteq X$ we denote with $\mathsf{proj}(\sigma, Y)$ the projection of σ onto the alphabet 2^Y. Given disjoint sets $X_1, \ldots, X_m$ and for each $i \in \{1, \ldots, m\}$ a word σ_i over the alphabet 2^{X_i}, we define the parallel composition $\|_{i=1}^m \sigma_i$ of the words $\sigma_1, \ldots, \sigma_m$ such that $(\|_{i=1}^m \sigma_i)[j] = \bigcup_{i=1}^m \sigma_i[j]$ for all j.

We will use several types of automata over infinite words, whose definitions we recall in this subsection, together with some operations and properties.

A *generalized nondeterministic Büchi automaton* (GNBA) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \alpha)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $\alpha \subseteq 2^Q$ a set of sets of accepting states. A run of $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \alpha)$ on an infinite word $\sigma \in \Sigma^\omega$ is an infinite sequence $\rho \in Q^\omega$ of states such that $q_0 \in Q_0$, and for every $i \in \mathbb{N}$ it holds that $\rho[i+1] \in \delta(\rho[i], \sigma[i+1])$. A run ρ of a GNBA is accepting if and only if for every $F \in \alpha$ it holds that for every $i \in \mathbb{N}$ there exists $j \geq i$ such that $\rho[j] \in F$, i.e., ρ visits each set in α infinitely often. An infinite word σ is accepted by a GNBA $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on σ.

A *nondeterministic Büchi automaton* (NBA) is a GNBA $\mathcal{A} = (\Sigma, Q, \delta, Q_0, \alpha)$ with $|\alpha| = 1$. When $\mathcal{A}$ is an NBA we will also write $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ where $F \subseteq Q$ is the single set of accepting states. A *safety automaton* is a Büchi automaton in which all states are accepting. Hence, for safety automata every infinite run is accepting, and words that are not accepted have no infinite run. An automaton is *deterministic* if $|Q_0| = 1$ and $|\delta(q, a)| \leq 1$ for all $q \in Q$, $a \in \Sigma$.

A *universal co-Büchi automaton* (UCB) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$, where Σ, Q, δ and Q_0 are as in NBA, but now $F \subseteq Q$ is a set of *rejecting states*. A run ρ of a UCB is accepting if and only if there exists $i \in \mathbb{N}$ such that for every $j \geq i$ it holds that $\rho[j] \notin F$, i.e., ρ visits the set F only finitely many times. A UCB $\mathcal{A}$ accepts an infinite word σ if *all* infinite runs of $\mathcal{A}$ on σ are accepting.

For $\mathcal{A}$ over alphabet Σ, we define $\mathcal{L}(\mathcal{A}) := \{\sigma \in \Sigma^\omega \mid \sigma \text{ is accepted by } \mathcal{A}\}$.

For NBA $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ it holds for the UCB $\mathcal{U} := (\Sigma, Q, \delta, Q_0, F)$ that $\mathcal{L}(\mathcal{U}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$. Thus, a UCB for a language $L \subseteq \Sigma^\omega$ can be obtained from an NBA for the complement language. For GNBA $\mathcal{A}_i = (\Sigma, Q^i, \delta^i, Q_0^i, \alpha^i)$ for $i \in \{1, 2\}$, the product automaton is defined as $\mathcal{A}_1 \times \mathcal{A}_2 := (\Sigma, Q^1 \times Q^2, \delta_\times, Q_0^1 \times Q_0^2, \alpha^1 \cup \alpha^2)$ where $(q_1', q_2') \in \delta_\times((q_1, q_2), a)$ if and only if $q_i' \in \delta^i(q_i, a)$ for all $i \in \{1, 2\}$. It holds that $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_1)$. The union is defined as $\mathcal{A}_1 \cup \mathcal{A}_2 := (\Sigma, Q^1 \cup Q^2, \delta_\cup, Q_0^1 \cup Q_0^2, \{F_1 \cup Q^2 \mid F_1 \in \alpha^1\} \cup \{F_2 \cup Q^1 \mid F_2 \in \alpha^2\})$, where $q' \in \delta_\cup(q, a)$ if and only if $q' \in \delta^i(q, a)$ for some $i \in \{1, 2\}$. It holds that $\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

For a GNBA $\mathcal{A} = (2^{AP}, Q, \delta, Q_0, \alpha)$ and $AP' \subseteq AP$, we define the existential projection of $\mathcal{A}$ with respect to AP' to be the GNBA $\mathsf{proj}_\exists(\mathcal{A}, AP') \stackrel{\text{def}}{=} (2^{AP}, Q, \delta', Q_0, \alpha)$ with $\delta'(q, a) = \bigcup_{b \in 2^{AP'}} \delta(q, \mathsf{proj}(a, 2^{AP \setminus AP'}) \cup b)$. By definition,

the projection automaton has the language $\mathcal{L}(\mathsf{proj}_{\exists}(\mathcal{A}, AP')) = \{\sigma \in (2^{AP})^\omega \mid \exists \sigma' \in (2^{AP'})^\omega.\ \mathsf{proj}(\sigma, 2^{AP \setminus AP'}) \parallel \sigma' \in \mathcal{L}(\mathcal{A})\}$.

2.2 The Temporal Logic LTL[$\mathcal{F}$]

In this section, we recall the temporal logic $LTL[\mathcal{F}]$ introduced in [2]. Let AP be a set of Boolean atomic propositions, and $\mathcal{F} \subseteq \{f : [0,1]^k \to [0,1] \mid k \in \mathbb{N}\}$ a set of functions. The LTL[$\mathcal{F}$] formulas are generated by the grammar $\varphi ::= p \mid \mathsf{true} \mid \mathsf{false} \mid f(\varphi_1, \ldots, \varphi_k) \mid \bigcirc \varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$, where $p \in AP$, and $f \in \mathcal{F}$.

We consider sets $\mathcal{F}$ that include functions that allow us to express the usual Boolean operators, i.e., $\{f_\neg, f_\wedge, f_\vee\} \subseteq \mathcal{F}$, where $f_\neg(x) \stackrel{\text{def}}{=} 1 - x$, $f_\wedge(x,y) \stackrel{\text{def}}{=} \min\{x,y\}$ and $f_\vee(x,y) \stackrel{\text{def}}{=} \max\{x,y\}$. For ease of notation, we use the operators $\neg, \wedge, \vee$ instead of the corresponding functions. As noted in [2], LTL coincides with LTL[$\mathcal{F}$] when $\mathcal{F} = \{\neg, \wedge, \vee\}$. One useful function is weighted average $x \oplus_\lambda y \stackrel{\text{def}}{=} \lambda \cdot x + (1-\lambda) \cdot y$, where $\lambda \in \{0,1\}$. We define the temporal operators *finally* $\diamondsuit \varphi \stackrel{\text{def}}{=} \mathsf{true} \mathcal{U} \varphi$ and *globally* $\square \varphi \stackrel{\text{def}}{=} \neg(\diamondsuit \neg \varphi)$.

For an LTL[$\mathcal{F}$] formula φ, we denote with $Vars(\varphi)$ the set of atomic propositions occurring in φ, and with $|\varphi|$ the description size of φ.

The semantics of LTL[$\mathcal{F}$] is defined with respect to words in $(2^{AP})^\omega$, and maps an LTL[$\mathcal{F}$] formula φ and a word $\sigma \in (2^{AP})^\omega$, to a value $[\![\varphi, \sigma]\!] \in [0,1]$. For $f \in \mathcal{F}$, we define $[\![f(\varphi_1, \ldots, \varphi_k), \sigma]\!] := f([\![\varphi_1, \sigma]\!], \ldots, [\![\varphi_k, \sigma]\!])$. The semantics of *until* is $[\![\varphi_1 \,\mathcal{U}\, \varphi_2, \sigma]\!] := \max_{i \geq 0}\{\min\{[\![\varphi_2, \sigma[i,\infty)]\!], \min_{0 \leq j < i}[\![\varphi_1, \sigma[j,\infty)]\!]\}\}$. We refer the reader to [2] for the full formal definition of the semantics of LTL[$\mathcal{F}$]. We denote with $Vals(\varphi) \stackrel{\text{def}}{=} \{[\![\varphi, \sigma]\!] \mid \sigma \in (2^{AP})^\omega\}$ the set of possible values of an LTL[$\mathcal{F}$] formula φ. In [2] it was established that for every LTL[$\mathcal{F}$] formula φ it holds that $|Vals(\varphi)| \leq 2^{|\varphi|}$. That is, each formula's set of possible values is finite, and its cardinality is at most exponential in the size of φ.

Theorem 1 ([2]). *Let φ be an LTL[$\mathcal{F}$] formula over AP and $V \subseteq [0,1]$ be a set of values. There exists an GNBA $\mathcal{A}_{\varphi,V}$ such that for every $\sigma \in (2^{AP})^\omega$ it holds that $[\![\varphi, \sigma]\!] \in V$ if and only if $\sigma \in \mathcal{L}(\mathcal{A}_{\varphi,V})$. Furthermore, $\mathcal{A}_{\varphi,V}$ has at most $2^{(|\varphi|^2)}$ states and at most $|\varphi|$ sets of accepting states.*

One relevant property of the construction in the proof of Theorem 1 for our automata-based compositional synthesis procedure is that the set of values V only plays a role in the construction of the set of initial states. In particular, one can construct the automaton $\mathcal{A}_{\varphi, Vals(\varphi)} = (2^{AP}, Q, \delta, Q, \alpha)$ where every state is initial. From $\mathcal{A}_{\varphi, Vals(\varphi)}$ we can obtain the respective automaton $\mathcal{A}_{\varphi,V}$ for every V by instantiating the corresponding set of initial states based on V. Intuitively, these are the states in Q where the formula φ has some value $v \in V$.

3 Good-Enough Assume-Guarantee Decomposition

We begin this section by formally introducing the problem we study in the paper, namely, the synthesis of multi-component reactive systems from LTL[$\mathcal{F}$] specifications. We first describe the system model we consider.

3.1 Multi-Component Reactive Systems

We consider reactive systems with a finite set I of *input atomic propositions* and a finite set O of *output atomic propositions* that are disjoint, i.e., $I \cap O = \emptyset$. We call the words in $(2^{AP})^\omega$ *(execution) traces.*

A *reactive component* is a Moore machine $M = (I_M, O_M, S, s^{init}, \rho, Out)$, where I_M and O_M are M's sets of input and output propositions respectively, S is a finite set of states, $s^{init} \in S$ is the initial state, $\rho : S \times 2^{I_M} \to S$ is the transition function, and $Out : S \to 2^{O_M}$ is the output labeling function. Given an input trace $\sigma_{I_M} \in (2^{I_M})^\omega$, M produces the output trace $M(\sigma_{I_M}) \in (2^{O_M})^\omega$ such that $M(\sigma_{I_M})[i] = Out(s_i)$, where the infinite sequence $s_0, s_1, \ldots$ of states is such that $s_0 = s^{init}$, and $s_{i+1} = \rho(s_i, \sigma_{I_M}[i])$ for every $i \in \mathbb{N}$.

A *multi-component reactive system* is a tuple $\mathcal{S} = (I, O, \mathcal{M})$, where I and O are the sets of input and output propositions respectively, $\mathcal{M} = \langle M_1, \ldots M_n \rangle$ is a tuple of reactive components $M_c = (I_c, O_c, S_c, s_c^{init}, \rho_c, Out_c)$ such that (1) for every $c, c' \in \{1, \ldots, n\}$ with $c \neq c'$ it holds that $O_c \cap O_{c'} = \emptyset$, (2) $\biguplus_{c=1}^{n} O_c = O$, and (3) $I_c = I \cup (O \setminus O_c)$. Conditions (1) and (2) stipulate that the sets of output propositions of the individual components partition O. Condition (3) stipulates that each component can read all input propositions I and all output propositions of the other components. We denote with $O_{\overline{c}} = \bigcup_{c' \in \{1,\ldots,n\} \setminus \{c\}} O_{c'}$ the set of outputs of components different from c. Given an input trace $\sigma_I \in (2^I)^\omega$, a multi-component reactive system generates an output trace $\sigma_O \in (2^O)^\omega$, which we denote by $(\|_{c=1}^{n} M_c)(\sigma_I)$, such that $\mathsf{proj}(\sigma_O, O_c) = M_c(\sigma_I \parallel \mathsf{proj}(\sigma_O, O_{\overline{c}}))$ for all c. That is, $(\|_{c=1}^{n} M_c)(\sigma_I)$ is the composition of all the output traces of the components in $\mathcal{S}$.

3.2 Good-Enough Realizability and Synthesis from LTL[$\mathcal{F}$]

We study the problem of synthesizing multi-component reactive systems from LTL[$\mathcal{F}$] specifications, precisely formulated below.

Problem 1: Given an LTL[$\mathcal{F}$] formula Φ over atomic propositions $AP = I \uplus O$, and a partitioning $O_1, \ldots, O_n$ of the set of output propositions, decide whether there exists a multi-component reactive system $\mathcal{S}$ such that $\mathcal{S} = (I, O, \mathcal{M})$ where O_c is the set of output propositions of component $c \in \{1, \ldots, n\}$, such that for every $\sigma_I \in (2^I)^\omega$ and every $v \in \mathit{Vals}(\Phi)$ the following condition (1) holds.

$$\begin{aligned} &\text{If there exists } \sigma_O \in (2^O)^\omega \text{ with } [\![\Phi, \sigma_I \parallel \sigma_O]\!] = v, \\ &\qquad\qquad \text{then } [\![\Phi, \sigma_I \parallel (\|_{c=1}^{n} M_c)(\sigma_I)]\!] \geq v. \end{aligned} \tag{1}$$

Since our definition of multi-component reactive systems allows for components to observe all inputs and outputs of other components, the above problem can be solved by considering the monolithic synthesis problem for the composed system. This problem is known to be 2EXPTIME-complete [3]. When a monolithic system is synthesized, it can be easily decomposed using projection. However, this creates a dependency between the implementations of the individual

components. Furthermore, for specifications that combine both local requirements on the components and shared system properties, a synthesis approach that treats such specifications compositionally is desirable.

For the remaining sections, we consider the problem of synthesizing n components, which we label with indices $c \in \{1, \dots, n\}$, and fix a corresponding partitioning $O_1, \dots, O_n$ of the set of output propositions.

3.3 Good-Enough Decomposition

As stated in Section 1, we consider specifications Φ expressed as a combination of local specifications for the individual components, as well as a shared requirement. Formally, we assume that $\Phi = \mathit{comb}(\varphi^1_{local}, \dots, \varphi^n_{local}, \Psi_{shared})$, where the function $\mathit{comb} : [0,1]^{n+1} \to [0,1]$ is non-decreasing in each subset of its arguments, that is, for every $v_1, \dots, v_{n+1}$ and $v'_1, \dots, v'_{n+1}$, if we have $\mathit{comb}(v'_1, \dots, v'_{n+1}) < \mathit{comb}(v_1, \dots, v_{n+1})$, then, $v'_i < v_i$ for some i.

We refer to Ψ_{shared} as the *shared specification* and to each φ^c_{local} as the *local specification of component c*. Below is an example of such a specification.

Example 1. Let $I = \{i\}, O_1 = \{o_1\}, O_2 = \{o_2\}$ and consider the specification $\Phi = (\varphi^1_{local} \oplus_{\frac{1}{3}} (\varphi^2_{local} \oplus_{\frac{1}{2}} \Psi_{shared}))$, where the function *comb* is a weighted sum in which each part is weighted $\frac{1}{3}$ and the individual specifications are

$$\varphi^1_{local} = \square o_1 \oplus_{\frac{1}{2}} \square\diamondsuit o_1, \qquad \varphi^2_{local} = \square o_2 \oplus_{\frac{1}{2}} \square\diamondsuit o_2,$$
$$\Psi_{shared} = \square(i \to (\bigcirc \neg o_1 \oplus_{\frac{1}{2}} \bigcirc \neg o_2)).$$

If for some $\sigma \in (2^{AP})^\omega$ we have $[\![\varphi^1_{local}, \sigma]\!] = 1$, $[\![\varphi^2_{local}, \sigma]\!] = \frac{1}{2}$ and $[\![\Psi_{shared}, \sigma]\!] = 0$, then we have the value $[\![\Phi, \sigma]\!] = \frac{1}{2}$ for the overall specification.

As mentioned in Section 1, we furthermore assume that the specification Φ satisfies two additional conditions, under which we establish the soundness of our compositional synthesis approach. The first condition restricts the shared specifications, while the second condition restricts the local specifications.

Condition 1. The shared specifications Ψ_{shared} is a *safety LTL[$\mathcal{F}$] specification.*

Definition 1 (Safety LTL[$\mathcal{F}$] specifications). *We say that an LTL[$\mathcal{F}$] formula is a* safety specification *if and only if for every word $\sigma \in (2^{AP})^\omega$ and every value $v \in \mathit{Vals}(\varphi)$, if $[\![\varphi, \sigma]\!] < v$, then there exists a prefix σ' of σ, such that for every possible infinite continuation $\sigma'' \in (2^{AP})^\omega$ of σ' we have $[\![\varphi, \sigma' \cdot \sigma'']\!] < v$.*

If we consider the LTL fragment of LTL[$\mathcal{F}$], then the above notion coincides with the notion of LTL-definable safety languages. To see this, note that for an LTL formula φ we have $\mathit{Vals}(\varphi) = \{0, 1\}$. Thus, $[\![\varphi, \sigma]\!] < 1$ corresponds to φ being violated by σ, and the condition corresponds to the existence of a bad prefix. In Example 1 above, the shared specification is a safety specification.

Condition 2. For each component c, the local specification φ^c_{local} refers only to output propositions in O_c and input propositions in I, i.e., $\mathit{Vars}(\varphi^c_{local}) \cap O \subseteq O_c$.

Note that Ψ_{shared} can refer to all the input and output signals.

Under the above assumptions, we consider the synthesis problem for LTL[$\mathcal{F}$] specifications of the above form in a compositional manner. We call a specification that satisfies all of the conditions *compositional*.

Our compositional synthesis approach is based on assume-guarantee contracts, which formalize the interface properties between the components.

Definition 2 (Assume-guarantee contract). *An* assume-guarantee contract *is a tuple $\langle (A_c, G_c)\rangle_{c=1}^{n}$ where $A_c \subseteq (2^{AP})^\omega$ is called the* assumption of component c, *and $G_c \subseteq (2^{AP})^\omega$ is called the* guarantee of component c, *where each A_c and G_c are safety languages, $\bigcap_{c' \in \{1,\dots,n\}\setminus\{c\}} G_{c'} \subseteq A_c$, and*

- *Let $\sigma \in (2^{AP})^*$ be a finite word such that there exists an infinite word $\sigma' \in (2^{AP})^\omega$ such that $\sigma \cdot \sigma' \in A_c$. Then, for every $o_c \in 2^{O_c}$, there exists $\sigma'' \in (2^{AP})^\omega$ such that $\sigma \cdot \sigma'' \in A_c$ and $\mathsf{proj}(\sigma''[0], O_c) = o_c$.*
- *Let $\sigma \in (2^{AP})^*$ be a finite word such that there exists an infinite word $\sigma' \in (2^{AP})^\omega$ such that $\sigma \cdot \sigma' \in G_c$. Then, for every $o_{\overline{c}} \in 2^{O_{\overline{c}}}$, there exists $\sigma'' \in (2^{AP})^\omega$ such that $\sigma \cdot \sigma'' \in G_c$ and $\mathsf{proj}(\sigma''[0], O_{\overline{c}}) = o_{\overline{c}}$.*

The first condition ensures that component c cannot on its own violate its assumption A_c by selecting a bad output o_c. The second condition states that the remaining components cannot violate the guarantee which c must provide, by selecting some bad output $o_{\overline{c}}$. We will employ assume-guarantee contracts to decompose the synthesis problem for Φ into local synthesis problems for the individual components. To guarantee soundness, we impose a condition on the assumptions, which we call *good-enough assumptions*. Intuitively, good-enough assumptions do not "eliminate" possible values of Ψ_{shared}.

We are now ready to state our compositional synthesis problem.

Definition 3 (Good-enough assumptions). *Let $\langle A_1, \dots A_n\rangle$ be assumptions for the components in $\{1, \dots, n\}$. We say that $\langle A_1, \dots A_n\rangle$ is a* good-enough *tuple of assumptions if and only if for all $\sigma_I \in (2^I)^\omega$, for all $v \in Vals(\Psi_{shared})$:*

$$\begin{aligned} &\text{if there exists } \sigma_O \in (2^O)^\omega \text{ with } [\![\Psi_{shared}, \sigma_I \parallel \sigma_O]\!] = v, \\ &\text{then there exists } \sigma'_O \in (2^O)^\omega \text{ with } [\![\Psi_{shared}, \sigma_I \parallel \sigma'_O]\!] \geq v \\ &\qquad\qquad \text{and } (\sigma_I \parallel \sigma'_O) \in \bigcap_{c \in \{1,\dots,n\}} A_c. \end{aligned} \tag{2}$$

Problem 2: Given a compositional LTL[$\mathcal{F}$] specification Φ over atomic propositions $AP = I \uplus O$ with partitioning $O_1, \dots, O_n$ of the set of output propositions, decide whether there exists a multi-component reactive system $\mathcal{S} = (I, O, \mathcal{M})$ where O_c is the set of output propositions of component $c \in \{1, \dots, n\}$, such that for each component $c \in \{1, \dots, n\}$ the conditions below are satisfied

$$\begin{aligned} &\forall \sigma_I \in (2^I)^\omega, \forall \sigma_{O_{\overline{c}}} \in (2^{O_{\overline{c}}})^\omega, \forall u \in Vals(\varphi^c_{local}), \forall w \in Vals(\Psi_{shared}) : \\ &\text{if there exists } \sigma_{O_c} \in (2^{O_c})^\omega, \text{ with } [\![\varphi^c_{local}, \sigma_I \parallel \sigma_{O_c} \parallel \sigma_{O_{\overline{c}}}]\!] = u, \\ &\qquad\qquad \text{and } [\![\Psi_{shared}, \sigma_I \parallel \sigma_{O_c} \parallel \sigma_{O_{\overline{c}}}]\!] = w \\ &\text{and if } (\sigma_I \parallel M_c(\sigma_I \parallel \sigma_{O_{\overline{c}}}) \parallel \sigma_{O_{\overline{c}}}) \in A_c, \\ &\text{then } [\![\varphi^c_{local}, \sigma_I \parallel M_c(\sigma_I \parallel \sigma_{O_{\overline{c}}}) \parallel \sigma_{O_{\overline{c}}}]\!] \geq u \text{ and} \\ &\qquad [\![\Psi_{shared}, \sigma_I \parallel M_c(\sigma_I \parallel \sigma_{O_{\overline{c}}}) \parallel \sigma_{O_{\overline{c}}}]\!] \geq w \end{aligned} \tag{3}$$

$$\forall \sigma_I \in (2^I)^\omega, \forall \sigma_{O_{\overline{c}}} \in (2^{O_{\overline{c}}})^\omega : (\sigma_I \parallel M_c(\sigma_I \parallel \sigma_{O_{\overline{c}}}) \parallel \sigma_{O_{\overline{c}}}) \in \overline{A}_c \cup G_c, \quad (4)$$

where $\langle (A_1, G_1), \dots, (A_n, G_n) \rangle$ is an assume-guarantee contract for components $\{1, \dots, n\}$ such that $\langle A_1, \dots A_n \rangle$ is a good-enough tuple of assumptions.

Intuitively, in *Problem 2* we consider only φ^c_{local} and Ψ_{shared} for each component c, without the remaining context of Φ, that is, the function *comb* and the local specifications of the other components. To ensure soundness of the decomposition, we consider *all possible* pairs (u, w) of values of φ^c_{local} an Ψ_{shared} and require that *for each pair for values that is possible*, the component's strategy ensures at least these values. In that way, an implementation for a component c that satisfies condition (3) does not restrict the possible values of Ψ_{shared} unnecessarily. We will see below that this results in the decomposition rule being sound but incomplete. But first, we show a simple example that demonstrates why we imposed the two conditions on the individual specifications in Φ.

Example 2. Let $I = \{i\}, O_1 = \{o_1\}, O_2 = \{o_2\}$ and $\Phi = \mathsf{true} \wedge \mathsf{true} \wedge \Psi_{shared}$, where $\Psi_{shared} = ((\square i) \leftrightarrow (\diamondsuit o_1)) \wedge ((\square i) \leftrightarrow (\diamondsuit o_2))$. Here the shared specification Ψ_{shared} is not a safety property. While for any $\sigma_I \in (2^I)^\omega$ there exist sequences of outputs that satisfy Φ, there is no system that satisfies Φ for every σ_I, as it would have to make a correct guess about the future inputs.

Consider an implementation of component $c \in \{1, 2\}$ that waits until the other component outputs true, and then does so itself, and otherwise outputs false. Such a pair of implementations satisfies condition (3), as it only requires that Φ is satisfied if the environment *and the other component* made it possible. However, the composition of these two implementations never sets any of o_1 and o_2 to true, and thus does not satisfy the conditions in *Problem 1*.

If we allow local specifications to refer to outputs of other components we can transform the above example into one with local specifications.

The next example shows a specification where the local synthesis problems in *Problem 2* are realizable without any extra assumptions.

Example 3. Let $I = \{i\}, O_1 = \{o_1\}, O_2 = \{o_2\}$ and $\Phi = \varphi^1_{local} \wedge \varphi^2_{local} \wedge \Psi_{shared}$, where $\varphi^c_{local} = \square o_c \oplus_{\frac{1}{2}} \square\diamondsuit o_c$ for each c and $\Psi_{shared} = \square(\neg i \rightarrow \bigcirc(\neg o_1 \wedge \neg o_2))$.

A system in which each component c sets o_c to true whenever i was true in the step before satisfies condition (1) for each input sequence and value v of Φ.

Taking $A_1 = G_1 = A_2 = G_2 = (2^{AP})^\omega$ we have that the same system satisfies conditions (3) and (4) for each component. This is because condition (3) only requires component c to ensure value 1 for Ψ_{shared} shared when i is false and $o_{\overline{c}}$ is false, and hence no explicit assumption is needed. This is in general not the case, and in many cases, cooperation and assumptions are necessary.

The next theorem establishes the soundness of the decomposition.

Theorem 2 (Soundness of GE A/G decomposition). *Let Φ be a compositional LTL[$\mathcal{F}$] specification with a partitioning $\{O_1, \dots, O_n\}$ of the output propositions. Let $\langle (A_1, G_1), \dots, (A_n, G_n) \rangle$ be an assume-guarantee contract for*

components $\{1, \dots, n\}$ *such that* $\langle A_1, \dots A_n \rangle$ *is a good-enough tuple of assumptions. Then, every multi-component reactive system* $\mathcal{S} = (I, O, \mathcal{M})$ *that is a solution to* Problem 2 *is also a solution to* Problem 1*.*

The converse to the statement of Theorem 2 is not true, as the following example demonstrates. That is, the decomposition rule is sound, but not complete.

Example 4 (Incompleteness of the Decomposition).
Let $I = \{i\}$, $O_1 = \{o_1\}$, $O_2 = \{o_2\}$ and $\Phi = \varphi^1_{local} \wedge \varphi^2_{local} \wedge \Psi_{shared}$, where

$$\varphi^1_{local} = \square((i \rightarrow \bigcirc o_1) \oplus_{\frac{1}{2}} \diamondsuit o_1), \quad \varphi^2_{local} = \square((i \rightarrow \bigcirc o_2) \oplus_{\frac{2}{3}} \diamondsuit o_2)$$
$$\Psi_{shared} = \square(i \oplus_{\frac{1}{2}} \bigcirc \neg(o_1 \wedge o_2))$$

For the input trace $\sigma_I = \{i\}^\omega$ there exists σ_O such that $[\![\Phi, \sigma_I \parallel \sigma_O]\!] = \frac{1}{2}$, and this is the best value achievable by the system for this input sequence. The system that outputs $o_1 = o_2 = \mathsf{true}$ if i held in the previous step, and alternates between o_1 and o_2 otherwise, achieves this value and satisfies the conditions of *Problem 1*. However, there exists no implementation that satisfies the conditions of the decomposition in *Problem 2*. To see this, note that for each c we have that for $\sigma_I = \{i\}^\omega$ and $\sigma_{O_{\overline{c}}} = \{o_{\overline{c}}\}^\omega$ there exists σ_{O_c} such that $[\![\varphi^c_{local}, \sigma_I \parallel \sigma_{O_c} \parallel \sigma_{O_{\overline{c}}}]\!] = 1$ and there exists σ_{O_c} such that $[\![\Psi_{shared}, \sigma_I \parallel \sigma_{O_c} \parallel \sigma_{O_{\overline{c}}}]\!] = 1$, but there is no implementation for c that ensures both values, and hence there is no implementation that satisfies (3).

4 Compositional Good-Enough Synthesis

We propose an approach to solving *Problem 2* using bounded synthesis, which we describe in this section. In a first step, our synthesis procedure constructs several automata from the given specifications and assume-guarantee contract (if given). In order to facilitate the generation of assumptions, we present a compositional synthesis method based on bounded synthesis [19], where for a given bound on the size of the implementations, a safety game is constructed from the automata constructed in the first step. If the current set of assumptions is insufficient, that is, the local synthesis problems for some component c has no solution, we present a method for strengthening the assumption A_c of component c.

4.1 Automata Constructions

We begin by detailing the different automata constructed by our procedure.

Automata for the specifications. Consider φ^c_{local} and Ψ_{shared}. Using the construction from Theorem 1, we construct the automata

- $\mathcal{B}_c = \mathcal{A}_{\varphi^c_{local}, Vals(\varphi^c_{local})} = (2^{AP}, Q^c, \delta^c, Q^c, \alpha^c)$, the GNBA for φ^c_{local} and
- $\mathcal{B}_s = \mathcal{A}_{\Psi_{shared}, Vals(\Psi_{shared})} = (2^{AP}, Q^s, \delta^s, Q^s, \alpha^s)$, the GNBA for Ψ_{shared}.

Both $\mathcal{B}_c$ and $\mathcal{B}_s$ are constructed for the respective full set of formula values.

We then construct the GNBA $\mathcal{B}'_{cs} = \mathsf{proj}_\exists(\mathcal{B}_c \times \mathcal{B}_s, O_c)$ and $\mathcal{B}''_{cs} = \mathcal{B}_c \cup \mathcal{B}_s$, which accept the existential projection of the product on O_c and the union.

Assumptions as automata. We consider assume-guarantee contracts represented as automata. The assumptions A_c and the complement languages $2^{AP} \setminus G_c$ of the guarantees of all components are represented respectively as the NBA

- $\mathcal{A}_c = (2^{AP}, Q^{c,a}, \delta^{c,a}, Q_0^{c,a}, \alpha^{c,a})$ is the assumption of component c.
- $\overline{\mathcal{G}_c} = (2^{AP}, Q^{c,g}, \delta^{c,g}, Q_0^{c,g}, \alpha^{c,g})$ is the complement of the guarantee of c.

Combining specifications and contract. From the automata $\mathcal{B}_c, \mathcal{B}_s, \mathcal{A}_c$ and $\overline{\mathcal{G}_c}$, we construct the GNBA $\mathcal{B} = (2^{AP}, \widehat{Q}, \widehat{\delta}, \widehat{Q}_0, \widehat{\alpha}) := (\mathcal{B}'_{cs} \times \mathcal{A}_c \times \mathcal{B}''_{cs}) \cup (\mathcal{A}_c \times \overline{\mathcal{G}_c})$.

We will use the GNBA $\mathcal{B}$ to characterize the language of the traces that violate at least one of conditions (3) and (4).

In the construction, we ensure that the states of $\mathcal{B}$ are of one of the forms (1) $(q_\exists^c, q_\exists^s, q^{ca}, q^{cs})$, where $q_\exists^c \in Q^c$, $q_\exists^s \in Q^s$, $q^{ca} \in Q^{c,a}$, $q^{cs} \in (Q^c \cup Q^s)$, or (2) $q^{cg} \in Q^{c,a} \times Q^{c,g}$. The set of initial states in $\mathcal{B}$ is $\widehat{Q}_0 = Q^c \times Q^s \times Q_0^{c,a} \times (Q^c \cup Q^s) \cup (Q_0^{c,a} \times Q_0^{c,g})$. With that, the states of the first form assign values to the formulas φ_{local}^c and Ψ_{shared} in the respective sub-states.

Let $u \in \mathit{Vals}(\varphi_{local}^c)$ and $w \in \mathit{Vals}(\Psi_{shared})$. The above property of $\mathcal{B}$ allows us to devise from $\mathcal{B}$ an automaton $\mathcal{B}_{(u,w)}$ as follows. First, for $\sim \in \{<, \geq, =\}$, let

$$Q_{\sim u}^c := \{q \in Q^c \mid q(\varphi_{local}^c) \sim u\} \text{ and } Q_{\sim w}^s := \{q \in Q^s \mid q(\Psi_{shared}) \sim w\}.$$

We define the set of initial states in $\mathcal{B}$ for the pair of values (u, w) as the set

$$\widehat{Q}_0^{(u,w)} := \big(Q_{\geq u}^c \times Q_{\geq w}^s \times Q_0^{c,a} \times (Q_{<u}^c \cup Q_{<w}^s)\big) \cup (Q_0^{c,a} \times Q_0^{c,g}).$$

With that, we define the automaton $\mathcal{B}_{(u,w)} := (2^{AP}, \widehat{Q}, \widehat{\delta}, \widehat{Q}_0^{(u,w)}, \widehat{\alpha})$.

Intuitively, the language of the automaton $\mathcal{B}_{(u,w)}$ consists of the traces that violate at least one of conditions (3) and (4) for the value pair (u, w).

Proposition 1. *For the GNBA $\mathcal{B}_{(u,w)}$ constructed above it holds that for $\sigma \in (2^{AP})^\omega$ we have $\sigma \in \mathcal{L}(\mathcal{B}_{(u,w)})$ iff $\sigma \notin (\overline{A}_c \cup G_c)$ or all of the following hold:*

- *there exists $\sigma_{O_c} \in (2^O)^\omega$ such that $[\![\varphi_{local}^c, \mathsf{proj}(\sigma, I \cup O_{\bar{c}}) \parallel \sigma_{O_c}]\!] \geq u$ and $[\![\Psi_{shared}, \mathsf{proj}(\sigma, I \cup O_{\bar{c}}) \parallel \sigma_{O_c}]\!] \geq w$, and $\sigma \in A_c$,*
- *$[\![\varphi_{local}^c, \sigma]\!] < u$ or $[\![\Psi_{shared}, \sigma]\!] < w$.*

Transformation to UCB. From the GNBA $\mathcal{B}$ constructed above, we obtain an NBA, which we then interpret as a UCB for the complement language.

For φ_{local}^c, Ψ_{shared}, $\mathcal{A}_c$ and $\overline{\mathcal{G}_c}$, we denote with $\mathsf{UCB}_c(\mathcal{B}_c, \mathcal{B}_s, \mathcal{A}_c, \overline{\mathcal{G}_c})$ the universal co-Büci automaton obtained in this way from the GNBA $\mathcal{B}_c$ and $\mathcal{B}_s$.

Given $\mathcal{U} := \mathsf{UCB}_c(\mathcal{B}_c, \mathcal{B}_s, \mathcal{A}_c, \overline{\mathcal{G}_c})$, $u \in \mathit{Vals}(\varphi_{local}^c)$ and $w \in \mathit{Vals}(\Psi_{shared})$, we denote with $\mathsf{Instantiate}_c(\mathcal{U}, u, w)$ the UCB obtained from the GNBA $\mathcal{B}_{(u,w)}$.

Proposition 1 provides us with a basis for a solution to *Problem 2* when we are given an assume-guarantee contract. For a given component c, using the product of the automata $\mathsf{Instantiate}_c(\mathcal{U}_c, u, w)$, for all $u \in \mathit{Vals}(\varphi_{local}^c)$ and $w \in \mathit{Vals}(\Psi_{shared})$, we can apply any suitable reactive synthesis method.

However, in order to facilitate the generation of assumptions, we propose a procedure based on bounded synthesis. In the next two subsections we first describe our compositional synthesis procedure with given assumptions, and then present the iterative generation of contracts.

function CompSynt $(\langle\varphi^c_{local}\rangle^n_{c=1}, \Psi_{shared}, \mathcal{B}_s, \langle\mathcal{B}_c\rangle^n_{c=1}, \langle\mathcal{A}_c, \overline{\mathcal{G}}_c\rangle^n_{c=1}, b_{init}, b_{max})$

```
1  if ¬GoodEnough(⟨A_c⟩_{c=1}^n, B_s) then return ⊥
2  b := b_init
3  while true do
4      done := true
5      for c = 1, ..., n do
6          U_c := UCB_c(B_c, B_s, A_c, G̅_c)
7          M_c := LocalSynt(U_c, I, O_c, O_c̄, Vals(φ^c_local), Vals(Ψ_shared), b)
8          if M_c = ⊥ then done := false; break
9      if done then return ⟨M_c⟩_{c=1}^n
10     if b < b_max then b := increment(b) else return unknown
```

Algorithm 1: Compositional bounded synthesis for a combined specification $\Phi = comb(\varphi^1_{local}, \ldots, \varphi^n_{local}, \Psi_{shared})$, with a given assume-guarantee contract consisting of assumptions $\langle\mathcal{A}_c\rangle^n_{c=1}$ and negated guarantees $\langle\overline{\mathcal{G}}_c\rangle^n_{c=1}$. The automata $\mathcal{B}_s = \mathcal{A}_{\Psi_{shared}, Vals(\Psi_{shared})}$ and $\mathcal{B}_c := \mathcal{A}_{\varphi^c_{local}, Vals(\varphi^c_{local})}$ are given as input.

function LocalSynt $(\mathcal{U}_c, I, O_c, O_{\overline{c}}, U, W, b)$

```
1  let D be a deterministic safety automaton for the language (2^AP)^ω
2  for (u, w) ∈ U × W do
3      U_(u,w) := Instantiate_c(U_c, u, w); D := Safety(U_(u,w), b, D)
4      (Win, strategy) := SolveSafetyGame(D, I ⊎ O_c̄, O_c)
5      if Win := ⊥ then return ⊥
6      D := PruneLosing(D, Win, I ⊎ O_c̄, O_c)
7  return ToMoore(strategy, I, O_c, O_c̄)
```

Algorithm 2: Bounded synthesis for a single component c, with specification given by the UCB $\mathcal{U}_c$ and sets of values U and W defining the initial states. I is the set of inputs, O_c is the set of outputs c, $O_{\overline{c}}$ are the outputs of the remaining components. b is the bound for the bounded synthesis algorithm.

4.2 Synthesis with a Given Assume-Guarantee Contract

For a given component c, our method, based on bounded synthesis, processes the automata for the different value pairs incrementally in the construction of the safety game for a given bound. The compositional synthesis procedure CompSynt described below is detailed in Algorithm 1, and the incremental bounded synthesis method LocalSynt for a single component in Algorithm 2.

The procedure CompSynt first verifies that the assumptions meet the good-enough condition by calling the function GoodEnough (described later). If this is the case, CompSynt iterates over the components, constructing the UCB $\mathcal{U}_c := \mathsf{UCB}_c(\mathcal{B}_c, \mathcal{B}_s, \mathcal{A}_c, \overline{\mathcal{G}}_c)$ and invoking LocalSynt (described later), which performs incremental bounded synthesis for a component c. If an implementation is found by LocalSynt for all c, then CompSynt returns a multi-component reactive system. Otherwise, the bound is increased if the maximum (given as parameter) is not reached. If the latter is the case, CompSynt returns unknown.

Checking for good-enough assumptions. The function GoodEnough verifies that a tuple of assumptions represented as automata $\langle \mathcal{A}_c \rangle_{c=1}^n$ is good-enough. It uses the GNBA $\mathcal{B}_s$ representing the shared specification Ψ_{shared}. For every $v \in \mathit{Vals}(\Psi_{shared})$ the procedure constructs the GNBA

- $\mathcal{D}_s^{=v}$ obtained from $\mathsf{proj}_\exists(\mathcal{B}_s, O)$ by setting the set of initial states to $Q^s_{=v}$,
- $\mathcal{D}_s^{\geq v}$ obtained from $\mathsf{proj}_\exists(\mathcal{B}_s \cap \bigcap_{c\in\{1,\dots,n\}} \mathcal{A}_c, O)$ with init. states $Q^s_{\geq v} \times Q_0^{c,a}$.

Then, we verify that $\langle \mathcal{A}_c \rangle_{c=1}^n$ is good-enough by checking if the language inclusion $\mathcal{L}(\mathcal{D}_s^{=v}) \subseteq \mathcal{L}(\mathcal{D}_s^{\geq v})$, which directly corresponds to condition (2), holds.

Incremental bounded synthesis. The procedure LocalSynt iterates over the pairs of values in the set $U \times W$, where $U := \mathit{Vals}(\varphi^c_{local})$, $W := \mathit{Vals}(\Psi_{shared})$. For each (u, w), it constructs the UCB $\mathsf{Instantiate}_c(\mathcal{U}_c, u, w)$ instantiated from $\mathcal{U}_c$ for this value pair. It then constructs and solves incrementally a safety game.

Function Safety constructs a deterministic safety automaton from the UCB $\mathcal{U}_{(u,w)}$ for bound $b \in \mathbb{N}$. Different from the single automaton case, in LocalSynt we are constructing a deterministic safety automaton for the product of all $\mathsf{Instantiate}_c(\mathcal{U}_c, u, w)$ for $(u, w) \in U \times W$. We perform the construction incrementally. LocalSynt maintains a deterministic safety automaton $\mathcal{D}$ which is the product constructed thus far. $\mathcal{D}$ is passed as an argument to Safety and used in an on-the-fly construction to prune losing choices from the deterministic safety automaton constructed from the current $\mathcal{U}_{(u,w)}$.

Function SolveSafetyGame applied to $\mathcal{D}$ performs the standard construction of transforming a deterministic automaton into a two-player game by splitting the input propositions $I \uplus O_{\overline{c}}$ (note that here the output of other components is treated as input) and the output propositions O_c. The safety game is solved and SolveSafetyGame returns a pair $(\mathit{Win}, \mathit{strategy})$, where Win is either $\bot$, in the case when the initial state of $\mathcal{D}$ is not winning for the output player, or otherwise Win is the set of states winning for the output player, and *strategy* is the most permissive winning strategy for the output player. Function PruneLosing takes as input the automaton $\mathcal{D}$ and the winning region $\mathit{Win} \neq \bot$ computed by SolveSafetyGame and prunes from $\mathcal{D}$ the choices of the output player that do not lead to states in Win. Since the implementation must satisfy (3) for all (u, w), all the calls to SolveSafetyGame must be successful to return an implementation.

4.3 Synthesis with Iterative Assumption Generation

Assume-guarantee contracts can be difficult to design, especially for the synthesis of good-enough implementations of quantitative specifications. We present a method for iterative generation of good-enough assumptions, inspired by the notion of negotiation introduced in [21]. The idea is to consider the components in turn, and, if the local synthesis problem is not realizable for a component, to generate assumptions on the behavior on the other components. The generated assumption is added to the guarantees of the other components, which can generate assumptions on their own. This process continues until finding an assume-guarantee contract that makes all local synthesis problems realizable.

function CompSyntAGen $(\langle\varphi^c_{local}\rangle^n_{c=1}, \Psi_{shared}, \mathcal{B}_s, \langle\mathcal{B}_c\rangle^n_{c=1}, b_{init}, b_{max})$
1 **let** $\mathcal{A}^0_c$ be a safety automaton for $(2^{AP})^\omega$ for every $c \in \{1, \dots, n\}$
2 $b := b_{init}$
3 **while** $b \leq b_{max}$ **do**
4 $done :=$ true
5 **for** $c = 1, \dots, n$ **do**
6 $\overline{\mathcal{G}}_c :=$ Augment$(\bigcup_{\overline{c}\in\{1,\dots,n\}\setminus\{c\}} \overline{\mathcal{A}}^0_c, O_{\overline{c}})$; $\mathcal{U}_c :=$ UCB$_c(\mathcal{B}_c, \mathcal{B}_s, \mathcal{A}_c, \overline{\mathcal{G}}_c)$
7 $M_c :=$ LocalSynt$(\mathcal{U}_c, I, O_c, O_{\overline{c}}, Vals(\varphi^c_{local}), Vals(\Psi_{shared}), b)$
8 **if** $M_c = \bot$ **then**
9 $done :=$ false
10 $(\mathcal{A}^0_c, \mathcal{A}_c) :=$ GenAssumption$(\mathcal{U}_c, b, I, O_c, O_{\overline{c}}, \langle\mathcal{A}^0_c\rangle^n_{c=1}, \langle\mathcal{A}_c\rangle^n_{c=1})$
11 **if** $(\mathcal{A}^0_c, \mathcal{A}_c) = (\bot, \bot)$ **then** $restart :=$ true; **break**
12 **if** $done$ **then return** $\langle M_c\rangle^n_{c=1}$
13 **if** $restart$ **then** $b :=$ increment(b)
14 **return** unknown

Algorithm 3: Compositional synthesis for a combined specification $\Phi = comb(\varphi^1_{local}, \dots, \varphi^n_{local}, \Psi_{shared})$ with iterative assumption generation.The automata $\mathcal{B}_s = \mathcal{A}_{\Psi_{shared}, Vals(\Psi_{shared})}$ and $\mathcal{B}_c := \mathcal{A}_{\varphi^c_{local}, Vals(\varphi^c_{local})}$ are given as input.

Our compositional synthesis method with assumption generation is shown in Algorithm 3. Similarly to Algorithm 1, the method is based on incremental bounded synthesis. Here the guarantees are obtained from the assumptions, which initially permit any trace in $(2^{AP})^\omega$. When the current assumption is not sufficient for the local synthesis problem for some component c to be realizable, the procedure GenAssumption, shown in Algorithm 4 is invoked. We maintain two automata for the assumption for each component: $\mathcal{A}_c$ is the actual assumption, and the automaton $\mathcal{A}^0_c$ is used to represent the combination of the languages constructed from the assumption generation procedure before the transformation to $\mathcal{A}_c$. We explain this when we present the assumption generation.

Assumption generation. To find an assumption for a locally unrealizable (with the given bound) specification $\mathcal{U}_c$ for component c, GenAssumption considers the synthesis problem where all the components collaborate on realizing $\mathcal{U}_c$. In the resulting safety game all the output propositions $O_c \uplus O_{\overline{c}}$ are under the control of the output player. If the initial state is winning for the output player, then the winning region *Win* represents the most general cooperative strategy. Function ExtractAssumption uses *Win* to generate a new assumption $\mathcal{A}_{new}$, represented as a safety automaton. $\mathcal{A}_{new}$ must satisfy the following conditions:
(i) The local specification for component c constructed with the updated tuple of assumptions must be realizable by component c alone.
(ii) The combined assumption of component c and the newly generated guarantees of the other components must satisfy the conditions of Definition 2.
(iii) The updated tuple of assumptions $\langle\mathcal{A}_1, \dots, \mathcal{A}_c \cap \mathcal{A}_{new}, \dots, \mathcal{A}_n\rangle$ must be good-enough with respect to the shared specification Ψ_{shared}.

Additionally, we give preference to assumptions that do not unnecessarily restrict the other components, although we do not give guarantees on minimality.

function GenAssumption($\mathcal{U}_c, b, I, O_c, O_{\overline{c}}, \langle \mathcal{A}_c^0 \rangle_{c=1}^n, \langle \mathcal{A}_c \rangle_{c=1}^n$)

let $\mathcal{D}$ be a deterministic safety automaton for the language $(2^{AP})^\omega$
for $(u, w) \in U \times W$ **do**
  $\mathcal{U}_{(u,w)} := \mathsf{Instantiate}_c(\mathcal{U}_c, u, w)$; $\mathcal{D} := \mathsf{Safety}(\mathcal{U}_{(u,w)}, b, \mathcal{D})$
  $(\mathit{Win}, \mathit{strategy}) := \mathsf{SolveSafetyGame}(\mathcal{D}, I, O_c \uplus O_{\overline{c}})$
  if $\mathit{Win} := \bot$ **then return** $\bot$
  $\mathcal{D} := \mathsf{PruneLosing}(\mathcal{D}, \mathit{Win}, I, O_c \uplus O_{\overline{c}})$
$\mathit{Invalid} := \emptyset$
while true **do**
  $\mathcal{A}^0_{new} := \mathsf{ExtractAssumption}(\mathcal{D}, \mathit{Win}, I, O_c, O_{\overline{c}}, \mathit{Invalid})$
  if $\mathsf{GoodEnough}(\langle \mathcal{A}_1, \ldots, \mathsf{Augment}(\mathcal{A}_c^0 \cap \mathcal{A}^0_{new}, O_c), \ldots, \mathcal{A}_n \rangle, \mathcal{B}_s)$ **then**
    return $(\mathcal{A}_c^0 \cap \mathcal{A}^0_{new}, \mathsf{Augment}(\mathcal{A}_c^0 \cap \mathcal{A}^0_{new}, O_c))$
  $\mathit{Invalid} := \mathit{Invalid} \cup \{\mathcal{A}^0_{new}\}$

Algorithm 4: Generation of a good enough assumption for component c from the winning region in the cooperative synthesis game for the specification $\mathcal{U}_c$.

The function ExtractAssumption constructs an intermediate safety automaton $\mathcal{A}^0_{new}$ which is obtained from $\mathcal{D}$ and *Win*. It receives as additional input the set *Invalid*, which consists of the previously extracted assumptions that violate condition (iii) above. The extraction process checks against *Invalid* to avoid repeating the failed assumptions. We now give the construction of $\mathcal{A}^0_{new}$.

Let $\mathcal{D} = (2^{AP}, Q, \delta, Q_0, Q)$ be the deterministic safety automaton. Note that since the output player wins the safety game defined by $\mathcal{D}$, we have $Q_0 \subseteq \mathit{Win}$.

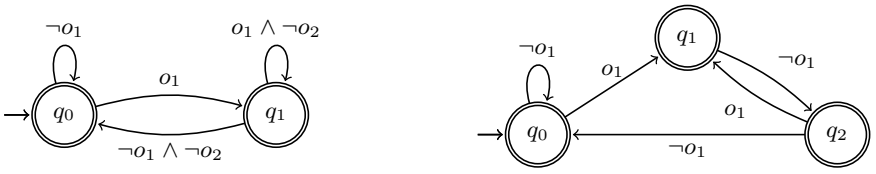
First we define a function $f_c : (Q \cap \mathit{Win}) \times 2^I \to 2^{O_c}$ such that for each $q \in Q \cap \mathit{Win}$, $i \in 2^I$ and all $\tilde{o} \in 2^{O_c}$ it holds that: $\{o_{\overline{c}} \in 2^{O_{\overline{c}}} \mid \exists q' \in \mathit{Win}.\ q' \in \delta(q, i \cup f_c(q,i)) \cup o_{\overline{c}})\}| \geq |\{o_{\overline{c}} \in 2^{O_{\overline{c}}} \mid \exists q' \in \mathit{Win}.\ q' \in \delta(q, i \cup \tilde{o} \cup o_{\overline{c}})\}|$.

That is, f_c maps each $q \in Q \cap \mathit{Win}$ and $i \in 2^I$ to the output of component c that allows for the maximal number of possible choices for the remaining components from $q \in Q \cap \mathit{Win}$ and $i \in 2^I$ landing in *Win*. This choice ensures local minimality of the restrictions on the other components.

Then, we define the function $f_{\overline{c}} : (Q \cap \mathit{Win}) \times 2^I \to 2^{2^{O_{\overline{c}}}}$ such that $f_{\overline{c}}(q,i) := \{o_{\overline{c}} \in 2^{O_{\overline{c}}} \mid \exists q' \in \mathit{Win}.\ q' \in \delta(q, i \cup f_c(q,i)) \cup o_{\overline{c}})\}$. Intuitively, $f_{\overline{c}}$ maps $q \in Q \cap \mathit{Win}$ and $i \in 2^I$ to the outputs of the components other than c that together with $f_c(q,i)$ lead to a state $q' \in \mathit{Win}$. Thus, the outputs of the other components are chosen such that they allow component c to realize $\mathcal{U}_c$ following f_c.

The automaton $\mathcal{A}^0_{new}$ is then constructed based on the function $f_{\overline{c}}$. We let $\mathcal{A}^0_{new} := (2^{AP}, Q \cap \mathit{Win}, \delta^0, Q_0, Q \cap \mathit{Win})$, where for every $q \in Q \cap \mathit{Win}$, $i \in 2^I$, $o_c \in 2^{O_c}$, $o_{\overline{c}} \in 2^{O_{\overline{c}}}$ and $q' \in Q \cap \mathit{Win}$ we have $q' \in \delta^0(q, i \cup o_c \cup o_{\overline{c}})$ if and only if $o_{\overline{c}} \in f_{\overline{c}}(q,i)$. Thus, the transition function δ^0 includes all transitions to states in $Q \cap \mathit{Win}$, where the output agrees with the function $f_{\overline{c}}$.

The automaton $\mathcal{A}^0_{new}$ satisfies condition (i). It does not necessarily satisfy (ii), since the outputs on the labels on the transitions in δ^0 are defined based on the function $f_{\overline{c}}$ that depends on f_c. Thus, it is possible that outputs of component c disagreeing with f_c could result in words rejected by $\mathcal{A}^0_{new}$, that is they will be violating the new assumption. To this end, we augment $\mathcal{A}^0_{new}$ by adding the



(a) Automaton $\mathcal{A}_1$ representing the assumption made by component 1 (guarantee provided by component 2).

(b) Automaton $\mathcal{A}_2$ representing the assumption made by component 2 (guarantee provided by component 1)

Figure 1: Assume-guarantee contract computed for Example 5

missing transitions, to ensure that such words are included in the language of the assumption automaton. We denote with $\mathsf{Augment}(\mathcal{A}_c^0 \cap \mathcal{A}_{new}^0, O_c)$ the augmented version of the new assumption, and with $\mathsf{Augment}(\bigcup_{\overline{c}\in\{1,\ldots,n\}\setminus\{c\}} \overline{\mathcal{A}}_c^0, O_{\overline{c}})$ the augmented updated guarantee.

With that, the updated assumption automaton for component c is obtained by constructing $\mathsf{Augment}(\mathcal{A}_c^0 \cap \mathcal{A}_{new}^0, O_c)$. What remains is to ensure condition (iii). The procedure GenAssumption keeps generating candidate assumptions until an updated assumption for c satisfies all the three conditions, or no more new assumptions can be generated from the given winning region *Win*. In such case it returns $(\bot, \bot)$, upon which CompSyntAGen restarts with larger bound.

Theorem 3 (Soundness). *Let Φ be a compositional LTL[$\mathcal{F}$] specification. If* CompSyntAGen *returns a multi-component reactive system $\mathcal{S} = (I, O, \mathcal{M})$, then $\mathcal{S}$ is a solution to* Problem 1.

We illustrate the process of iterative assumption generation on an example.

Example 5. Let $I = \{i\}, O_1 = \{o_1\}, O_2 = \{o_2\}$ and consider the specification $\Phi = \varphi^1_{local} \wedge \varphi^2_{local} \wedge \Psi_{shared}$, where the individual specifications are

$$\varphi^1_{local} = (\square\diamondsuit o_1) \oplus_{\frac{1}{2}} \square\diamondsuit(i \wedge \bigcirc \neg o_1), \qquad \varphi^2_{local} = \square\diamondsuit o_2,$$
$$\Psi_{shared} = \square(o_1 \rightarrow \bigcirc \neg o_2).$$

The first call to procedure LocalSynt for component 1 in CompSyntAGen determines that there exists no implementation for component 1 that is a solution to the local synthesis problem with bound 2. The local synthesis problem with specifications φ^1_{local} and Ψ_{shared} is not good-enough realizable by component 1 on its own, because the satisfaction of Ψ_{shared} depends on the future values of variable o_2, which is not under the control of component 1. In particular, since φ^1_{local} requires setting o_1 to true infinitely often in order to achieve a good enough satisfaction value, component 1 is unable to avoid the requirement that Ψ_{shared} puts on o_2. Thus, procedure GenAssumption is invoked to generate an assumption on the behavior of component 2. GenAssumption solves a safety game in which o_2 is controllable output. In this game, the system player has a winning strategy, from which the assumption $\mathcal{A}_1$ made by component 1,

depicted in Fig. 1a, is extracted. This assumption is added to the guarantee $\mathcal{G}_2$ of component 2, after which procedure LocalSynt is applied to component 2.

Again, there exists no implementation for component 2 that is a solution to the local good-enough synthesis problem with bound 2. The reason is that the local specification φ^2_{local} cannot be satisfied in conjunction with the guarantee $\mathcal{G}_2$ in case o_1 is always set to true. Following that, GenAssumption is invoked to generate an assumption on the behavior of component 1. GenAssumption solves the cooperative safety game constructed for bound 2, and generates the assumption that component 1 does not set o_1 to true twice in a row (as the bound is 2). The generated safety automaton $\mathcal{A}_2$ is given in Fig. 1b. With the assume-guarantee contract in Fig. 1, the subsequent calls to LocalSynt for both components succeed, producing a multi-component reactive system.

5 Experimental Evaluation

We implemented our compositional synthesis procedure in a prototype. The tool takes the compositional $LTL[\mathcal{F}]$ specification as input. If realizable, it produces a set of implementations, one for each component, that satisfy the conditions of *Problem 2*. Should the specification require cooperation between the components, the tool iteratively generates additional assumptions in the form of automata. Our tool uses *Spot* [14] (v2.10.6) for the automata operations and for solving the safety games in bounded synthesis.

We compare the compositional approach and the monolithic approach using our prototype, demonstrating the benefits of compositional synthesis for the same underlying implementation. We extract the UCB before starting the construction of the safety game to apply *sdf*[1] for reference, which takes this UCB as input and performs bounded synthesis. To our knowledge, there are currently no other tools available that would apply to our setting or could easily be extended.

We performed experiments on several examples on a laptop with an Intel Core i7 processor at 2.8 GHz and 16 GB of memory. Table 1 shows the results.

The first three examples are realizable without explicit assumptions. The others need additional assumptions, which can be given or derived. Example `intro_ex` is the one described in Section 1.

The results show that the compositional synthesis scales better than the monolithic approach on the considered benchmarks. In particular, most specifications can only be handled when treated in the decomposition. This is expected, as the decomposed specifications are smaller, and in the monolithic case the automata become prohibitively large. It should be noted that even in the cases when assumptions are necessary and require multiple iterations to be generated, the compositional approach is still faster than the monolithic one. For example `color_change`, it takes half the time to complete and the size of the UCB in the monolithic case is around the combined size for both components separately.

When increasing the number of components, the monolithic specification necessarily increases in size, whereas, depending on the specification, it can remain

[1] `https://github.com/5nizza/sdf-hoa`

Table 1: Experimental results, time in seconds, timeout of 30 minutes. Size is the size of initial automata for $(\varphi_l^1, \varphi_l^2, \ldots, \Psi_s)$, and largest final UCB. We differentiate between compositional with iterative assumptions (iter.), predefined assumptions (pred.), or no assumptions (none). In some cases (n/a.), there were no automata to execute sdf on.

Example	n	Autom. size		Monolithic	Compositional			Using sdf	
		init	UCB		none	iter.	pred.	mono.	comp.
max_use	2	13,7,7	108	335	25.73	-	-	TO	51.26
use_if_req2	2	17,17,17	671	TO	79.10	-	-	TO	TO
use_if_req3	3	17,17,17,29	2612	TO	1074.81	-	-	TO	TO
intro_ex	2	9,5,7	1115	290.37	-	57.73	17.6	TO	11.4
color_change	2	6,6,13	689	196.45	-	108.19	23.15	TO	34.8
two_foll_one	2	33,8,7	672	TO	-	442.74	215.12	TO	TO
perm_to_r3	3	7,7,8,13	3363	TO	-	286.42	194.84	n/a	TO
perm_to_r4	4	7,7,7,11,25	5558	TO	-	TO	TO	n/a	TO

small for the individual components. Still, constructing the safety game from the UCB is the most time-consuming step, and is performed for each component once per iteration. As the local synthesis problems become more complex, the timeout may be reached before iterating through all components, such as in `perm_to_r4`.

Also apparent from these results is that our prototype implementation does not scale well yet. Growing `use_if_req2` to three components in `use_if_req3` increases the runtime significantly. For `perm_to_r3` and `perm_to_r4`, it runs into the timeout going from three to four. It should be noted that for these benchmarks sdf reaches the timeout as well, when executed with the resulting UCB.

As expected, with assumptions given a priori, the performance of the compositional approach improves as each component is considered only once. The challenge here lies in manually finding suitable assumptions.

Lastly, when using sdf, we can only compare to the execution of our prototype without assumption generation. Applicable cases are when assumptions are not needed or are given, and the monolithic case. For larger automata, sdf was in most cases unable to finish within the timeout. This is expected as the extracted UCB encodes the behavior for all pairs of values.

6 Conclusion

We investigated the compositional synthesis problem for good-enough synthesis from specifications in a fragment of LTL[$\mathcal{F}$], considering fully-informed components with shared and local specifications. We identified sufficient conditions on the specifications that guarantee the soundness of the proposed decomposition rule. One of the directions for future work is the development of a more sophisticated analysis of the combination of local and shared specifications, to allow taking weights and other factors into account in the local synthesis problem. Another direction is the consideration of partial information. In order for the technique to be viable, we plan to improve the scalability of our prototype.

References

1. Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 424–439. Springer, 2014.
2. Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *J. ACM*, 63(3):24:1–24:56, 2016.
3. Shaull Almagor and Orna Kupferman. Good-enough synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020.
4. Shaull Almagor, Orna Kupferman, Jan Oliver Ringert, and Yaron Velner. Quantitative assume guarantee synthesis. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 353–374. Springer, 2017.
5. Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 26–33. IEEE, 2013.
6. Rajeev Alur, Salar Moarref, and Ufuk Topcu. Pattern-based refinement of assume-guarantee specifications in reactive synthesis. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 2015.
7. Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi, and Shufang Zhu. Compositional safety ltl synthesis. In Akash Lal and Stefano Tonetta, editors, *Verified Software. Theories, Tools and Experiments.*, pages 1–19, Cham, 2023. Springer International Publishing.
8. Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.
9. Roderick Bloem, Krishnendu Chatterjee, Swen Jacobs, and Robert Könighofer. Assume-guarantee synthesis for concurrent reactive programs with partial information. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2015.
10. Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017.

11. Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2007.
12. Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.
13. Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2014.
14. Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022.
15. Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods Syst. Des.*, 39(3):261–296, 2011.
16. Bernd Finkbeiner, Gideon Geier, and Noemi Passing. Specification decomposition for reactive synthesis. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2021.
17. Bernd Finkbeiner and Noemi Passing. Dependency-based compositional synthesis. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 2020.
18. Bernd Finkbeiner and Noemi Passing. Compositional synthesis of modular systems. *Innov. Syst. Softw. Eng.*, 18(3):455–469, 2022.
19. Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013.
20. Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.
21. Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Damien Zufferey. Assume-guarantee distributed synthesis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):3215–3226, 2020.