Compositional Synthesis of Modular Systems*

Bernd Finkbeiner and Noemi Passing

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{finkbeiner,noemi.passing}@cispa.de

Abstract. Given the advances in reactive synthesis, it is a natural next step to consider more complex multi-process systems. Distributed synthesis, however, is not yet scalable. Compositional approaches can be a game changer. Here, the challenge is to decompose a given specification of the global system behavior into requirements on the individual processes. In this paper, we introduce a compositional synthesis algorithm that, for each process, constructs, in addition to the implementation, a certificate that captures the necessary interface between the processes. The certificates then allow for constructing separate requirements for the individual processes. By bounding the size of the certificates, we can bias the synthesis procedure towards solutions that are desirable in the sense that the assumptions between the processes are small. Our experimental results show that our approach is much faster than standard methods for distributed synthesis as long as reasonably small certificates exist.

1 Introduction

In the last decade, there have been breakthroughs in terms of realistic applications and practical tools for reactive synthesis, demonstrating that concentrating on *what* a system should do instead of *how* it should be done is feasible. A natural next step is to consider complex multi-process systems. For distributed systems, though, there are no scalable tools that are capable of automatically synthesizing strategies from formal specifications for arbitrary system architectures.

For the scalability of verification algorithms, compositionality, i.e., breaking down the verification of a complex system into several smaller tasks over individual components, has proven to be a key technique [21]. For synthesis, however, developing compositional approaches is much more challenging: In practice, an individual process can rarely guarantee the satisfaction of the specification alone. Typically, there exist input sequences that prevent a process from satisfying the specification. The other processes in the system then ensure that these sequences are not produced. Thus, a process needs information about the strategies of the other processes to be able to satisfy the specification. Hence, distributed synthesis cannot easily be broken down into tasks over the individual processes.

*This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (No. 683300).

In this paper, we introduce a compositional synthesis algorithm addressing this problem by synthesizing additional *guarantees on the behavior* of every process. These guarantees, the so-called *certificates*, then provide essential information for the individual synthesis tasks: A strategy is only required to satisfy the specification if the other processes do not deviate from their guaranteed behavior. This allows for considering a process independent of the other processes' strategies. Our algorithm is an extension of bounded synthesis [14] that incorporates the search for certificates into the synthesis task for the strategies.

The benefits of synthesizing additional certificates are threefold. First, it *guides the synthesis procedure*: Bounded synthesis searches for strategies up to a given size. Beyond that, our algorithm introduces a bound on the size of the certificates. Hence, it bounds the size of the interface between the processes and thus the size of the assumptions made by them. By starting with small bounds and by only increasing them if the specification is unrealizable for the given bounds, the algorithm restricts synthesis to search for solutions with small interfaces.

Second, the certificates *increase the understandability* of the synthesized solution: It is challenging to recognize the interconnections in a distributed system. The certificates capture which information a process needs about the behavior of the other processes to be able to satisfy the specification, immediately encapsulating the system's interconnections. Furthermore, the certificates abstract from behavior that is irrelevant for the satisfaction of the specification. This allows for analyzing the strategies locally without considering the whole system's behavior.

Third, synthesizing certificates *enables modularity* of the system: The strategies only depend on the certificates of the other processes, not on their particular strategies. As long as the processes do not deviate from their certificates, the parallel composition of the strategies satisfies the specification. Hence, the certificates form a contract between the processes. After defining the contract, the strategies can be exchanged safely with other ones that respect the contract. Thus, strategies can be adapted flexibly without synthesizing a solution for the whole system again if requirements that do not affect the contract change.

We introduce two representations of certificates, as LTL formulas and as labeled transition systems. We show soundness and completeness of our certifying synthesis algorithm for both of them. Furthermore, we present a technique for determining *relevant processes* for each process. This allows us to reduce the number of certificates that a process has to consider to satisfy the specification while maintaining soundness and completeness. Focusing on the representation of certificates as transition systems, we present an algorithm for synthesizing certificates that is based on a reduction to a SAT constraint system.

We implemented the algorithm and compared it to an extension [2] of the synthesis tool BoSy [9] to distributed systems and to a compositional synthesis algorithm based on dominant strategies [7]. The results clearly demonstrate the advantage of synthesizing certificates: If solutions with a small interface between the processes exist, our algorithm outperforms the other synthesis tools significantly. Otherwise, the overhead of synthesizing additional guarantees is small.

Further details and all proofs are available in the full version of this paper [13].

Related Work: There are several approaches to compositional synthesis for monolithic systems [17,10,16,12,11]. As we are considering distributed systems, we focus on distributed synthesis algorithms. Assume-guarantee synthesis [5] is closest to our approach. There, each process provides a guarantee on its own behavior and makes an assumption on the behavior of the other processes. If there is a strategy for each process that satisfies the specification under the hypothesis that the other processes respect the assumption, and if its guarantee implies the assumptions of the other processes, a solution for the whole system is found. In contrast to our approach, most assume-guarantee synthesis algorithms [5,4,3,1] either rely on the user to provide the assumptions or require that a strategy profile on which the strategies can synchronize is constructed prior to synthesis.

A recent extension of assume-guarantee synthesis [19] algorithmically synthesizes assume-guarantee contracts for each process. In contrast to our approach, the guarantees do not necessarily imply the assumptions of the other processes. Thus, the algorithm needs to iteratively refine assumptions and guarantees until a valid contract is found. This iteration is circumvented in our algorithm since only assumptions that are guaranteed by the other processes are used.

Using a weaker winning condition for synthesis, remorse-free dominance [6], avoids the explicit construction of assumptions and guarantees [7]. The assumptions are implicit, but they do not always suffice. Thus, although a dependency analysis of the specification allows for solutions for further, more interconnected systems and specifications [12], compositional solutions do not always exist.

2 Running Example

In many modern factories, autonomous robots are a crucial component in the production line. The correctness of their implementation is essential and therefore they are a natural target for synthesis. Consider a factory with two robots that carry production parts from one machine to another. In the factory, there is a crossing that is used by both robots. The robots are required to prevent a crash: $\varphi_{safe} := \square\neg((at\_crossing_1 \wedge \bigcirc go_1) \wedge (at\_crossing_2 \wedge \bigcirc go_2))$, where $at\_crossing_i$ is an input variable denoting that robot r_i arrived at the crossing, and go_i is an output variable of robot r_i denoting that r_i moves ahead. Moreover, both robots need to cross the intersection at some point in time after arriving there: $\varphi_{cross_i} := \square(at\_crossing_i \rightarrow \bigcirc\diamondsuit go_i)$. In addition to these requirements, both robots have further objectives φ_{add_i} that are specific to their area of application. For instance, they may capture which machines have to be approached.

None of the robots can satisfy $\varphi_{safe} \wedge \varphi_{cross_i}$ alone: The crossing needs to be entered eventually by r_i but no matter when it is entered, r_j might enter it at the same time. Thus, strategies cannot be synthesized individually without information on the other robot's behavior. Due to φ_{add_i}, the parallel composition of the strategies can be large and complex. Hence, understanding why the overall specification is met and recognizing the individual strategies is challenging.

If both robots commit to their behavior at crossings, a robot r_i can satisfy $\varphi_{safe} \wedge \varphi_{cross_i}$ individually since it is allowed to assume that the other robot does

not deviate from its guaranteed behavior, the so-called certificate. For instance, if r_2 commits to always giving priority to r_1, entering the crossing regardless of r_2 satisfies $\varphi_{safe} \wedge \varphi_{cross_1}$ for r_1. If r_1 guarantees to not block crossings, r_2 can satisfy $\varphi_{safe} \wedge \varphi_{cross_2}$ as well. Hence, if both robots can satisfy the whole part of the specification that affects them, i.e., $\varphi_i = \varphi_{safe} \wedge \varphi_{cross_i} \wedge \varphi_{add_i}$, under the assumption that the other robot sticks to its certificate, then the parallel composition of their strategies satisfies the whole specification. Furthermore, we then know that the robots do not interfere in any other situation. Thus, the certificates provide insight in the required communication of the robots.

Moreover, when analyzing the strategy s_i of r_i, only taking r_j's certificate into account abstracts away r_j's behavior aside from crossings. This allows us to focus on the relevant aspects of r_j's behavior for r_i, making it significantly easier to understand why r_i's strategy satisfies φ_i. Lastly, the certificates form a contract of safe behavior at crossings: If r_i's additional objectives change, it suffices to synthesize a new strategy for r_i. Provided r_i does not change its behavior at crossings, r_j's strategy can be left unchanged.

3 Preliminaries

Notation. In the following, we denote the prefix of length t of an infinite word $\sigma = \sigma_1\sigma_2 \cdots \in (2^V)^\omega$ by $\sigma_{..t} := \sigma_1 \dots \sigma_t$. Moreover, for a set X and an infinite word $\sigma = \sigma_1\sigma_2 \cdots \in (2^V)^\omega$, we define $\sigma \cap X = (\sigma_1 \cap X)(\sigma_2 \cap X) \cdots \in (2^X)^\omega$.

LTL. Linear-time temporal logic (LTL) [20] is a specification language for linear-time properties. Let Σ be a finite set of atomic propositions and let $a \in \Sigma$. The syntax of LTL is given by $\varphi, \psi ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \psi$. We define $\diamondsuit\varphi = \mathit{true}\,\mathcal{U}\,\varphi$, and $\square\varphi = \neg\diamondsuit\neg\varphi$ and use the standard semantics. The language $\mathcal{L}(\varphi)$ of a formula φ is the set of infinite words that satisfy φ. The atomic propositions in φ are denoted by $\mathrm{prop}(\varphi)$. We represent a formula $\varphi = \xi_1 \wedge \cdots \wedge \xi_k$ also by the set of its conjuncts, i.e., $\varphi = \{\xi_1, \dots, \xi_k\}$.

Automata. A universal co-Büchi automaton $\mathcal{A} = (Q, q_0, \delta, F)$ over a finite alphabet Σ consists of a finite set of states Q, an initial state $q_0 \in Q$, a transition relation $\delta : Q \times 2^\Sigma \times Q$, and a set $F \subseteq Q$ of rejecting states. For an infinite word $\sigma = \sigma_0\sigma_1 \cdots \in (2^\Sigma)^\omega$, a run of σ on $\mathcal{A}$ is an infinite sequence $q_0q_1 \cdots \in Q^\omega$ of states with $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \geq 0$. A run is accepting if it contains only finitely many visits to rejecting states. $\mathcal{A}$ accepts a word σ if all runs of σ on $\mathcal{A}$ are accepting. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all accepted words. An LTL specification φ can be translated into an equivalent universal co-Büchi automaton $\mathcal{A}_\varphi$, i.e., with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$, with a single exponential blow up [18].

Architectures. An architecture is a tuple $A = (P, V, I, O)$, where P is a set of processes consisting of the environment process *env* and a set of n system processes $P^- = P \setminus \{env\}$, V is a set of variables, $I = \langle I_1, \dots, I_n \rangle$ assigns a set $I_j \subseteq V$ of input variables to each system process p_j, and $O = \langle O_{env}, O_1, \dots O_n \rangle$

assigns a set $O_j \subseteq V$ of output variables to each process p_j. For all $p_j, p_k \in P^-$ with $j \neq k$, we have $I_j \cap O_j = \emptyset$ and $O_j \cap O_k = \emptyset$. The variables V_j of $p_j \in P^-$ are its inputs and outputs, i.e., $V_j = I_j \cup O_j$. The variables V of the whole system are defined by $V = \bigcup_{p_j \in P^-} V_j$. We define $inp = \bigcup_{p_j \in P^-} I_j$ and $out = \bigcup_{p_j \in P^-} O_j$. An architecture is called distributed if $|P^-| \geq 2$ and monolithic otherwise. In the remainder of this paper, we assume that a distributed architecture is given.

Transition Systems. Given sets I and O of input and output variables, a Moore transition system (TS) $\mathcal{T} = (T, t_0, \tau, o)$ consists of a finite set of states T, an initial state t_0, a transition function $\tau : T \times 2^I \to T$, and a labeling function $o : T \to 2^O$. For an input sequence $\gamma = \gamma_0\gamma_1 \ldots \in (2^I)^\omega$, $\mathcal{T}$ produces a path $\pi = (t_0, \gamma_0 \cup o(t_0))(t_1, \gamma_1 \cup o(t_1)) \ldots \in (T \times 2^{I \cup O})^\omega$, where $(t_j, \gamma_j, t_{j+1}) \in \tau$. The projection of a path to the variables is called trace. The parallel composition of two TS $\mathcal{T}_1 = (T_1, t_0^1, \tau_1, o_1)$, $\mathcal{T}_2 = (T_2, t_0^2, \tau_2, o_2)$, is a TS $\mathcal{T}_1 \,||\, \mathcal{T}_2 = (T, t_0, \tau, o)$ with $T = T_1 \times T_2$, $t_0 = (t_0^1, t_0^2)$, $\tau((t, t'), \boldsymbol{i}) = (\tau_1(t, (\boldsymbol{i}_1 \cup o_2(t')) \cap I_1), \tau_2(t', (\boldsymbol{i}_2 \cup o_1(t)) \cap I_2))$, and $o((t, t')) = o_1(t) \cup o_2(t')$. A TS $\mathcal{T}_1 = (T_1, t_0^1, \tau_1, o_1)$ over I and O_1 *simulates* $\mathcal{T}_2 = (T_2, t_0^2, \tau_2, o_2)$ over I and O_2 with $O_1 \subseteq O_2$, denoted $\mathcal{T}_2 \preceq \mathcal{T}_1$, if there is a simulation relation $R : T_2 \times T_1$ with $(t_0^2, t_0^1) \in R$, $\forall (t_2, t_1) \in R.\ o(t_2) \cap O_1 = o(t_1)$, and $\forall t_2' \in T_2. \forall \boldsymbol{i} \in 2^I.\ (\tau_2(t_2, \boldsymbol{i}) = t_2') \rightarrow (\exists t_1' \in T_1.\ \tau_1(t_1, \boldsymbol{i}) = t_1' \wedge (t_2', t_1') \in R)$.

Strategies. We model a strategy s_i of $p_i \in P^-$ as a Moore transition system $\mathcal{T}_i$ over I_i and O_i. The trace produced by $\mathcal{T}_i$ on $\gamma \in (2^{I_i})^\omega$ is called the *computation* of s_i on γ, denoted $comp(s_i, \gamma)$. For an LTL formula φ over V, s_i satisfies φ, denoted $s_i \models \varphi$, if $comp(s, \gamma) \cup \gamma' \models \varphi$ holds for all $\gamma \in (2^{I_i})^\omega$, $\gamma' \in (2^{V \setminus V_i})^\omega$.

Synthesis. For a specification φ, synthesis derives strategies $s_1, \ldots, s_n$ for the system processes such that $s_1 \,||\, \ldots \,||\, s_n \models \varphi$ holds. If such strategies exist, φ is realizable in the architecture. Bounded synthesis [14] additionally bounds the size of the strategies. The search for strategies is encoded into a constraint system that is satisfiable if, and only if, φ is realizable for the bound. There are SMT, SAT, QBF, and DQBF encodings for monolithic [8] and distributed [2] architectures.

4 Compositional Synthesis with Certificates

In this section, we describe a sound and complete compositional synthesis algorithm for distributed systems. The main idea is to synthesize strategies for the system processes individually. Hence, in contrast to classical distributed synthesis, where strategies $s_1, \ldots, s_n$ are synthesized such that $s_1 \,||\, \ldots \,||\, s_n \models \varphi$ holds, we require that $s_i \models \varphi_i$ holds for all system processes $p_i \in P^-$. Here, φ_i is a subformula of φ that, intuitively, captures the part of φ that affects p_i. As long as φ_i contains all parts of φ that restrict the behavior of s_i, the satisfaction of φ by the parallel composition of all strategies is guaranteed. Computing specification decompositions is not the main focus of this paper; in fact, our algorithm can be used with any decomposition that fulfills the above requirement. There is work on obtaining small subspecifications, e.g., [11], we, however, use an easy decomposition algorithm in the remainder of this paper for simplicity:

Definition 1 (Specification Decomposition). *Let $\varphi = \xi_1 \wedge \dots \wedge \xi_k$ be an LTL formula. The* decomposition of φ *is a vector $\langle \varphi_1, \dots, \varphi_n \rangle$ of LTL formulas with $\varphi_i = \{\xi_j \mid \xi_j \in \varphi \wedge (\mathrm{prop}(\xi_j) \cap O_i \neq \emptyset \vee \mathrm{prop}(\xi_j) \cap \mathit{out} = \emptyset)\}$.*

Intuitively, the subspecification φ_i contains all conjuncts of φ that contain outputs of p_i as well as all input-only conjuncts. In the remainder of this paper, we assume that both $\mathrm{prop}(\varphi) \subseteq V$ and $\mathcal{L}(\varphi) \in (2^V)^\omega$ hold for all specifications φ. Then, every atomic proposition occurring in a formula φ is an input or output of at least one system process and thus $\bigwedge_{p_i \in P^-} \varphi_i = \varphi$ holds.

Although we decompose the specification, a process p_i usually cannot guarantee the satisfaction of φ_i alone; rather, it depends on the cooperation of the other processes. For instance, robot r_1 from Section 2 cannot guarantee that no crash will occur when entering the crossing since r_2 can enter it at the same point in time. Thus, we additionally synthesize a *guarantee on the behavior* of each process, the so-called *certificate*. The certificates then provide essential information to the processes: If p_i commits to a certificate, the other processes can rely on p_i's strategy to not deviate from this behavior. In particular, the strategies only need to satisfy the specification as long as the other processes stick to their certificates. Thus, a process is not required to react to *all* behaviors of the other processes but only to those that truly occur when the processes interact.

In this section, we represent the certificate of $p_i \in P^-$ by an LTL formula ψ_i. For instance, robot r_2 may guarantee to always give priority to r_1 at crossings, yielding the certificate $\psi_2 = \square((\mathit{at\_crossing}_1 \wedge \mathit{at\_crossing}_2) \rightarrow \bigcirc \neg \mathit{go}_2)$. Since r_1 can assume that r_2 does not deviate from its certificate ψ_2, a strategy for r_1 that enters crossings regardless of r_2 satisfies $\varphi_{\mathit{safe}} \wedge \varphi_{\mathit{cross}_1}$.

To ensure that p_i does not deviate from its own certificate, we require its strategy s_i to satisfy the LTL formula ψ_i describing it. To model that s_i only has to satisfy its specification if the other processes stick to their certificates, it has to satisfy $\Psi_i \rightarrow \varphi_i$, where $\Psi_i = \{\psi_j \mid p_j \in P^- \setminus \{p_i\}\}$, i.e., Ψ_i is the conjunction of the certificates of the other processes. Using this, we define certifying synthesis:

Definition 2 (Certifying Synthesis). *Let φ be an LTL formula with decomposition $\langle \varphi_1, \dots, \varphi_n \rangle$.* Certifying synthesis *derives strategies $s_1, \dots, s_n$ and LTL certificates $\psi_1, \dots, \psi_n$ for the system processes such that $s_i \models \psi_i \wedge (\Psi_i \rightarrow \varphi_i)$ holds for all $p_i \in P^-$, where $\Psi_i = \{\psi_j \mid p_j \in P^- \setminus \{p_i\}\}$.*

Classical distributed synthesis algorithms reason *globally* about the satisfaction of the full specification by the parallel composition of the synthesized strategies. Certifying synthesis, in contrast, reasons *locally* about the satisfaction of the subspecifications for the individual processes, i.e., without considering the parallel composition of the strategies. This greatly improves the understandability of the correctness of synthesized solutions since we are able to consider the strategies separately. Furthermore, local reasoning is still sound and complete:

Theorem 1 (Soundness and Completeness). *Let φ be an LTL formula and let $\mathcal{S} = \langle s_1, \dots, s_n \rangle$ be a vector of strategies for the system processes. There exists a vector $\Psi = \langle \psi_1, \dots, \psi_n \rangle$ of LTL certificates such that $(\mathcal{S}, \Psi)$ is a solution of certifying synthesis for φ if, and only if $s_1 \,||\, \dots \,||\, s_n \models \varphi$ holds.*

Soundness of certifying synthesis follows from the fact that every system process is required to satisfy its own certificate. Completeness is obtained since every strategy can serve as its own certificate: Intuitively, if $s_1 \,||\, \ldots \,||\, s_n \models \varphi$, then LTL certificates that capture the exact behavior of the corresponding strategy satisfy the requirements of certifying synthesis. The proof is given in [13].

Thus, certifying synthesis enables local reasoning and therefore better understandability of the solution as well as modularity of the system, while ensuring to find correct solutions for all specifications that are realizable in the architecture. Furthermore, the parallel composition of the strategies obtained with certifying synthesis for a specification φ is a solution for the whole system.

5 Certifying Synthesis with Deterministic Certificates

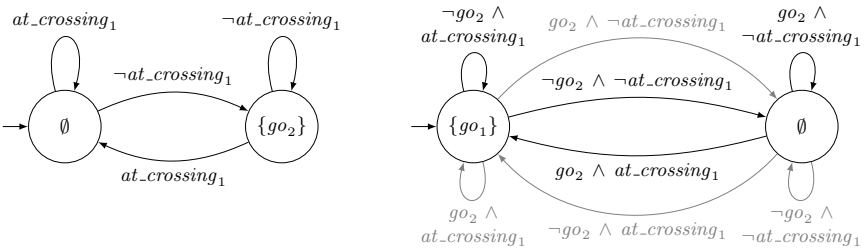
There are several quality measures for certificates, for instance their size. We, however, focus on certificates that are *easy to synthesize*: To determine whether a strategy sticks to its own certificate, a check for language containment has to be performed. Yet, efficient algorithms only exist for deterministic properties [23]. While certificates represented by LTL formulas are easily human-readable, they can be nondeterministic. Thus, the ω-automaton representing the LTL certificate needs to be determinized, yielding an exponential blowup in its size [22].

In this section, we introduce a representation of certificates that ensures determinism to avoid the blowup. Note that while enforcing determinism might yield larger certificates, it does not rule out any strategy that can be found with nondeterministic certificates: Since strategies are per se deterministic, there exists at least one deterministic certificate for them: The strategy itself.

We model the guaranteed behavior g_i of a system process p_i as a labeled transition system $\mathcal{T}_i^G$, called *guarantee transition system* (GTS), over inputs I_i and *guarantee output variables* $O_i^G \subseteq O_i$. Only considering a subset of O_i as output variables allows the certificate to abstract from outputs of p_i whose valuation is irrelevant for all other system processes. In the following, we assume the guarantee output variables of p_i to be both an output of p_i and an input of some other system process, i.e., $O_i^G := O_i \cap \mathit{inp}$. Intuitively, a variable $v \in O_i \setminus O_i^G$ cannot be observed by any other process. Thus, a guarantee on its behavior does not influence any other system process and hence it can be omitted. The variables V_i^G of the GTS of $p_i \in P^-$ are then given by $V_i^G := I_i \cup O_i^G$.

In certifying synthesis, it is essential that a strategy only needs to satisfy the specification if the other processes do not deviate from their certificates. In the previous section, we used an implication in the local objective to model this. When representing certificates as transition systems, we use *valid histories* to determine whether a sequence matches the certificates of the other processes.

Definition 3 (Valid History). *Let $\mathcal{G}_i$ be a set of guarantee transition systems. A* valid history of length t with respect to $\mathcal{G}_i$ *is a finite sequence $\sigma \in (2^V)^*$ of length t, where for all $g_j \in \mathcal{G}_i$, $\sigma_k \cap O_j^G = \mathit{comp}(g_j, \hat{\sigma} \cap I_j)_k \cap O_j^G$ holds for all points in time k with $1 \leq k \leq t$ and all infinite extensions $\hat{\sigma}$ of σ. The set of all valid histories of length t with respect to $\mathcal{G}_i$ is denoted by $\mathcal{H}_{\mathcal{G}_i}^t$.*



(a) Guarantee transition system for r_2 (b) Strategy transition system for r_1

Fig. 1: Strategy and GTS for robots r_1 and r_2 from Section 2, respectively. The labels of the states denote the output of the TS in the respective state.

Intuitively, a valid history respecting a set $\mathcal{G}_i$ of guarantee transition systems is a finite sequence that is a prefix of a computation of all GTS in $\mathcal{G}_i$. Thus, a valid history can be produced by the parallel composition of the GTS. Note that since strategies cannot look into the future, a finite word satisfies the requirements of a valid history either for all of its infinite extensions or for none of them.

As an example for valid histories, consider the manufacturing robots again. Assume that r_2 guarantees to always give priority to r_1 at crossings and to move forward if r_1 is not at the crossing. A GTS g_2 for r_2 is depicted in Figure 1a. Since r_2 never outputs go_2 if r_1 is at the crossing (left state), the finite sequence $\{at\_crossing_1\}\{go_2\}$ is no valid history respecting g_2. Since r_2 outputs go_2 otherwise (right state), e.g., $\{at\_crossing_2\}\{go_2\}$ is a valid history respecting g_2.

We use valid histories to determine whether the other processes stick to their certificates. Thus, intuitively, a strategy is required to satisfy the specification if its computation is a valid history respecting the GTS of the other processes:

Definition 4 (Local Satisfaction). *Let $\mathcal{G}_i$ be a set of guarantee transition systems. A strategy s_i for $p_i \in P^-$* locally satisfies *an LTL formula φ_i* with respect to $\mathcal{G}_i$*, denoted $s_i \models_{\mathcal{G}_i} \varphi_i$, if $comp(s_i, \gamma) \cup \gamma' \models \varphi_i$ holds for all $\gamma \in (2^{I_i})^\omega$, $\gamma' \in (2^{V \setminus V_i})^\omega$ with $comp(s_i, \gamma)_{..t} \cup \gamma'_{..t} \in \mathcal{H}^t_{\mathcal{G}_i}$ for all points in time t.*

If r_2, for instance, sticks to its guaranteed behavior g_2 depicted in Figure 1a, then r_1 can enter crossings regardless of r_2. Such a strategy s_1 for r_1 is shown in Figure 1b. Since neither $\sigma := \{at\_crossing_1\}\{go_2\}$ nor any finite sequence containing σ is a valid history respecting g_2, no transition for input go_2 has to be considered for local satisfaction when r_1 is at the crossing (left state of s_1). Therefore, these transitions are depicted in gray. Analogously, no transition for $\neg go_2$ has to be considered when r_1 is not at the crossing (right state). The other transitions match valid histories and thus they are taken into account. Since no crash occurs when considering the black transitions only, $s_1 \models_{\{g_2\}} \varphi_{safe}$ holds.

Using local satisfaction, we now define certifying synthesis in the setting where certificates are represented by labeled transition systems: Given an architecture A and a specification φ, certifying synthesis for φ derives strategies $s_1, \ldots, s_n$ and *guarantee transition systems* $g_1, \ldots, g_n$ for the system processes.

For all $p_i \in P^-$, we require s_i to locally satisfy its specification with respect to the guarantee transition systems of the other processes, i.e., $s_i \models_{\mathcal{G}_i} \varphi_i$, where $\mathcal{G}_i = \{g_j \mid p_j \in P^- \setminus \{p_i\}\}$. To ensure that a strategy does not deviate from its own certificate, g_i is required to simulate s_i, i.e., $s_i \preceq g_i$ needs to hold.

In the following, we show that solutions of certifying synthesis with LTL certificates can be translated into solutions with GTS and vice versa. Given a solution of certifying synthesis with GTS, the main idea is to construct LTL certificates that capture the exact behavior of the GTS. For the formal certificate translation and its proof of correctness, we refer to [13].

Lemma 1. *Let φ be an LTL formula. Let $\mathcal{S}$ and $\mathcal{G}$ be vectors of strategies and guarantee transition systems, respectively, for the system processes. If $(\mathcal{S}, \mathcal{G})$ is a solution of certifying synthesis for φ, then there exists a vector Ψ of LTL certificates such that $(\mathcal{S}, \Psi)$ is a solution for certifying synthesis for φ as well.*

Given a solution of certifying synthesis with LTL certificates, we can construct GTS that match the strategies of the given solution. Then, these strategies as well as the GTS form a solution of certifying synthesis with GTS. The full construction and its proof of correctness is given in [13].

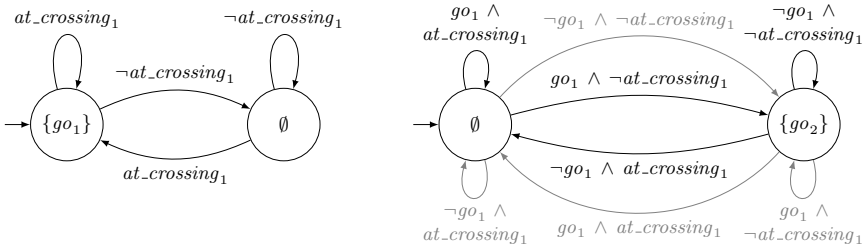
Lemma 2. *Let φ be an LTL formula. Let $\mathcal{S}$ and Ψ be vectors of strategies and LTL certificates, respectively, for the system processes. If $(\mathcal{S}, \Psi)$ is a solution of certifying synthesis for φ, then there exists a vector $\mathcal{G}$ of guarantee transition system such that $(\mathcal{S}, \mathcal{G})$ is a solution for certifying synthesis for φ as well.*

Hence, we can translate solutions of certifying synthesis with LTL formulas and with GTS into each other. Thus, we can reuse the results from Section 4, in particular Theorem 1, and then soundness and completeness of certifying synthesis with guarantee transition systems follows with Lemmas 1 and 2:

Theorem 2 (Soundness and Completeness with GTS). *Let φ be an LTL formula. Let $\mathcal{S} = \langle s_1, \dots, s_n \rangle$ be a vector of strategies for the system processes. Then, there exists a vector $\mathcal{G}$ of guarantee transition systems such that $(\mathcal{S}, \mathcal{G})$ is a solution of certifying synthesis for φ if, and only if, $s_1 \,||\, \dots \,||\, s_n \models \varphi$ holds.*

Thus, similar to LTL certificates, certifying synthesis with GTS allows for local reasoning and thus enables modularity of the system while it still ensures that correct solutions for all realizable specifications are found. In particular, enforcing deterministic certificates does not rule out strategies that can be obtained with either nondeterministic certificates or with classical distributed synthesis.

As an example of the whole synthesis procedure of a distributed system with certifying synthesis and GTS, consider the manufacturing robots from Section 2. For simplicity, suppose that the robots do not have individual additional requirements φ_{add_i}. Hence, the full specification is given by $\varphi_{safe} \wedge \varphi_{cross_1} \wedge \varphi_{cross_2}$. Since go_i is an output variable of robot r_i, we obtain the subspecifications $\varphi_i = \varphi_{safe} \wedge \varphi_{cross_i}$. A solution of certifying synthesis is then given by the strategies and GTS depicted in Figures 1 and 2. Note that s_2 only locally satisfies φ_{cross_2} with respect to g_1 when assuming that r_1 is not immediately again



(a) Guarantee transition system for r_1 (b) Strategy transition system for r_2

Fig. 2: GTS and strategy for robots r_1 and r_2 from Section 2, respectively. The labels of the states denote the output of the TS in the respective state.

at the intersection after crossing it. However, there are solutions with slightly more complicated certificates that do not need this assumption. The parallel composition of s_1 and s_2 yields a strategy that allows r_1 to move forwards if it is at the crossing and that allows r_2 to move forwards otherwise.

6 Computing Relevant Processes

Both representations of certificates introduced in the last two sections consider the certificates of *all* other system processes in the local objective of every system process p_i. This is not always necessary since in some cases φ_i is satisfiable by a strategy for p_i even if another process deviates from its guaranteed behavior.

In this section, we present an optimization of certifying synthesis that reduces the number of considered certificates. We compute a set of *relevant processes* $\mathcal{R}_i \subseteq P^- \setminus \{p_i\}$ for every $p_i \in P^-$. Certifying synthesis then only considers the certificates of the relevant processes: For LTL certificates, it requires that $s_i \models \psi_i \wedge (\Psi_i^{\mathcal{R}} \rightarrow \varphi_i)$ holds, where $\Psi_i^{\mathcal{R}} = \{\psi_j \in \Psi \mid p_j \in \mathcal{R}_i\}$. For GTS, both $s_i \preceq g_i$ and $s_i \models_{\mathcal{G}_i^{\mathcal{R}}} \varphi_i$ need to hold, where $\mathcal{G}_i^{\mathcal{R}} = \{g_j \in \mathcal{G} \mid p_j \in \mathcal{R}_i\}$. Such solutions of certifying synthesis are denoted by $(\mathcal{S}, \Psi)_{\mathcal{R}}$ and $(\mathcal{S}, \mathcal{G})_{\mathcal{R}}$, respectively.

The construction of the relevant processes $\mathcal{R}_i$ has to ensure that certifying synthesis is still sound and complete. In the following, we introduce a definition of relevant processes that does so. It excludes processes from p_i's set of relevant processes $\mathcal{R}_i$ whose output variables do not occur in the subspecification φ_i:

Definition 5 (Relevant Processes). *Let φ be an LTL formula with decomposition $\langle \varphi_1, \ldots, \varphi_n \rangle$. The* relevant processes $\mathcal{R}_i \subseteq P^- \setminus \{p_i\}$ *of system process $p_i \in P^-$ are given by $\mathcal{R}_i = \{p_j \in P^- \setminus \{p_i\} \mid O_j \cap \mathrm{prop}(\varphi_i) \neq \emptyset\}$.*

Intuitively, since $O_j \cap \mathrm{prop}(\varphi_i) = \emptyset$ holds for a process $p_j \in P^- \setminus \mathcal{R}_i$ with $i \neq j$, the subspecification φ_i does not restrict the satisfying valuations of the output variables of p_j. Thus, in particular, if a sequence satisfies φ_i, then it does so for any valuations of the variables in O_j. Hence, the guaranteed behavior of p_j does not influence the satisfiability of φ_i and thus p_i does not need to consider it. The proof of the following theorem stating this property is given in [13].

Theorem 3 (Correctness of Relevant Processes). *Let φ be an LTL formula. Let $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$ be a vector of strategies for the system processes.*

1. *Let Ψ be a vector of LTL certificates. If $(\mathcal{S}, \Psi)_{\mathcal{R}}$ is a solution of certifying synthesis for φ, then $s_1 \,||\, \ldots \,||\, s_n \models \varphi$ holds. If $s_1 \,||\, \ldots \,||\, s_n \models \varphi$ holds, then there exists a vector Ψ' of LTL certificates and a vector $\mathcal{S}'$ of strategies such that $(\mathcal{S}', \Psi')_{\mathcal{R}}$ is a solution of certifying synthesis for φ.*
2. *Let $\mathcal{G}$ be a vector of guarantee transition systems. If $(\mathcal{S}, \mathcal{G})_{\mathcal{R}}$ is a solution of certifying synthesis for φ, then $s_1 \,||\, \ldots \,||\, s_n \models \varphi$. If $s_1 \,||\, \ldots \,||\, s_n \models \varphi$ holds, then there exists a vector $\mathcal{G}'$ of guarantee transition systems and a vector $\mathcal{S}'$ of strategies such that $(\mathcal{S}', \mathcal{G}')_{\mathcal{R}}$ is a solution of certifying synthesis for φ.*

Note that for certifying synthesis with relevant processes, we can only guarantee that for every vector of strategies $\langle s_1, \ldots, s_n \rangle$ whose parallel composition satisfies the specification, there exist *some* strategies that are a solution of certifying synthesis. These strategies are not necessarily $s_1, \ldots, s_n$: A strategy s_i may make use of the certificate of a process p_j outside of $\mathcal{R}_i$. That is, it may violate its specification φ_i on an input sequence that does not stick to g_j although φ_i is satisfiable for this input. Strategy s_i is not required to satisfy φ_i on this input, a strategy that may only consider the certificates of the relevant processes, however, is. As long as the definition of relevant processes allows for finding *some* solution of certifying synthesis, like the one introduced in this section does as a result of Theorem 3, certifying synthesis is nevertheless sound and complete.

7 Synthesizing Certificates

In this section, we describe an algorithm for practically synthesizing strategies and deterministic certificates represented by GTS. Our approach is based on *bounded synthesis* [14] and bounds the size of the strategies and of the certificates. This allows for producing size-optimal solutions in either terms of strategies or certificates. Like for monolithic bounded synthesis [14,8], we encode the search for a solution of certifying synthesis of a certain size into a SAT constraint system. We reuse parts of the constraint system for monolithic systems.

An essential part of bounded synthesis is to determine whether a strategy satisfies an LTL formula φ_i. To do so, we first construct the equivalent universal co-Büchi automaton $\mathcal{A}_i$ with $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}(\varphi_i)$. Then, we check whether $\mathcal{A}_i$ accepts $\mathit{comp}(s_i, \gamma) \cup \gamma'$ for all $\gamma \in (2^{I_i})^\omega$, $\gamma' \in (2^{V \setminus V_i})^\omega$, i.e., whether all runs of $\mathcal{A}_i$ induced by $\mathit{comp}(s_i, \gamma) \cup \gamma'$ contain only finitely many visits to rejecting states. So far, we used *local satisfaction* to formalize that in compositional synthesis with GTS a strategy only needs to satisfy its specification as long as the other processes stick to their guarantees. That is, we changed the satisfaction condition. To reuse existing algorithms for bounded synthesis and, in particular, for checking whether a strategy is winning, however, we incorporate this property of certifying synthesis into the labeled transition system representing the strategy instead. In fact, we utilize the following observation: A finite run of a universal co-Büchi automaton can never visit a rejecting state infinitely often. Hence, by

ensuring that the automaton produces finite runs on all sequences that deviate from a guarantee, checking whether a strategy satisfies a specification can still be done by checking whether the runs of the corresponding automaton induced by the computations of the strategy visit a rejecting state only finitely often.

Therefore, we represent strategies by *incomplete* transition systems in the following. The domain of definition of their transition function is defined such that the computation of a strategy is infinite if, and only if, the other processes stick to their guarantees. To formalize this, we utilize valid histories:

Definition 6 (Local Strategy). *A* local strategy s_i *for process* $p_i \in P^-$ with respect to *a set* $\mathcal{G}_i$ *of GTS is represented by a TS* $\mathcal{T}_i = (T, t_0, \tau, o)$ *with a partial transition function* $\tau : T \times 2^{I_i} \rightharpoonup T$*. The domain of definition of* τ *is defined such that* $comp(s_i, \gamma)$ *is infinite for* $\gamma \in (2^{I_i})^\omega$ *if, and only if, there exists* $\gamma' \in (2^{V \setminus V_i})^\omega$ *such that* $comp(s_i, \gamma)_{..t} \cup \gamma'_{..t} \in \mathcal{H}^t_{\mathcal{G}_i}$ *holds for all points in time* t*.*

As an example, consider strategy s_1 for robot r_1 and guarantee transition system g_2 for robot r_2, both depicted in Figure 1, again. From s_1, we can construct a local strategy s'_1 for r_1 with respect to g_2 by eliminating the gray transitions.

We now define certifying synthesis with local strategies: Given a specification φ, certifying synthesis derives GTS $g_1, \dots, g_n$ and *local strategies* $s_1, \dots, s_n$ respecting these guarantees, such that for all $p_i \in P^-$, $s_i \preceq g_i$ holds and all runs of $\mathcal{A}_i$ induced by $comp(s_i, \gamma) \cup \gamma'$ contain finitely many visits to rejecting states for all $\gamma \in (2^{I_i})^\omega$, $\gamma' \in (2^{V \setminus V_i})^\omega$, where $\mathcal{A}_i$ is a universal co-Büchi automaton with $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}(\varphi_i)$. Thus, we can reuse existing algorithms for checking satisfaction of a formula in our certifying synthesis algorithm when synthesizing local strategies instead of complete ones. Similar to monolithic bounded synthesis, we construct a constraint system encoding the search for local strategies and GTS:

Theorem 4. *Let* A *be an architecture, let* φ *be an LTL formula, and let* $\mathcal{B}$ *be the size bounds. There is a SAT constraint system* $\mathcal{C}_{A,\varphi,\mathcal{B}}$ *such that (1) if* $\mathcal{C}_{A,\varphi,\mathcal{B}}$ *is satisfiable, then* φ *is realizable in* A*, (2) if* φ *is realizable in* A *for the bounds* $\mathcal{B}$ *and additionally* $\text{prop}(\varphi_i) \subseteq V_i$ *holds for all* $p_i \in P^-$*, then* $\mathcal{C}_{A,\varphi,\mathcal{B}}$ *is satisfiable.*

Intuitively, the constraint system $\mathcal{C}_{A,\varphi,\mathcal{B}}$ consists of n slightly adapted copies of the SAT constraint system for monolithic systems [14,8] as well as additional constraints that ensure that the synthesized local strategies correspond to the synthesized guarantees and that they indeed fulfill the conditions of certifying synthesis. The constraint system $\mathcal{C}_{A,\varphi,\mathcal{B}}$ is presented in [13].

Note that we build a *single* constraint system for the whole certifying synthesis task. That is, the strategies and certificates of the individual processes are not synthesized completely independently. This is one of the main differences of our approach to the negotiation-based assume-guarantee synthesis algorithm [19]. While this prevents separate synthesis tasks and thus parallelizability, it eliminates the need for a negotiation between the processes. Moreover, it allows for completeness of the synthesis algorithm. Although the synthesis tasks are not fully separated, the constraint system $\mathcal{C}_{A,\varphi,\mathcal{B}}$ is in most cases still significantly smaller and easier to solve than the one of classical distributed synthesis.

As indicated in Theorem 4, certifying synthesis with local strategies is not complete in general: We can only ensure completeness if the satisfaction of each subspecification solely depends on the variables that the corresponding process can observe. This incompleteness is due to a slight difference in the satisfaction of a specification with local strategies and local satisfaction with complete strategies: The latter requires a strategy s_i to satisfy φ_i if *all processes* stick to their guarantees. The former, in contrast, requires s_i to satisfy φ_i if *all processes producing observable outputs* stick to their guarantees. Hence, if p_i cannot observe whether p_j sticks to its guarantee, satisfaction with local strategies requires s_i to satisfy φ_i even if p_j deviates, while local satisfaction does not.

This slight change in definition is needed in order to incorporate the requirements of certifying synthesis into the transition system representing the strategy and thus to be able to reuse existing bounded synthesis frameworks. Although this advantage is at general completenesses expanse, we experienced that in practice many distributed systems, at least after rewriting the specification, indeed satisfy the condition that is needed for completeness in our approximation of certifying synthesis. In fact, all benchmarks described in Section 8 satisfy it.

8 Experimental Results

We have implemented certifying synthesis with local strategies and guarantee transition systems. It expects an LTL formula and its decomposition as well as the system architecture, and bounds on the sizes of the strategies and certificates as input. Specification decomposition can easily be automated by, e.g., implementing Definition 1. The implementation extends the synthesis tool BoSy [9] for monolithic systems to certifying synthesis for distributed systems. In particular, we extend and adapt BoSy's SAT encoding [8] as described in [13].

We compare our implementation to two extensions of BoSy: One for distributed systems [2] and one for synthesizing individual dominant strategies, implementing the compositional synthesis algorithm presented in [7]. The results are shown in Table 1. We used the SMT encoding of distributed BoSy since the other ones either cause memory errors on almost all benchmarks (SAT), or do not support most of our architectures (QBF). Since the running times of the underlying SMT solver vary immensely, we report on the average running time of 10 runs. Synthesizing individual dominant strategies is incomplete and hence we can only report on results for half of our benchmarks. We could not compare our implementation to the iterative assume-guarantee synthesis tool Agnes [19], since it currently does not support most of our architectures or specifications.

The first four benchmarks stem from the synthesis competition [15]. The latch is parameterized in the number of bits, the generalized buffer in the number of senders, the load balancer in the number of servers, and the shift benchmark in the number of inputs. The fourth benchmark is a ripple-carry adder that is parameterized in the number of bits and the last benchmark describes the manufacturing robots from Section 2 and is parameterized in the size of the objectives φ_{add_i} of the robots. The system architectures are given in [13].

Table 1: Experimental results on scalable benchmarks. Reported is the parameter and the running time in seconds. We used a machine with a 3.1 GHz Dual-Core Intel Core i5 processor and 16 GB of RAM, and a timeout of 60min. For dist. BoSy, we use the SMT encoding and give the average runtime of 10 runs.

Benchmark	Param.	Cert. Synth.	Dist. BoSy	Dom. Strat.
n-ary Latch	2	**0.89**	41.26	4.75
	3	**0.91**	TO	6.40
	...	...	...	...
	6	**12.26**	TO	13.89
	7	105.69	TO	**15.06**
Generalized Buffer	1	**1.20**	6.59	5.23
	2	**2.72**	3012.51	10.53
	3	**122.09**	TO	961.60
Load Balancer	1	**0.98**	1.89	2.18
	2	**1.64**	2.39	–
Shift	2	**1.10**	1.99	4.76
	3	**1.13**	4.16	7.04
	4	**1.14**	TO	11.13
	...	...	...	...
	7	**9.01**	TO	16.08
	8	71.89	TO	**19.38**
Ripple-Carry Adder	1	**0.878**	1.83	–
	2	**2.09**	36.84	–
	3	**106.45**	TO	–
Manufacturing Robots	2	**1.10**	2.45	–
	4	**1.18**	2.43	–
	6	**1.67**	3.20	–
	8	**2.88**	5.67	–
	10	**48.83**	221.16	–
	12	**1.44**	TO	–
	...	...	...	...
	42	**373.90**	TO	–

For the latch, the generalized buffer, the ripple-carry adder, and the shift, certifying synthesis clearly outperforms distributed BoSy. For many parameters, BoSy does not terminate within 60min, while certifying synthesis solves the tasks in less than 13s. For these benchmarks, a process does not need to know the full behavior of the other processes. Hence, the certificates are notably smaller than the strategies. A process of the ripple-carry adder, for instance, only needs information about the carry bit of the previous process, the sum bit is irrelevant.

For the load balancer, in contrast, the certificates need to contain the full behavior of the processes. Hence, the benefit of the compositional approach lies solely in the specification decomposition. This advantage suffices to produce a solution faster than distributed BoSy. Yet, for other benchmarks with full certificates, the overhead of synthesizing certificates dominates the benefit of specifi-

cation decomposition for larger parameters, showcasing that certifying synthesis is particularly beneficial if a small interface between the processes exists.

The manufacturing robot benchmark is designed such that the interface between the processes stays small for all parameters. Hence, it demonstrates the advantage of abstracting from irrelevant behavior. Certifying synthesis clearly outperforms distributed BoSy on all instances. The parameter corresponds to the minimal solution size with distributed BoSy which does not directly correspond to the solution size with certifying synthesis. Thus, the running times do not grow in parallel. For more details on this benchmark we refer to [13].

Thus, certifying synthesis is extremely beneficial for specifications where small certificates exist. This directly corresponds to the existence of a small interface between the processes of the system. Hence, bounding the size of the certificates indeed guides the synthesis procedure in finding solutions fast.

When synthesizing dominant strategies, the weaker winning condition poses implicit assumptions on the behavior of the other processes. These assumptions do not always suffice: There are no independent dominant strategies for the load balancer, the ripple-carry adder, and the robots. While certifying synthesis performs better for the generalized buffer, the slight overhead of synthesizing explicit certificates becomes clear for the latch and the shift: For larger parameters, synthesizing dominant strategies outperforms certifying synthesis. Yet, the implicit assumptions do not encapsulate the required interface between the processes and thus they do not increase the understandability of the system's interconnections.

9 Conclusions

We have presented a synthesis algorithm that reduces the complexity of distributed synthesis by decomposing the synthesis task into smaller ones for the individual processes. To ensure completeness, the algorithm synthesizes additional certificates that capture a certain behavior a process commits to. A process then makes use of the certificates of the other processes by only requiring its strategy to satisfy the specification if the other processes do not deviate from their certificates. Synthesizing additional certificates increases the understandability of the system and the solution since the certificates capture the interconnections of the processes and which agreements they have to establish. Moreover, the certificates form a contract between the processes: The synthesized strategies can be substituted as long as the new strategy still complies with the contract, i.e., as long as it does not deviate from the guaranteed behavior, enabling modularity.

We have introduced two representations of the certificates, as LTL formulas and as labeled transition systems. Both ensure soundness and completeness of the compositional certifying synthesis algorithm. For the latter representation, we presented an encoding of the search for strategies and certificates into a SAT constraint solving problem. Moreover, we have introduced a technique for reducing the number of certificates that a process needs to consider by determining relevant processes. We have implemented the certifying synthesis algorithm and compared it to two extensions of the synthesis tool BoSy to distributed systems.

The results clearly show the advantage of compositional approaches as well as of guiding the synthesis procedure by bounding the size of the certificates: For benchmarks where small interfaces between the processes exist, certifying synthesis outperforms the other distributed synthesis tools significantly. If no solution with small interfaces exist, the overhead of certifying synthesis is small.

References

1. Alur, R., Moarref, S., Topcu, U.: Pattern-Based Refinement of Assume-Guarantee Specifications in Reactive Synthesis. In: TACAS (2015)
2. Baumeister, J.E.: Encodings of Bounded Synthesis for Distributed Systems. Bachelor's Thesis, Saarland University (2017)
3. Bloem, R., Chatterjee, K., Jacobs, S., Könighofer, R.: Assume-Guarantee Synthesis for Concurrent Reactive Programs with Partial Information. In: TACAS (2015)
4. Brenguier, R., Raskin, J., Sankur, O.: Assume-Admissible Synthesis. Acta Informatica (2017)
5. Chatterjee, K., Henzinger, T.A.: Assume-Guarantee Synthesis. In: TACAS (2007)
6. Damm, W., Finkbeiner, B.: Does It Pay to Extend the Perimeter of a World Model? In: FM (2011)
7. Damm, W., Finkbeiner, B.: Automatic Compositional Synthesis of Distributed Systems. In: FM (2014)
8. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of Bounded Synthesis. In: TACAS (2017)
9. Faymonville, P., Finkbeiner, B., Tentrup, L.: BoSy: An Experimentation Framework for Bounded Synthesis. In: CAV (2017)
10. Filiot, E., Jin, N., Raskin, J.: Compositional Algorithms for LTL Synthesis. In: ATVA (2010)
11. Finkbeiner, B., Geier, G., Passing, N.: Specification Decomposition for Reactive Synthesis. In: NFM (2021)
12. Finkbeiner, B., Passing, N.: Dependency-Based Compositional Synthesis. In: ATVA (2020)
13. Finkbeiner, B., Passing, N.: Compositional Synthesis of Modular Systems (Full Version). CoRR **abs/2106.14783** (2021)
14. Finkbeiner, B., Schewe, S.: Bounded Synthesis. STTT (2013)
15. Jacobs, S., Bloem, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, P.J., Michaud, T., Sakr, M., Sickert, S., Tentrup, L., Walker, A.: The 5th Reactive Synthesis Competition (SYNTCOMP 2018): Benchmarks, Participants & Results. CoRR **abs/1904.07736** (2019)
16. Kugler, H., Segall, I.: Compositional Synthesis of Reactive Systems from Live Sequence Chart Specifications. In: TACAS (2009)
17. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless Compositional Synthesis. In: CAV (2006)
18. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS (2005)
19. Majumdar, R., Mallik, K., Schmuck, A., Zufferey, D.: Assume-Guarantee Distributed Synthesis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (2020)
20. Pnueli, A.: The Temporal Logic of Programs. In: FOCS (1977)
21. de Roever, W.P., Langmaack, H., Pnueli, A. (eds.): Compositionality: The Significant Difference, COMPOS (1998)
22. Safra, S.: On the Complexity of omega-Automata. In: FOCS (1988)
23. Touati, H.J., Brayton, R.K., Kurshan, R.P.: Testing Language Containment for omega-Automata Using BDD's. Inf. Comput. (1995)