Pattern-Based Refinement of Assume-Guarantee Specifications in Reactive Synthesis[⋆]

Rajeev Alur, Salar Moarref, and Ufuk Topcu

University of Pennsylvania, Philadelphia, USA
{alur,moarref,utopcu}@seas.upenn.edu

Abstract. We consider the problem of compositional refinement of components' specifications in the context of compositional reactive synthesis. Our solution is based on automatic refinement of assumptions and guarantees expressed in linear temporal logic (LTL). We show how behaviors of the environment and the system can be inferred from counter-strategies and strategies, respectively, as formulas in special forms called patterns. Instantiations of patterns are LTL formulas which hold over all runs of such strategies, and are used to refine the specification by adding new input assumptions or output guarantees. We propose three different approaches for compositional refinement of specifications, based on how much information is shared between the components, and demonstrate and compare the methods empirically.

1 Introduction

Given a specification in a formal language such as linear temporal logic (LTL), reactive synthesis problem is to find a finite-state system that satisfies the specification, no matter how its environment behaves. The synthesis problem can be viewed as a game between two players: the system and its environment. The system attempts to satisfy the specification, while its environment tries to violate it. The specification is realizable, if there is a system that can satisfy it. Otherwise, a counter-strategy can be computed for the environment which describes the way it can behave so that no system can satisfy the specification.

The reactive synthesis problem is known to be intractable for general LTL specifications [1]. However, there are fragments of LTL, such as Generalized Reactivity(1) (GR(1)), for which the realizability and synthesis problems can be solved in polynomial time in the number of states of the reactive system [2]. Yet scalability is a big challenge as increasing the number of formulas in a specification may cause an exponential blowup in the size of its state space [2]. Compositional synthesis techniques can potentially address this issue by solving the synthesis problem for smaller components and merging the results such that the composition satisfies the specification. The challenge is then to find proper

⋆ This research was partially supported by NSF Expedition in Computing project ExCAPE (grant CCF 1138996) and AFOSR (grant number FA9550-12-1-0302).

decompositions and assumptions-guarantees such that each component is realizable, its expectations of its environment can be discharged on the environment and other components, and circular reasoning is avoided, so that the local controllers can be implemented simultaneously and their composition satisfies the original specification [3].

We study the problem of compositional refinement of components' specifications in the context of compositional reactive synthesis. We consider the special case in which the system consists of two components C_1 and C_2 and that a global specification is given which is realizable and decomposed into two local specifications, corresponding to C_1 and C_2, respectively. Furthermore, we assume that there is a serial interconnection between the components [3], i.e., only the output variables of C_2 depend on those of C_1. We are interested in computing refinements such that the refined local specifications are both realizable and when implemented, the resulting system satisfies the global specification.

Our solution is based on automated refinement of assumptions and guarantees expressed in LTL. In [4] we showed how an unrealizable specification can be refined by adding assumptions on its environment. The core of the method is the synthesis of a set of LTL formulas of special form, called patterns, which hold over all runs of an abstraction of the counter-strategy computed for the unrealizable specification. If the local specification for a component C_2 is unrealizable, we refine its environment assumptions, while ensuring that the other component C_1 can indeed guarantee those assumptions. To this end, it is sometimes necessary to refine C_1's specification by adding guarantees to it. We extend the methods in [4] to be able to refine guarantees as well as assumptions.

The main contributions of the paper are as follow. We extend our work in [4] in several aspects. We improve the scalability of the methods proposed in [4] by showing how a more compact abstraction can be constructed for counter-strategies and strategies. We extend the forms of patterns that can be synthesized, and show how a similar technique for refining unrealizable specifications can be used to refine the requirements of the system. We propose three different approaches that can be used to refine the specifications of the components in the context of compositional synthesis. Intuitively, these approaches differ in how much information about one component is shared with the other one. We show that providing more knowledge of one component's behavior for the other component can make it significantly easier to refine the local specifications, with the expense of increasing the coupling between the components. We illustrate and compare the methods with examples and a case study.

Related Work. The problem of refining the environment assumptions is also considered in [5,6]. Synthesizing distributed systems from global specification is a hard problem [7]. However, distributed controller synthesis algorithms exists for special architectures [8]. Assume-guarantee synthesis problem is considered in [9] and solved by computing secure-equilibrium strategies. We use a different approach for refining the specifications which is based on strategies and counter-strategies.

2 Preliminaries

Let P be the set of atomic propositions (Boolean variables) partitioned into input I and output O propositions. Linear temporal logic (LTL) is a formal specification language with two types of operators: logical connectives ($\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and $\rightarrow$ (implication)) and temporal operators (e.g., $\bigcirc$ (next), $\Diamond$ (eventually), and $\Box$ (always)). An LTL formula is interpreted over infinite words $w \in (2^P)^\omega$. The language of an LTL formula ϕ, denoted by $\mathcal{L}(\phi)$, is the set of infinite words that satisfy ϕ, i.e., $\mathcal{L}(\phi) = \{w \in (2^P)^\omega \mid w \models \phi\}$. We assume some familiarity of the reader with LTL. In this paper, We consider GR(1) specifications which are of the form $\phi = \phi_e \rightarrow \phi_s$, where ϕ_α for $\alpha \in \{e, s\}$ can be written as a conjunction of the following parts:

- ϕ_i^α: A Boolean formula over I if $\alpha = e$ and over $I \cup O$ otherwise, characterizing the initial state.
- ϕ_g^α: A formula of the form $\bigwedge_i \Box\Diamond B_i$ characterizing fairness/liveness, where each B_i is a Boolean formula over $I \cup O$.
- ϕ_t^α: An LTL formula of the form $\bigwedge_i \Box\psi_i$ characterizing safety and transition relations, where ψ_i is a Boolean formula over expressions v and $\bigcirc v'$ where $v \in I \cup O$ and, $v' \in I$ if $\alpha = e$ and $v' \in I \cup O$ if $\alpha = s$.

Intuitively, ϕ_e indicates the assumptions on the environment and ϕ_s characterizes the requirements of the system. Any correct implementation that satisfies the specification guarantees to satisfy ϕ_s, provided that the environment satisfies ϕ_e.

A labeled transition system (LTS) is a tuple $\mathcal{T} = \langle Q, Q_0, \delta, \mathcal{L}\rangle$ where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta \subseteq Q \times Q$ is a transition relation, and $\mathcal{L} : Q \rightarrow \phi$ is a labeling function which maps each state to a propositional formula $\phi = \bigwedge_i l_{p_i}$ expressed as a conjunction of literals l_{p_i} over propositions $p_i \in P$. The projection of a label ϕ with respect to a set of variables $U \subseteq P$ is defined as the propositional formula $\phi_{\downarrow U}$ where any literal ℓ_{p_i} over $p_i \in P \backslash U$ in ϕ is replaced by `True`, i.e., $\phi_{\downarrow U}$ only contains the variables from U. A *run* of an LTS is an infinite sequence of states $\sigma = q_0 q_1 q_2 ...$ where $q_0 \in Q_0$ and for any $i \geq 0$, $q_i \in Q$ and $(q_i, q_{i+1}) \in \delta$. The language of an LTS $\mathcal{T}$ is defined as the set $\mathcal{L}(\mathcal{T}) = \{w \in Q^\omega \mid w \text{ is a run of } \mathcal{T}\}$, i.e., the set of (infinite) words generated by the runs of $\mathcal{T}$.

A *Moore transducer* is a tuple $M = (S, s_0, \mathcal{I}, \mathcal{O}, \delta, \gamma)$, where S is a set of states, $s_0 \in S$ is an initial state, $\mathcal{I} = 2^I$ is the input alphabet, $\mathcal{O} = 2^O$ is the output alphabet, $\delta : S \times \mathcal{I} \rightarrow S$ is a transition function and $\gamma : S \rightarrow \mathcal{O}$ is a state output function. A *Mealy* transducer is similar, except that the state output function is $\gamma : S \times \mathcal{I} \rightarrow \mathcal{O}$. For an infinite word $w \in \mathcal{I}^\omega$, a run of M is an infinite sequence $\sigma \in S^\omega$ such that $\sigma_0 = s_0$ and for all $i \geq 0$ we have $\sigma_{i+1} = \delta(\sigma_i, \omega_i)$. The run σ on input word w produces an infinite word $M(w) \in (2^P)^\omega$ such that $M(w)_i = \gamma(\sigma_i) \cup w_i$ for all $i \geq 0$. The language of M is the set $\mathcal{L}(M) = \{M(w) \mid w \in \mathcal{I}^\omega\}$ of infinite words generated by runs of M.

An LTL formula ϕ is *satisfiable* if there exists an infinite word $w \in (2^P)^\omega$ such that $w \models \phi$. A Moore (Mealy) transducer M satisfies an LTL formula ϕ, written as $M \models \phi$, if $\mathcal{L}(M) \subseteq \mathcal{L}(\phi)$. An LTL formula ϕ is *Moore (Mealy) realizable* if there exists a Moore (Mealy, respectively) transducer M such that $M \models \phi$. The *realizability problem* asks whether there exists such a transducer for a given ϕ. Given an LTL formula ϕ over P and a partitioning of P into I and O, the *synthesis problem* is to find a Mealy transducer M with input alphabet $\mathcal{I} = 2^I$ and output alphabet $\mathcal{O} = 2^O$ that satisfies ϕ. A *counter-strategy* for the synthesis problem is a strategy for the environment that can falsify the specification, no matter how the system plays. Formally, a counter-strategy can be represented by a Moore transducer $M_c = (S', s'_0, \mathcal{I}', \mathcal{O}', \delta', \gamma')$ that satisfies $\neg\phi$, where $\mathcal{I}' = \mathcal{O}$ and $\mathcal{O}' = \mathcal{I}$ are the input and output alphabet for M_c which are generated by the system and the environment, respectively.

For a specification $\phi = \phi_e \rightarrow \phi_s$, we define an *assumption refinement* $\psi_e = \bigwedge_i \psi_{e_i}$ as a conjunction of a set of environment assumptions such that $(\phi_e \wedge \psi_e) \rightarrow \phi_s$ is realizable. Similarly, $\psi_s = \bigwedge_i \psi_{s_i}$ is a *guarantee refinement* if $\phi_e \rightarrow (\phi_s \wedge \psi_s)$ is realizable. An assumption refinement ψ_e is consistent with ϕ if $\phi_e \wedge \psi_e$ is satisfiable. Note that an inconsistent refinement $\phi_e \wedge \psi_e = \texttt{False}$, leads to an specification which is trivially realizable, but neither interesting nor useful.

3 Overview

Assume a global LTL specification is given which is realizable. Furthermore, assume the system consists of a set of components, and that a decomposition of the global specification into a set of local ones is given, where each local specification corresponds to a system component. The decomposition may result in components whose local specifications are unrealizable, e.g., due to the lack of adequate assumptions on their environment. The general question is how to refine the local specifications such that the refined specifications are all realizable, and when implemented together, the resulting system satisfies the global specification. In this paper we consider a special case of this problem. We assume the system consists of two components C_1 and C_2, where there is a serial interconnection between the components [3]. Intuitively, it means that the dependency between the output variables of the components is acyclic, as shown in Fig. 2. Let I be the set of input variables controlled by the environment and O be the set of output variables controlled by the system, partitioned into O_1 and O_2, the set of output variables controlled by C_1 and C_2, respectively. Formally, we define the problem as follows.

Problem Statement. Consider a realizable global specification $\phi = \phi_e \rightarrow \phi_s$. We assume ϕ is decomposed into two local specifications $\phi_1 = \phi_{e_1} \rightarrow \phi_{s_1}$ and $\phi_2 = \phi_{e_2} \rightarrow \phi_{s_2}$ such that $\phi_e \rightarrow (\phi_{e_1} \wedge \phi_{e_2})$ and $(\phi_{s_1} \wedge \phi_{s_2}) \rightarrow \phi_s$. We assume ϕ_e, ϕ_s, ϕ_{e_1}, ϕ_{s_1}, ϕ_{e_2}, and ϕ_{s_2} are GR(1) formulas which contain variables only from the sets I, $I \cup O$, I, $I \cup O_1$, $I \cup O_1$, and $I \cup O$, respectively. We would like to find refinements ψ and ψ' such that the refined specifications $\phi_1^{ref} = \phi_{e_1} \rightarrow (\phi_{s_1} \wedge \psi')$ and $\phi_2^{ref} = (\phi_{e_2} \wedge \psi) \rightarrow \phi_{s_2}$ are both realizable and $\psi' \rightarrow \psi$ is valid.

From Proposition 2 in [3] it follows that if such refinements exist, then the resulting system from implementing the refined specifications satisfies the global specification ϕ. We use this fact to establish the correctness of the decomposition and refinements in our proposed solutions. As ϕ is realizable, and C_1 is independent from C_2, it follows that ϕ_1 (in case it is not realizable) can become realizable by adding assumptions on its environment. Especially, providing all the environment assumptions of the global specification for C_1 is enough to make its specification realizable. However, it might not be the case for ϕ_2. In the rest of the paper, we assume that ϕ_1 is realizable, while ϕ_2 is not. We investigate how the strategy and counter-strategy computed for C_1 and C_2, respectively, can be used to find suitable refinements for the local specifications.

Our solution is based on an automated refinement of assumptions and guarantees expressed in LTL. In [4], we showed how an unrealizable specification can be refined by adding assumptions on the environment. The refinement is synthesized step by step guided by counter-strategies. When the specification is unrealizable, a counter-strategy is computed and a set of formulas of the forms $\Diamond\Box\psi$, $\Diamond\psi$, and $\Diamond(\psi \wedge \bigcirc\psi')$, which hold over *all* runs of the counter-strategy, is inferred. Intuitively, these formulas describe potentially "bad" behaviors of the environment that may cause unrealizability. Their complements form the set of *candidate* assumptions, and adding any of them as an assumption to the specification will prevent the environment from behaving according to the counter-strategy (without violating its assumptions). We say the counter-strategy is ruled out from the environment's possible behaviors. Counter-strategy-guided refinement algorithm in [4] iteratively chooses and adds a candidate assumption to the specification, and the process is repeated until the specification becomes realizable, or the search cannot find a refinement within the specified search depth. The user is asked to specify a subset of variables to be used in synthesizing candidate assumptions. This subset may reflect the designer's intuition on the source of unrealizability and help search for finding a proper refinement.

In this paper, we extend the algorithms in [4] to refine the guarantees of a specification. When the specification is realizable, a winning strategy can be computed for the system. We can use patterns to infer the behaviors of the strategies as LTL formulas. Formulas of the form $\Box\Diamond\psi$, $\Box\psi$, and $\Box(\psi \rightarrow \bigcirc\psi')$ can be used to infer *implicit* guarantees provided by the given strategy, i.e., they can be added to the original specification as guarantees, and the same strategy satisfies the new specification as well as the original one. These formulas can be seen as additional guarantees a component can provide in the context of compositional synthesis. Formulas of the form $\Diamond\Box\psi$, $\Diamond\psi$, and $\Diamond(\psi \wedge \bigcirc\psi')$ can be used to restrict the system by adding their complements to the specifications as guarantees. As a result, the current strategy is ruled out from system's possible strategies and hence, the new specification, if still realizable, will have a different strategy which satisfies the original specification, and also provides additional guarantees. Algorithm 1 shows how a set of additional guarantees $\mathcal{P}$ is computed for the specification ϕ and subset of variables U. For the computed strategy $\mathcal{M}_s$, the procedure **Infer-GR(1)-Formulas** synthesizes formulas of the forms $\Box\Diamond\psi, \Box\psi$, and $\Box(\psi \rightarrow \bigcirc\psi')$ which hold over all runs of the strategy. Similarly, the procedure **Infer-Complement-GR(1)-Formulas** synthesizes formulas of

```
Algorithm 1: FindGuarantees
   Input: $\phi = \phi_e \rightarrow \phi_s$: a realizable specification, $U$: subset of variables
   Output: $\mathcal{P}$: A set of formulas $\psi$ such that $\phi_e \rightarrow (\phi_s \wedge \psi)$ is realizable
 1 $\mathcal{M}_s$ = ComputeStrategy($\phi_1$);
 2 $\mathcal{P}$ := Infer-GR(1)-Formulas($\mathcal{M}_s, U$);
 3 $\mathcal{P}'$ := Infer-Complement-GR(1)-Formulas($\mathcal{M}_s, U$);
 4 foreach $\psi \in \mathcal{P}'$ do
 5     if ($\phi_e \rightarrow (\phi_s \wedge \neg\psi)$) is realizable then
 6         $\mathcal{P} = \mathcal{P} \cup \neg\psi$ ;
 7 return $\mathcal{P}$ ;
```

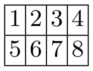

Fig. 1: Room in Ex. 1

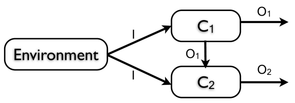

Fig. 2: Serial interconnection.

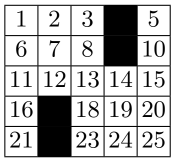

Fig. 3: Grid-world for the case study

the form $\Diamond\Box\psi$, $\Diamond\psi$, and $\Diamond(\psi \wedge \bigcirc\psi')$. These procedures are explained in Sect. 4. In what follows, we will use *grid-world* examples commonly used in robot motion planning case studies to illustrate the concepts and techniques [10].

Example 1. Assume there are two robots, R_1 and R_2, in a room divided into eight cells as shown in Fig. 1. Both robots must infinitely often visit the goal cell 4. Besides, they cannot be in the same cell simultaneously (no collision). Finally, at any time step, each robot can either stay put or move to one of its neighbor cells. In the sequel, assume i ranges over $\{1, 2\}$. We denote the location of robot R_i with Loc_{R_i}, and cells by their numbers. Initially $Loc_{R_1} = 1$ and $Loc_{R_2} = 8$.

The global specification is realizable. Note that in this example, all the variables are controlled and there is no external environment. Assume that the specification is decomposed into ϕ_1 and ϕ_2, where $\phi_i = \phi_{e_i} \rightarrow \phi_{s_i}$ is the local specification for R_i. Assume $\phi_{e_1} = \texttt{True}$, and ϕ_{s_1} only includes the initial location of R_1, its transition rules, and its goal to infinitely often visit cell 4. ϕ_{s_2} includes the initial location of R_2, its transition rules, its objective to infinitely often visit cell 4, while avoiding collision with R_1. Here R_1 serves as the environment for R_2 which can play adversarially. ϕ_{e_2} only includes the initial location of R_1.

Inferring formulas: ϕ_1 is realizable. A winning strategy $\mathcal{M}_{S_1}$ for R_1 is to move to cell 2 from the initial location, then to cell 3, and then to move back and forth between cells 4 and 3 forever. The following are examples of formulas inferred from this strategy: eventually always: $\Diamond\Box(Loc_{R_1} \in \{3, 4\})$, eventually: $\Diamond(Loc_{R_1} = 3)$, eventually next: $\Diamond(Loc_{R_1} = 3 \wedge \bigcirc Loc_{R_1} = 4)$, always eventually: $\Box\Diamond(Loc_{R_1} = 3)$, always: $\Box(Loc_{R_1} \in \{1, 2, 3, 4\})$, and always next: $\Box(Loc_{R_1} = 2 \rightarrow \bigcirc Loc_{R_1} = 3)$.

Refining assumptions: Note that ϕ_2 includes no assumption on R_1 other than its initial location. Specifically, ϕ_2 does not restrict the way R_1 can move. The specification ϕ_2 is unrealizable. A counter-strategy for R_1 is to move from cell 1 to the goal cell 4, and stay there forever, preventing R_2 from fulfilling its requirements. Using the method of [4] for refining the assumptions on the environment, we find the refinements $\psi_1 = \Box\Diamond(Loc_{R_1} \neq 4)$, $\psi_2 = \Box(Loc_{R_1} \neq 4)$, and $\psi_3 = \Box(Loc_{R_1} = 4 \rightarrow \bigcirc Loc_{R_1} \neq 4)$. Intuitively, these refinements suggest that R_1 is not present in cell 4 at some point during the execution. Adding any of these formulas to the assumptions of ϕ_2 makes it realizable.

Refining guarantees: Formula $\varphi = \Diamond\Box(Loc_{R_1} \in \{3, 4\})$ is satisfied by $\mathcal{M}_{S_1}$, meaning that R_1 eventually reaches and stays at the cells 3 and 4 forever. An example of a guarantee refinement is to add the guarantee $\neg\varphi$ to ϕ_1. A winning strategy for the new specification is to move back and forth in the first row between initial and goal cells. That is, R_1 has the infinite run $(1, 2, 3, 4, 3, 2)^\omega$.

We use these techniques to refine the interface specifications. We propose three different approaches for finding suitable refinements, based on how much information about the strategy of the realizable component is allowed to be shared with the unrealizable component. The first approach has no knowledge of the strategy chosen by C_1, and tries to find a refinement by analyzing counter-strategies. The second approach iteratively extracts some information from the strategies computed for ϕ_1, and uses them to refine the specifications. The third approach encodes the strategy as a conjunction of LTL formulas, and provides it as a set of assumptions for C_2, allowing it to have a full knowledge of the strategy. These approaches are explained in detail in Sect. 5.

Compositional Refinement: Assume $\mathcal{M}_{S_1}$ is the computed strategy for R_1. The first approach, computes a refinement for the unrealizable specification, then checks if the other component can guarantee it. For example, ψ_3 is a candidate refinement for ϕ_2. ϕ_1 can be refined by ψ_3 added to its guarantees. The strategy $\mathcal{M}_{S_1}$ still satisfies the new specification, and refined specifications are both realizable. Thus, the first approach returns ψ_3 as a possible refinement. Using the second approach, formula $\psi_4 = \Box\Diamond(Loc_{R_1} = 3)$ is inferred from $\mathcal{M}_{S_1}$. Refining both specifications with ψ_4 leads to two realizable specifications, hence ψ_4 is returned as a refinement. The third approach encodes $\mathcal{M}_{S_1}$ as conjunction of transition formulas $\psi_5 = \bigwedge_{i=1}^{3} \Box(Loc_{R_1} = i \rightarrow \bigcirc Loc_{R_1} = i + 1) \wedge \Box(Loc_{R_1} = 4 \rightarrow \bigcirc Loc_{R_1} = 3)$. Refining assumptions of ϕ_2 with ψ_5 makes it realizable.

4 Inferring Behaviors as LTL Formulas

In this section we show how certain types of LTL formulas which hold over all runs of a counter-strategy or strategy can be synthesized. The user chooses the subset of variables U to be used in synthesizing the formulas. These formulas are computed as follows: First an LTS $\mathcal{T}$ is obtained from the given Moore (Mealy) transducer $\mathcal{M}$ which represents the counterstrategy (strategy, respectively). Next, using the set U, an abstraction $\mathcal{T}^a$ of $\mathcal{T}$ is constructed which is

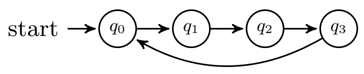

Fig. 4: An LTS $\mathcal{T}$

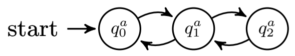

Fig. 5: Abstract LTS $\mathcal{T}^a$ of $\mathcal{T}$

also an LTS. A set of patterns which hold over all runs of $\mathcal{T}^a$ is then synthesized. The instantiations of these patterns form the set of formulas which hold over all runs of the input transducer. Next we explain these steps in more detail.

4.1 Constructing the Abstract LTS

We briefly show how an abstraction of a given strategy or counter-strategy is obtained as an LTS. Given a Moore (Mealy) transducer $\mathcal{M}$, first an LTS $\mathcal{T} = (Q, \{q_0\}, \delta_{\mathcal{T}}, \mathcal{L})$ is obtained which keeps the structure of $\mathcal{M}$ while removing its input and output details. The states of $\mathcal{T}$ are labeled in a way that is consistent with the input/output valuations of $\mathcal{M}$. Next, using a user-specified subset of variables $U \subseteq I \cup O$, an abstraction $\mathcal{T}^a = (Q^a, Q_0^a, \delta_{\mathcal{T}^a}, \mathcal{L}^a)$ of $\mathcal{T}$ is computed based on the state labels $\mathcal{L}$. There is a surjective function $F : Q \to Q^a$ which maps each state of $\mathcal{T}$ to a unique state of $\mathcal{T}^a$. Intuitively, the abstraction $\mathcal{T}^a$ has a unique state for each maximal subset of states of $\mathcal{T}$ which have the same projected labels with respect to U, and if there is a transition between two states of $\mathcal{T}$, there will be a transition between their mapped states in $\mathcal{T}^a$. It can be shown that $\mathcal{T}^a$ simulates $\mathcal{T}$. Therefore, any formula φ which is satisfied by $\mathcal{T}^a$ is also satisfied by $\mathcal{T}$.

Remark 1. Patterns can be synthesized from either $\mathcal{T}$ or $\mathcal{T}^a$. It is sometimes necessary to use $\mathcal{T}^a$ due to the high complexity of the algorithms for computing certain types of patterns (e.g., eventually patterns), as $\mathcal{T}^a$ may have significantly less number of states compared to $\mathcal{T}$ which improves the scalability of the methods. However, abstraction may introduce additional non-determinism into the model, leading to refinements which are more "conservative." Besides, some of the formulas which are satisfied by $\mathcal{T}$, cannot be computed from $\mathcal{T}^a$. It is up to the user to choose techniques which serve her purposes better.

4.2 Synthesizing Patterns

Next we discuss how patterns of certain types can be synthesized from the given LTS $\mathcal{T}$. A pattern $\psi_{\mathcal{P}}$ is an LTL formula which is satisfied over all runs of $\mathcal{T}$, i.e., $\mathcal{T} \models \psi_{\mathcal{P}}$. We are interested in patterns of the forms $\Diamond\Box\psi_{\mathcal{P}}$, $\Diamond\psi_{\mathcal{P}}$, $\Diamond(\psi_{\mathcal{P}} \wedge \bigcirc\psi'_{\mathcal{P}})$, $\Box\Diamond\psi_{\mathcal{P}}$, $\Box\psi_{\mathcal{P}}$, and $\Box(\psi_{\mathcal{P}} \to \bigcirc\psi'_{\mathcal{P}})$, where $\psi_{\mathcal{P}}$ and $\psi'_{\mathcal{P}}$ are propositional formulas expressed as a disjunction of subset of states of $\mathcal{T}$. Patterns are synthesized using graph search algorithms which search for special *configurations*. For an LTS $\mathcal{T} = (Q, Q_0, \delta, \mathcal{L})$, a configuration $C \subseteq Q$ is a subset of states of $\mathcal{T}$. A configuration C is a $\bowtie$-configuration where $\bowtie \in \{\Box, \Box\Diamond, \Diamond, \Diamond\Box\}$ if $\mathcal{T} \models \bowtie \bigvee_{q \in C} q$. For example, C is an $\Box\Diamond$-*configuration* if any run of $\mathcal{T}$ always eventually visits a state from C. A $\bowtie$-configuration C is minimal, if there is no configuration $C' \subset C$ which is an $\bowtie$-configuration, i.e., removing any state from C leads to a configuration which is not a $\bowtie$-configuration anymore. Minimal $\bowtie$-configurations are interesting since they lead to the *strongest* patterns of $\bowtie$-form [4]. Algorithms for computing $\Diamond\Box\psi_{\mathcal{P}}$, $\Diamond\psi_{\mathcal{P}}$, and $\Diamond(\psi_{\mathcal{P}} \wedge \bigcirc\psi'_{\mathcal{P}})$ patterns can be found in [4]. Here we give algorithms for computing patterns of the forms in GR(1).

$\Box\Diamond\psi_{\mathcal{P}}$ **Patterns**: The following theorem establishes the complexity of computing all minimal always eventually patterns over a given LTS.

Theorem 1. *Computing all minimal $\Box\Diamond$-configurations is NP-hard.*[1]

Consequently, computing all minimal (always) eventually patterns is infeasible in practice even for medium sized specifications. We propose an alternative algorithm which computes *some* of the always eventually patterns.[2] Although the algorithm has an exponential upper-bound, it is simpler and terminates faster in our experiments, as it avoids enumerating all configurations. It starts with the configuration $\{q_0\}$, and at each step computes the next configuration, i.e., the set of states that the runs of $\mathcal{T}$ can reach at the next step from the current configuration. A sequence $C_0, C_1, ..., C_j$ of configurations is discovered during the search, where $C_0 = \{q_0\}$ and $j \geq 0$. The procedure terminates when a configuration C_i is reached which is already visited, i.e., there exists $0 \leq j < i$ such that $C_j = C_i$. There is a cycle between C_j and C_{i-1} and thus, all the configurations in the cycle will always eventually be visited over all runs of $\mathcal{T}$.

$\Box\psi_{\mathcal{P}}$ **Pattern**: For a given LTS $\mathcal{T}$, a safety pattern of the form $\Box\psi$ is synthesized where ψ is simply the disjunction of all the states in $\mathcal{T}$, i.e., $\psi = \bigvee_{q \in Q} q$. It is easy to see that removing any state from ψ leads to a formula which is not satisfied by $\mathcal{T}$ anymore. The synthesis procedure is of complexity $O(|Q|)$.

$\Box(\psi_{\mathcal{P}} \rightarrow \bigcirc\psi'_{\mathcal{P}})$ **Patterns**: For a given LTS $\mathcal{T}$, a set of transition patterns of the form $\Box(\psi \rightarrow \bigcirc\psi')$ is synthesized. Each ψ consists of a single state $q \in Q$, for which the ψ' is disjunction of its successors, i.e. $\psi' = \bigvee_{q' \in Next(q)} q'$ where $Next(q) = \{q' \in Q \mid \delta(q) = q'\}$. Intuitively, each transition pattern states that always when a state is visited, its successors will be visited at the next step. The synthesis procedure is of complexity $O(|Q| + |\delta|)$.

4.3 Instantiating the Patterns

To obtain LTL formulas over a specified subset U of variables from patterns, we replace the states in patterns by their projected labels. For example, from an eventually pattern $\Diamond\psi_{\mathcal{P}} = \Diamond(\bigvee_{q \in Q_{\psi_{\mathcal{P}}}} q)$ where $Q_{\psi_{\mathcal{P}}} \subseteq Q$ is a configuration for $\mathcal{T} = (Q, \{q_0\}, \delta, \mathcal{L})$, we obtain the formula $\psi = \Diamond(\bigvee_{q \in Q_{\psi_{\mathcal{P}}}} \mathcal{L}(q)_{\downarrow U})$.

Example 2. Let $\Sigma = \{a, b, c\}$ be the set of variables. Consider the LTS $\mathcal{T}$ shown in Fig. 4, where $\mathcal{L}(q_0) = \neg a \wedge \neg b \wedge \neg c$, $\mathcal{L}(q_1) = \neg a \wedge b \wedge \neg c$, $\mathcal{L}(q_2) = a \wedge \neg b \wedge \neg c$, $\mathcal{L}(q_3) = \neg a \wedge b \wedge \neg c$. Let $U = \{a, b\}$ be the set of variables specified by the

[1] Computing all minimal eventually patterns is also NP-hard

[2] We use a similar algorithm for computing *some* of the eventually patterns.

designer to be used in all forms of formulas. Figure 5 shows $\mathcal{T}^a$ which is an abstraction of $\mathcal{T}$ with respect to U, where the mapping function F is defined such that $F^{-1}(q_0^a) = \{q_0\}$, $F^{-1}(q_1^a) = \{q_1, q_3\}$, and $F^{-1}(q_2^a) = \{q_2\}$, and the labels are defined as $\mathcal{L}(q_0^a) = \neg a \wedge \neg b$, $\mathcal{L}(q_1^a) = \neg a \wedge b$, and $\mathcal{L}(q_2^a) = a \wedge \neg b$. A set of patterns are synthesized using the input LTS. For example, $\psi_{\mathcal{P}} = \diamond(q_1^a)$ is an eventually pattern where $\mathcal{T}^a \models \psi_{\mathcal{P}}$, meaning that eventually over all runs of the $\mathcal{T}^a$ the state q_1^a is visited. An LTL formula is obtained using the patterns, labels and specified subset of variables. For example, $\psi = \diamond(\neg a \wedge b)$ is obtained from the pattern $\psi_{\mathcal{P}}$, where the states q_1^a is replaced by its label. Note that the formula $\psi' = \diamond((\neg a \wedge b) \wedge \bigcirc(a \wedge \neg b))$ can be synthesized from the pattern $\psi'_{\mathcal{P}} = \diamond(q_1 \wedge \bigcirc q_2)$ from $\mathcal{T}$, however, $\mathcal{T}^a$ does not satisfy ψ'. A more conservative formula $\diamond((\neg a \wedge b) \wedge \bigcirc((a \wedge \neg b) \vee (\neg a \wedge \neg b))$ is obtained using the abstraction.

5 Compositional Refinement

We propose three approaches for compositional refinement of the specifications ϕ_1 and ϕ_2 in the problem stated in Sect. 3. These approaches differ mainly in how much information about the strategy of the realizable component is shared with the unrealizable component. All three approaches use bounded search to compute the refinements. The search depth (number of times the refinement procedure can be called recursively) is specified by the user. Note that the proposed approaches are not complete, i.e., failure to compute a refinement does not mean that there is no refinement.

Approach 1 ("No knowledge of the strategy of C_1"): One way to synthesize the refinements ψ and ψ' is to compute a refinement ψ' for the unrealizable specification ϕ_2 using the counter-strategy-guided refinement method in [4]. The specification ϕ_2 is refined by adding assumptions on its environment that rule out all the counter-strategies for ϕ_2, as explained in Sect. 3, and the refined specification $\phi_2^{ref} = (\phi_{e_1} \wedge \psi') \rightarrow \phi_{s_1}$ is realizable. We add $\psi = \psi'$ to guarantees of ϕ_1 and check if ϕ_1^{ref} is realizable. If ϕ_1^{ref} is not realizable, another assumption refinement for ϕ_2 must be computed, and the process is repeated for the new refinement. Note that if adding ψ to the guarantees of ϕ_1 does not make it realizable, there is no ψ'' such that $\psi'' \rightarrow \psi$, and adding ψ'' keeps ϕ_1 realizable. Therefore, a new refinement must be computed.

An advantage of this approach is that the assumption refinement ψ' for ϕ_2 is computed independently using the weakest assumptions that rule out the counter-strategies. Thus, ψ' can be used even if C_1 is replaced by another component C_1' with different specification, as long as C_1' can still guarantee ψ'.

Approach 2 ("Partial knowledge of the strategy of C_1"): For a given counter-strategy, there may exist many different candidate assumptions that can be used to refine the specification. Checking the satisfiability and realizability of the resulting refined specification is an expensive process, so it is more desirable to remove the candidates that are not promising. For example, a counter-strategy might represent a problem which cannot happen due to the strategy chosen by the other component. Roughly speaking, the more one component knows about the other one's implementation, the less number of scenarios it needs to consider and react to. The second approach shares information about the strategy synthesized for C_1 with C_2 as follows. It computes a set $\mathcal{P}$ of candidate LTL formulas which can be used to refine guarantees of ϕ_1. Then at each iteration, a formula $\psi \in \mathcal{P}$ is chosen, and it is checked if the counter-strategy for ϕ_2 satisfies $\neg\psi$ (similar to assumption mining in [5]). If it does and ψ is consistent with ϕ_2, it is checked if ψ is an assumption refinement for ϕ_2, in which case ψ can be used to refine the guarantees (assumptions) of ϕ_1 (ϕ_2 , respectively), and ψ is returned as a suggested refinement. Otherwise, the local specifications are refined by ψ and the process is repeated with the new specifications. In this approach, some information about C_1's behavior is shared as LTL formulas extracted from the C_1's strategy. Only those formulas which completely rule out the counter-strategy are kept, hence reducing the number of candidate refinements, and keeping the more promising ones, while sharing as much information as needed from one component to the other one.

Approach 3 ("Full knowledge of the strategy of C_1") It might be preferred to refine the specification by adding formulas that are already satisfied by the current implementation of the realizable component in order not to change the underlying implementation. For example, assume a strategy $\mathcal{M}_S$ is already computed and implemented for ϕ_1, and the designer prefers to find a refinement ψ that is satisfied by $\mathcal{M}_S$. Yet in some cases, the existing strategy for C_1 must be changed, otherwise C_2 will not be able to fulfill its requirements. In this setting, the guarantees of C_1 can be refined to find a different winning strategy for it. The third approach is based on this idea. It shares the full knowledge of strategy computed for C_1 with C_2 by encoding the strategy as an LTL formula and providing it as an assumption for ϕ_2. Knowing exactly how C_1 plays might make it much easier for C_2 to synthesize a strategy for itself, if one exists. Furthermore, a counter-strategy produced in this case indicates that it is impossible for C_2 to fulfill its goals if C_1 sticks to its current strategy. Therefore, both specifications are refined and a new strategy is computed for the realizable component.

Algorithm 2 summarizes the third approach. Once a strategy is computed for the realizable specification, its corresponding LTS $\mathcal{T} = (Q, \{q_0\}, \delta, \mathcal{L})$ is obtained, and encoded as a conjunction of transition formulas as follows. We define a set of new propositions $\mathcal{Z} = \left\{z_0, z_1, \cdots, z_{\lceil log|Q|\rceil}\right\}$ which encode the states Q of $\mathcal{T}$. Intuitively, these propositions represent the memory of the strategy in generated transition formulas, and are considered as environment variables in the refined specification ϕ'_2. For ease of notation, let $|\mathcal{Z}|_i$ indicate the truth assignment to the propositions in $\mathcal{Z}$ which represents the state $q_i \in Q$. We encode $\mathcal{T}$ with the conjunctive formula $\psi = (|\mathcal{Z}|_0 \wedge \mathcal{L}(q_0) \wedge \bigwedge_{q_i \in Q} \Box((|\mathcal{Z}|_i \wedge \mathcal{L}(q_i)) \rightarrow \bigcirc(\bigvee_{q_j \in \text{Next}(q_i)} |\mathcal{Z}|_j \wedge \mathcal{L}(q_j))$, where $\text{Next}(q_i)$ is the set of states in $\mathcal{T}$ with a transition from q_i to them. We refer to ψ as *full encoding* of $\mathcal{T}$. Intuitively, ψ states that always when the strategy is in state $q_i \in Q$ with truth assignment to the variables given as $\mathcal{L}(q_i)$, then at next step it will be in one of the adjacent states $q_j \in \text{Next}(q_i)$ with truth assignment $\mathcal{L}(q_j)$ to the variables, and initially it is in state q_0. The procedure **Encode-LTS** in Alg. 2 takes an LTS and returns a conjunctive LTL formula representing it.

The unrealizable specification ϕ_2 is then refined by adding the encoding of the strategy as assumptions to it. If the refined specification ϕ_2' is realizable, there exists a strategy for C_2, assuming the strategy chosen for C_1, and the encoding is returned as a possible refinement. Otherwise, the produced counter-strategy $\mathcal{CS}'$ shows how the strategy for C_1 can prevent C_2 from realizing its specification. Hence, the specification of both components need to be refined. Procedure **findCandidateAssumptions** computes a set $\mathcal{P}$ of candidate assumptions that can rule out $\mathcal{CS}'$, and at each iteration, one candidate is chosen and tested by both specifications for satisfiability and realizability. If any of these candidate formulas can make both specifications realizable, it is returned as a refinement. Otherwise, the process is repeated with only those candidates that are consistent with ϕ_2, and keep ϕ_1 realizable. As a result, the set of candidate formulas is pruned and the process is repeated with the more promising formulas. If no refinement is found within the specified search depth, `False` is returned.

Remark 2. Introducing new propositions representing the memory of the strategy $\mathcal{S}_1$ computed for ϕ_1 leads to assumptions that provide C_2 with full knowledge of how C_1 reacts to its environment. Therefore, if the new specification refined by these assumptions is not realizable, the counter-strategy would be an example of how $\mathcal{S}_1$ might prevent ϕ_2 from being realizable, giving the designer the certainty that a different strategy must be computed for C_1, or in other words both specifications must be refined. However, if introducing new propositions is undesirable, an *abstract encoding* of the strategy (without memory variables) can be obtained by returning conjunction of *all* transition formulas $\square(\psi \rightarrow \bigcirc\psi')$ computed over the strategy. The user can specify the set of variables in which she is interested. This encoding represents an abstraction of the strategy that might be *non-deterministic*, i.e., for the given truth assignment to environment variables, there might be more than one truth assignment to outputs of C_1 that are consistent with the encoding. Such relaxed encoding can be viewed as sharing partial information about the strategy of C_1 with C_2.

As an example, consider the LTS $\mathcal{T}$ in Fig. 4 which can be encoded as $(q_0 \wedge \neg a \wedge \neg b \wedge \neg c) \wedge \square((q_0 \wedge \neg a \wedge \neg b \wedge \neg c) \rightarrow \bigcirc(q_1 \wedge \neg a \wedge b \wedge \neg c)) \wedge \cdots \wedge \square((q_3 \wedge \neg a \wedge b \wedge \neg c) \rightarrow \bigcirc(q_0 \wedge \neg a \wedge \neg b \wedge \neg c))$. An abstract encoding without introducing new variables and considering only a and b results in formula $\square((\neg a \wedge \neg b) \rightarrow \bigcirc(\neg a \wedge b)) \wedge \square((\neg a \wedge b) \rightarrow \bigcirc((\neg a \wedge \neg b) \vee (a \wedge \neg b)) \wedge \square((a \wedge \neg b) \rightarrow \bigcirc(\neg a \wedge b))$.

6 Case Study

We now demonstrate the techniques on a robot motion planning case study. We use RATSY [11] for computing counter-strategies, JTLV [12] for synthesizing strategies, and Cadence SMV model checker [13] for model checking. The experiments are performed on a Intel core i7 3.40 GHz machine with 16GB memory.

Consider the robot motion planning example over the discrete workspace shown in Fig. 3. Assume there are two robots R_1 and R_2 initially in cells 1 and 25, respectively. Robots can move to one of their neighbor cells at each step.

Algorithm 2: CompositonalRefinement3

```
Input: $\phi_1 = \phi_{e_1} \rightarrow \phi_{s_1}$: a realizable specification, $\phi_2 = \phi_{e_2} \rightarrow \phi_{s_2}$: an
        unrealizable specification, $\alpha$: search depth, $U$: subset of variables
Output: $\psi$ such that $\phi_{e_1} \rightarrow (\phi_{s_2} \wedge \psi)$ and $(\phi_{e_2} \wedge \psi) \rightarrow \phi_{s_2}$ are realizable
1  if $\alpha < 0$ then
2      return False;
3  Let $\mathcal{S}$ be the strategy for $\phi_1$;
4  $\psi$ := Encode-LTS($\mathcal{S}$);
5  $\phi'_2 := (\psi \wedge \phi_{e_2}) \rightarrow \phi_{s_2}$;
6  if $\phi'_2$ is realizable then
7      return $\psi$;
8  else
9      Let $\mathcal{CS}'$ be a counter-strategy for $\phi'_2$;
10     $\mathcal{P}$ := findCandidateAssumptions($\mathcal{CS}', U$);
11     foreach $\varphi \in \mathcal{P}$ do
12         Let $\phi''_2$ be $(\varphi \wedge \phi_{e_2}) \rightarrow \phi_{s_2}$;
13         Let $\phi''_1$ be $\phi_{e_1} \rightarrow (\phi_{s_1} \wedge \varphi)$;
14         if $\phi''_1$ is realizable and $\phi''_2$ is satisfiable then
15             if $\phi''_2$ is realizable then
16                 return $\varphi$;
17             else
18                 $\psi$ := compositionalRefinement3($\phi''_1, \phi''_2, \alpha - 1, U$);
19                 if $\psi \neq$ False then
20                     return $\psi \wedge \varphi$;
21 return False;
```

There are two rooms in bottom-left and the upper-right corners of the workspace protected by two doors D_1 (cell 10) and D_2 (cell 16). The robots can enter or exit a room through its door and only if it is open. The objective of R_1 (R_2) is to infinitely often visit the cell 5 (21, respectively). The global specification requires each robot to infinitely often visit their goal cells, while avoiding collision with each other, walls and the closed doors, i.e., the robots cannot occupy the same location simultanously, or switch locations in two following time steps, they cannot move to cells $\{4, 9, 17, 22\}$ (walls), and they cannot move to cells 10 or 16 if the corresponding door is closed. The doors are controlled by the environment and we assume that each door is always eventually open.

The global specification is realizable. We decompose the specification as follows. A local specification $\phi_1 = \phi_{e_1} \rightarrow \phi_{s_1}$ for R_1 where ϕ_{e_1} is the environment assumption on the doors and ϕ_{s_1} is a conjunction of R_1's guarantees which consist of its initial location, its transition rules, avoiding collision with walls and closed doors, and its goal to visit cell 5 infinitely often. A local specification $\phi_2 = \phi_{e_2} \rightarrow \phi_{s_2}$ for R_2 where ϕ_{e_2} includes assumptions on the doors, R_1's initial location, goal, and its transition rules, and ϕ_{s_2} consists of R_2's initial location, its transition rules, avoiding collision with R_1, walls and closed doors while fulfilling its goal. The specification ϕ_1 is realizable, but ϕ_2 is not. We use the algorithms outlined in Sect. 5 to find refinements for both components. We slightly modified the algorithms to find all refinements within the specified search depth. We use the variables corresponding to the location of R_1 for computing the abstraction and generating the candidate formulas. Furthermore, since the counter-strategies are large, computing all eventually and always eventually patterns is not feasible (may take years), and hence we only synthesize some of them.

Using the first approach along with abstraction, three refinements are found in 173 minutes which are conjunctions of safety and transition formulas. One of the computed refinements is $\psi_1 = \Box(Loc_{R_1} = 7 \rightarrow \bigcirc(Loc_{R_1} \notin \{7,8,12\})) \wedge \Box(Loc_{R_1} = 13 \rightarrow \bigcirc(Loc_{R_1} \notin \{12,14\})) \wedge \Box(Loc_{R_1} = 11 \rightarrow \bigcirc(Loc_{R_1} \neq 16)) \wedge \Box(Loc_{R_1} = 2 \rightarrow \bigcirc(Loc_{R_1} \neq 7)) \wedge \Box(Loc_{R_1} \notin \{2,12\})$. Intuitively, ψ_1 assumes some restrictions on how R_1 behaves, in which case a strategy for R_2 can be computed. Indeed, R_1 has a strategy that can guarantee ψ_1. Without using abstraction, four refinements are found within search depth 1 in 17 minutes. A suggested refinement is $\Box(Loc_{R_1} \notin \{7,12,16\})$, i.e., if R_1 avoids cells $\{7,12,16\}$, a strategy for R_2 can be computed. Using abstraction reduces the number of states of the counter-strategy from 576 to 12 states, however, not all the formulas that are satisfied by the counter-strategy, can be computed over its abstraction, as mentioned in Remark 1. Note that computing all the refinements within search depth 3 without using abstraction takes almost 5 times more time compared to when abstraction is used. Using the second approach (with and without abstraction) the refinement $\psi_2 = \Box(Loc_{R_1} = 10 \rightarrow Loc_{R_1} = 5)$ is found by infering fromulas from the strategy computed for R_1. Using abstraction slightly improves the process. Finally, using the third approach, providing either the full encoding or the abstract encoding of the strategy computed for ϕ_1 as assumptions for ϕ_2, makes the specification realizable. Therefore, no counter-strategy is produced, as knowing how R_1 behaves enables R_2 to find a strategy for itself.

Table 1 shows the experimental results for the case study. The columns specify the approach, whether abstraction is used or not, the total time for the experiment in minutes, number of strategies (counter-strategies) and number of states of the largest strategy (counter-strategy, respectively), the depth of the search, number of refinements found, and number of candidate formulas generated during the search. As it can be seen from the table, knowing more about the strategy chosen for the realizable specification can significantly reduce the time needed to find suitable refinement (from hours for the first approach to seconds for the third approach). However, the improvement in time comes with the cost of introducing more coupling between the components, i.e., the strategy computed for C_2 can become too dependent on the strategy chosen for C_1.

7 Conclusion and Future Work

We showed how automated refinement of specifications can be used to refine the specifications of the components in the context of compositional synthesis. We proposed three different approaches for compositional refinement of specifications. The choice of the appropriate approach depends on the size of the problem

Table 1: Evaluation of approaches on robot motion planning case study

Appr.	abstraction	time (min)	$\#_{\mathcal{S}}$	max $\lvert Q\rvert_{\mathcal{S}}$	$\#_{\mathcal{CS}}$	max $\lvert Q\rvert_{\mathcal{CS}}$	depth	$\#_{ref.}$	$\#_{candid.}$
1	yes	173.05	-	-	17	12	3	3	104
1	no	17.18	-	-	1	576	1	4	22
1	no	869.84	-	-	270	644	3	589	7911
2	yes	69.21	1	8	18	576	1	2	19
2	no	73.78	1	22	19	576	1	2	24
3	yes	0.01	1	8	0	0	1	1	0
3	no	0.02	1	22	0	0	1	1	0

(e.g., number of states in strategies and counter-strategies) and the level of acceptable coupling between components. Supplying more information about the strategies of the components with realizable local specifications to unrealizable specification under refinement, reduces the number of scenarios the game solver needs to consider, and facilitates the synthesis procedure, while increasing the coupling between components. Overall, patterns provide a tool for the designer to refine and complete temporal logic specifications. In future we plan to extend the methods to more general architectures.

References

1. Rosner, R.: Modular synthesis of reactive systems. Ann Arbor **1050** (1991) 48106–1346
2. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. Journal of Computer and System Sciences **78**(3) (2012) 911–938
3. Ozay, N., Topcu, U., Murray, R.: Distributed power allocation for vehicle management systems. In: CDC-ECC. (2011) 4841–4848
4. Alur, R., Moarref, S., Topcu, U.: Counter-strategy guided refinement of GR(1) temporal logic specifications. In: FMCAD. (2013) 31–44
5. Li, W., Dworkin, L., Seshia, S.: Mining assumptions for synthesis. In: MEMOCODE. (2011) 43–50
6. Chatterjee, K., Henzinger, T., Jobstmann, B.: Environment assumptions for synthesis. In: CONCUR. Springer (2008) 147–161
7. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: FoCS. (1990) 746–757
8. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: LICS, IEEE (2005) 321–330
9. Chatterjee, K., Henzinger, T.A.: Assume-guarantee synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2007) 261–275
10. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
11. Bloem, R., Cimatti, A., Greimel, K., Hofferek, G., Könighofer, R., Roveri, M., Schuppan, V., Seeber, R.: RATSY–a new requirements analysis tool with synthesis. In: CAV, Springer (2010) 425–429
12. Pnueli, A., Sa'ar, Y., Zuck, L.D.: JTLV: A framework for developing verification algorithms. In: CAV, Springer (2010) 171–174
13. McMillan, K.: Cadence SMV. http://www.kenmcmil.com/smv.html