Diagnostic Information for Realizability

A. Cimatti[1], M. Roveri[1], V. Schuppan[2], and A. Tchaltsev[1]

[1] Fondazione Bruno Kessler — IRST, Via Sommarive 18, 38050 Povo (TN) – Italy
{cimatti,roveri,tchaltsev}@fbk.eu
[2] Verimag/CNRS, 2 Av. de Vignate, 38610 Gières – France
Viktor.Schuppan@imag.fr

Abstract. Realizability – checking whether a specification can be implemented by an open system – is a fundamental step in the design flow. However, if the specification turns out not to be realizable, there is no method to pinpoint the causes for unrealizability. In this paper, we address the open problem of providing diagnostic information for realizability: we formally define the notion of (minimal) explanation of (un)realizability, we propose algorithms to compute such explanations, and provide a preliminary experimental evaluation.

1 Introduction

The role of properties in the design flow is becoming increasingly important [19,2]. Properties are used to describe design intent and to document designs and components, and play a fundamental role both in dynamic and static verification. As a result, research has been devoted to the development of new algorithms and tools for requirements analysis, in order to guarantee that the starting point of the process is indeed free from flaws. Typical forms of analysis are consistency checking, and compatibility with scenarios [14,4]. However, most property verification algorithms and tools are currently lacking the ability to provide diagnostic information that can support the debugging. This is potentially a major shortcoming. In fact, the practical success of model checking is tightly related to the ability of producing counterexamples (e.g., [10]): when the system violates a requirement, model checking algorithms are able to provide a simulation trace witnessing the violation, which may help the designer to find suitable fixes.

In this paper, we address the problem of providing diagnostic information for the realizability of the specification of an open system (e.g., a component). In this setting, requirements are typically separated in *assumptions* (i.e., the admissible behaviors of the environment), and *guarantees* (i.e., the behaviors must be implemented by the system-to-be). Intuitively, realizability is the problem of checking the existence of a system implementing the required guarantees, given that the environment can do whatever allowed by the assumptions.

We make two contributions. First, we tackle the problem of precisely characterizing the idea of diagnostic information for realizability problems. We propose notions for explanation and minimal explanation of (un)realizability. This issue is in fact non trivial: realizability could be achieved by relaxing the assertions on the system, or strengthening the assumptions on the environment. These notions can be also used to provide

F. Logozzo et al. (Eds.): VMCAI 2008, LNCS 4905, pp. 52–67, 2008.

diagnostic information for realizable specifications, i.e., we allow pinpointing minimal subsets of the specification that might be covered by the remaining part.

Second, we propose two methods to extend a practical, symbolic algorithm for realizability, in order to extract explanations and minimal explanations in case of (un)realizability. One of the algorithms is based on a explicit search in the space of subsets of the specification, and is able to compute one explanation at a time. The other one is fully symbolic in nature, and relies on the idea of activation variables to extract all explanations. We implemented the methods within the NuSMV system, for the class of Generalized Reactivity(1) [15] specifications, and we tested them on some industrial cases. The symbolic computation of all the explanations of (un)realizability turns out to be computationally expensive. On the other hand, the explicit algorithm can produce, with moderate performance penalty, explanations that are significantly smaller - sometimes more than an order of magnitude - than the original specifications.

Related Work. To the best of our knowledge, the notion of explanation of realizability has never been defined in terms of requirements. Production of diagnostic information in case of unrealizability was addressed in [18,6,3] and in [20]. In [18,6,3] a counter-strategy is constructed showing how the environment can force the system to violate its guarantees. Yoshiura [20] developed heuristics to classify reasons for unrealizability based on notions that are harder to fulfil than temporal satisfiability but easier than realizability. In both cases, (i) the diagnostic information is "global", i.e., it takes into account all the input problem, and (ii) the link to the requirements in the original problem is lost. Our approach can complement both [18,6,3] and [20] by providing a smaller, yet unrealizable specification to work on. In particular, a counter-strategy might exploit more than one source of unrealizability. Our approach can help to obtain a more focused counter-strategy. In terms of techniques, the fully symbolic algorithm is inspired by the idea of activation variables for the case of Boolean satisfiability [13]. Closely related is also the extension to the case of unsatisfiable core for LTL specifications proposed in [9], for the less complex case of satisfiability. Finally, there is a large body of work on fault localization and explanation in a verification context, where both a program and a (potentially implicit) specification are given. We refer the reader to the section on related work in Groce's Ph.D. thesis [12] for a survey.

Document structure. In Sect. 2 we define some technical background. In Sect. 3, we informally discuss and formalize the notion of explanation. In Sect. 4, we present the explanation-extracting algorithm. In Sect. 5, we discuss the implementation and present some experimental evaluation. Finally, in Sect. 6, we draw some conclusions and outline directions for future work.

2 Preliminaries

2.1 Synthesis of Open Controllers

We are interested in the question of *realizability* of an LTL property [16,1]. [1] We start from two disjoint sets $\mathcal{E}$ and $\mathcal{S}$ of input and output signals respectively, and from a formula φ expressed in LTL over atomic propositions on $\mathcal{E} \cup \mathcal{S}$ (written $\varphi(\mathcal{E}, \mathcal{S})$).

[1] We assume the reader being familiar with LTL syntax and semantics.

$\mathcal{E}$ is the set of variables controlled by the environment, while $\mathcal{S}$ is the set of variables controlled by the system. The *realizability problem* for a property φ consists of checking whether there exists a *program* such that its behavior satisfies φ [16]. An LTL formula $\varphi(\mathcal{E}, \mathcal{S})$ is then *realizable* iff there exists such a program. Properties for which such a program exists are called *realizable* or *implementable*. Dually, properties for which such a program does not exist are called *not realizable* or *unrealizable*.

The realizability problem can be formalized as a two player game among the system we are going to realize and the environment: the system plays against the environment in such a way that at every step of the game the environment moves and then the system tries to move by producing behaviors compatible with the property. The system wins if it produces a correct behavior regardless of the behavior of the environment. In this framework, checking for realizability amounts to check for the existence of a winning strategy for the system in the corresponding game. This is tackled by generating from the property a deterministic Rabin automaton using the Safra construction [17]. This automaton is interpreted as a two player game among the system and environment and it is traversed as to find a witness of the non emptiness of the language of the automaton (which corresponds to a correct implementation of the given property) [16].

2.2 Assumptions and Guarantees

Practically a specification is often represented with two distinguished sets – a set of *assumptions* A and a set of *guarantees* G – plus a function f that turns such a set of constraints into an actual temporal formula φ using Boolean and temporal connectives. Under this assumption a specification is given as a tuple $\langle A, G \rangle$. Intuitively, assumptions are those constraints which the environment is supposed to obey to and guarantees are those constraints which the system has to satisfy. The function f has to have such a form that realizability is preserved by adding assumptions to or removing guarantees from an already realizable specification and, conversely, unrealizability is preserved by removing assumptions from or adding guarantees to an unrealizable specification. Similarly, adding a valid constraint to either assumptions or guarantees must not influence the realizability of a specification.[2] Note, that both A and G may be structured, such that f may not treat all elements of A and G in the same way. In the conceptual part of this work in Sect. 3 we are not concerned with the exact nature of the translation and view A and G as flat sets of atomic elements; only when we consider a concrete class of specifications (see below) for implementation we look into the structure of assumptions and guarantees. We denote the temporal formula resulting from $\langle A, G \rangle$ by applying f with $\phi_{\langle A,G \rangle} = f(\langle A, G \rangle)$. We say that $\langle A, G \rangle$ is realizable iff $\phi_{\langle A,G \rangle}$ is realizable.

2.3 Synthesis of GR(1) Properties

The high complexity established in [16] and the intricacy of Safra's determinization construction have caused the synthesis process to be identified as hopelessly intractable

[2] In this paper we need this property only in Sect. 4.2.

and discouraged many practitioners from ever attempting to implement it. However, there are several classes of properties restricted to particular subsets of LTL, which can be synthesized with more efficient algorithms. One of the most recent and advanced results is achieved in [15] where for the class of *Generalized Reactivity*(1) specifications (from now on referred to as GR(1) specification) is presented a (symbolic) algorithm for extracting a program from a GR(1) specification that runs in time polynomial in the size of the state space of the design. The class of GR(1) properties is sufficiently expressive to provide complete specifications of many designs [15].

A GR(1) specification has the form $\langle A, G\rangle = (\{\varphi_I^{\mathcal{E}}, \varphi_R^{\mathcal{E}}, \varphi_\psi^{\mathcal{E}}\}, \{\varphi_I^{\mathcal{S}}, \varphi_R^{\mathcal{S}}, \varphi_\psi^{\mathcal{S}}\})$. [3] For $\alpha \in \{\mathcal{E}, \mathcal{S}\}$, $\varphi_I^\alpha, \varphi_R^\alpha, \varphi_\psi^\alpha$ represent the *initial conditions*, the *transition relation* and the *liveness* or *fairness* conditions of the environment and system, respectively. They are such that:

- φ_I^α - a formula of the form $\bigwedge_i I_i$ where every I_i is a propositional formula over signals ($\varphi_I^{\mathcal{E}}$ is over $\mathcal{E}$ and $\varphi_I^{\mathcal{S}}$ is over $\mathcal{E} \cup \mathcal{S}$).
- φ_R^α - temporal formulas of the form $\bigwedge_i R_i$ where every R_i is a propositional formula over signals $\mathcal{E} \cup \mathcal{S}$ and expressions of the form $\mathbf{X}\, v$ where $v \in \mathcal{E}$ if $\alpha = \mathcal{E}$ and $v \in \mathcal{E} \cup \mathcal{S}$ if $\alpha \in \mathcal{S}$.
- φ_ψ^α - temporal formulas of the form $\bigwedge_i \mathbf{G}\,\mathbf{F}\, A_i$ where A_i is propositional formula over signals $\mathcal{E} \cup \mathcal{S}$.

Intuitively, the play is initialized in such a way that the environment chooses initial values for its signals as to satisfy $\varphi_I^{\mathcal{E}}$, and the system initializes its signals to satisfy $\varphi_I^{\mathcal{S}}$. At every consecutive step of the play at first the environment assigns its signals, trying to satisfy the environment transition relation $\varphi_R^{\mathcal{E}}$, and then the system does the same with its signals and its transition relation $\varphi_R^{\mathcal{S}}$. For an infinite behavior the environment and the system try to satisfy their liveness conditions $\varphi_\psi^{\mathcal{E}}$ and $\varphi_\psi^{\mathcal{S}}$, respectively. The player who first violates its constraints loses.

Realizability of a GR(1) specification can be reduced to the problem of computing the set of winning states W_S in a two-player game among the environment and the system and then checking W_S against initial conditions [15]. In the following we will use the algorithm of [15] to check for the realizability of a GR(1) specification $\langle A, G\rangle$.

3 Diagnosing (Un)Realizability

In this section we discuss what information can be returned to a developer in the case a given specification $\langle A, G\rangle$ turns out to be either unrealizable or realizable. We focus on "zooming into" the specification by pointing out fragments of the specification that are by themselves (un)realizable, in order to facilitate the understanding of the problem.

We therefore suggest to use a specification $\langle A', G'\rangle$ as an *explanation* for a specification $\langle A, G\rangle$ where A', G' are subsets of A, G. We first formalize minimality and maximality constraints on A' or G'. We then introduce a notion of unhelpfulness of assumptions or guarantees in an explanation, where unhelpful assumptions or guarantees can be removed from an explanation. We illustrate the concept with an example.

[3] We refer the reader to [15] for details on the corresponding LTL formula.

3.1 Explanations for (Un)Realizability

We first notice that assumptions and guarantees can be viewed as interacting, but opposing forces. As outlined in Sect. 2, adding assumptions or removing guarantees will "push" a specification towards realizability. Conversely, a realizable specification may become unrealizable when deleting assumptions or adding guarantees. These concepts are formalized as follows.

Definition 1 ((un-) fulfillable, (in-) sufficient). *Let $\mathcal{A}$ be a set of available assumptions, let $\mathcal{G}$ be a set of available guarantees, and let $A \subseteq \mathcal{A}$ and $G \subseteq \mathcal{G}$.*

If a specification $\langle A, G\rangle$ is realizable, we say that G is fulfillable *w.r.t. A, and, conversely, A is* sufficient *w.r.t. G. Otherwise, G is* unfulfillable *w.r.t. A, and A is* insufficient *w.r.t. G, respectively.*

G is minimally unfulfillable *w.r.t. A iff $\langle A, G\rangle$ is unrealizable and removal of any element of G leads to realizability: $\forall g \in G \,.\, \langle A, G \setminus \{g\}\rangle$ is realizable.*

G is maximally fulfillable *w.r.t. A in $\mathcal{G}$ iff $\langle A, G\rangle$ is realizable and addition of any element of $\mathcal{G} \setminus G$ leads to unrealizability: $\forall g \in \mathcal{G} \setminus G \,.\, \langle A, G \cup \{g\}\rangle$ is unrealizable.*

A is minimally sufficient *w.r.t. G iff $\langle A, G\rangle$ is realizable and removal of any element of A leads to unrealizability: $\forall a \in A \,.\, \langle A \setminus \{a\}, G\rangle$ is unrealizable.*

A is maximally insufficient *w.r.t. G in $\mathcal{A}$ iff $\langle A, G\rangle$ is unrealizable and addition of any element of $\mathcal{A} \setminus A$ leads to realizability: $\forall a \in \mathcal{A} \setminus A \,.\, \langle A \cup \{a\}, G\rangle$ is realizable.*

All above definitions are also transferable to a whole specification, i.e., a specification $\langle A, G\rangle$ is *maximally insufficient* iff A is maximally insufficient w.r.t. G, etc.

Why is separate terminology introduced for assumptions and guarantees? After all, if a specification $\langle A, G\rangle$ is unrealizable, then A is insufficient w.r.t. G and G is unfulfillable w.r.t. A (similarly for a realizable specification). However, while A is insufficient w.r.t. G iff G is unfulfillable w.r.t. A, A might, e.g., be maximally insufficient w.r.t. G although G is unfulfillable but not minimally unfulfillable w.r.t. A. In other words, minimality and maximality require introduction of separate terminology for both sides.

We now show how the above definitions can provide an explanation for an (un)realizable specification $\langle A, G\rangle$.

Minimally Unfulfillable Sets of Guarantees. First, assume that $\langle A, G\rangle$ is unrealizable. To understand the nature of the problem, the developer needs to see which sets of guarantees are not supported by sufficient assumptions or which sets of guarantees are conflicting. Hence, we suggest to return an explanation $\langle A, G'\rangle$ such that $G' \subseteq G$ is minimally unfulfillable. Each such G' is a minimal set of guarantees such that either A is not strong enough to realize G, or the elements of G are in conflict with each other. Clearly, there may be several such sets. The quest for minimality is based on the intuition that if a guarantee does not contribute to making a specification unrealizable then it can be omitted from the explanation.

Maximally Fulfillable Sets of Guarantees. While an explanation of the previous kind helps to find the cause of unrealizability, it does not immediately suggest a fix. Our second suggestion provides fixes in the restricted case that a fix is only allowed to remove guarantees. Obviously, such fix should remove as few guarantees as possible to achieve realizability. Hence, we suggest to provide the developer with a maximally fulfillable set of guarantees G' as an explanation. Notice that, addition of any $g \in G \setminus G'$

will make $\langle A, G' \cup \{g\}\rangle$ unrealizable. I.e., the complement of each such G' constitutes a minimal set of guarantees that, when removed from G, leads to realizability.

Note that, the distinction to minimally unfulfillable sets of guarantees as an explanation becomes particularly interesting when there is more than one set of unfulfillable guarantees. In that case a minimal fix is easier to see by providing the developer with a maximally fulfillable set of guarantees rather than with several minimally unfulfillable sets of guarantees as in the latter case the developer has to figure out herself which combinations of guarantees need to be removed to avoid all "unfulfillabilities".

A slightly different scenario where maximally fulfillable sets of guarantees can help is finding out the set of guarantees that may be realized with a given set of assumptions, i.e., strengthening the guarantees that the system under design will provide. In this case, given a set of available guarantees, $\mathcal{G} \supset G$, it is enough to compute the maximally fulfillable sets of guarantees for $\langle A, \mathcal{G}\rangle$.

Minimally Sufficient Sets of Assumptions. If $\langle A, G\rangle$ is realizable, the need for debugging information is less urgent. Still, the developer might benefit from additional information that helps her understanding. In particular, we suggest to point out minimal sets of assumptions A' that, on their own, are sufficient to realize a given set of guarantees G. If $\langle A, G\rangle$ is the original specification, A' may help to reduce the assumptions the environment has to fulfill. Another scenario is that G is only a subset of the guarantees under consideration. Here, the developer might want to understand which subset(s) of assumptions A' are responsible for realizability of this particular set of guarantees. It's easy to see that in both cases A' is a set of minimally sufficient assumptions.

If $\langle A, G\rangle$ turns out to be unrealizable and the set of available assumptions has not been exhausted (i.e., $A \subset \mathcal{A}$), minimally sufficient sets of assumptions for $\langle \mathcal{A}, G\rangle$ can help to find a minimal strengthening of A such that G can be realized.

Maximally Insufficient Sets of Assumptions We have not found a good intuition on how to use these as a debugging aid. We omit such sets from further consideration.

3.2 Criteria for Unhelpful Parts of an Explanation

Till now we proposed to remove constraints either only from assumptions or only from guarantees. We now proceed to remove constraints from the remaining side of an explanation. We first argue why constraints can be removed from both sides of a specification. We then formulate a criterion to identify *helpful* and *unhelpful* constraints.

Removing Constraints from the Remaining Side of an Explanation. As argued above, removing guarantees from an unrealizable specification or removing assumptions from a realizable specification is a natural approach to obtain a core of a specification that explains its (un)realizability. However, previously we have only modified one side of a specification to obtain an explanation. As mentioned, removing assumptions pushes towards unrealizability and removing guarantees pushes towards realizability. Hence, one might think that an explanation for unrealizability (in the form of a minimally unfulfillable specification) should contain the full set of assumptions A and, similarly, an explanation for realizability (in the form of a minimally sufficient specification) should contain the full set of guarantees G. Note, though, that an assumption or a guarantee might be redundant. In other words, it might be covered by the remaining

set of assumptions or guarantees. Moreover, some assumptions or guarantees might be irrelevant w.r.t. a given explanation, i.e., they may have no influence on the behavior of a specification, we are interested in. We believe that the developer should be informed about such assumptions and guarantees. Below we expand that idea for the three types of explanations proposed in Sect. 3.1.

Minimally Unfulfillable Sets of Guarantees. The aim of a minimally unfulfillable explanation $\langle A, G\rangle$ is to show a conflict among the set of guarantees G or the lack of assumptions required for realizability of G. It is possible that some of the assumptions in A do not contribute to that aim, i.e., they do not influence the conflict among, or the realizability of, the guarantees G. Such assumptions may be removed from an explanation without losing valuable information, thereby making it simpler for understanding.

Maximally Fulfillable Sets of Guarantees. The purpose of a maximally fulfillable explanation $\langle A, G\rangle$ is to show which set(s) of guarantees can be realizable with a given set of assumptions or which set of guarantees are enough to remove to make the specification realizable. If removing an assumption a does not change realizability of an explanation, i.e., $\langle A \setminus \{a\}, G\rangle$ is realizable, then presence of such an assumption does not influence the property of the set G being maximally fulfillable. Indeed, since $\langle A, G \cup \{g\}\rangle$ is unrealizable for any $g \in \mathcal{G} \setminus G$ then $\langle A \setminus \{a\}, G \cup \{g\}\rangle$ is also unrealizable for any a because removing an assumption cannot make an unrealizable specification realizable. Therefore, if such an assumption is removed an explanation still fulfills its purpose and shows a maximal set of realizable guarantees.

Minimally Sufficient Sets of Assumptions. The purpose of a minimally sufficient explanation $\langle A, G\rangle$ is to point out a set of assumptions A that is enough to make a given set of guarantees G realizable such that each assumption $a \in A$ is essential for realizability. This case is symmetrical to the case of minimally unfulfillable set of guarantees, i.e., not every guarantee may be useful in such an explanation — some guarantees may be realizable independent of the assumptions, or one assumption may be essential for realizability of several guarantees therefore only one of such guarantees may be left in the explanation to show necessity of that assumption.

Formalization. We are now ready to formulate a criterion of when a constraint in an explanation should be considered unhelpful. Our intuition is as follows. Let $\langle A, G\rangle$ be an explanation, let $a \in A$ be an assumption. We say that a is *helpful* iff there is some subset of guarantees $G' \subseteq G$ s.t. $\langle A, G'\rangle$ is realizable, while $\langle A \setminus \{a\}, G'\rangle$ is not. In other words, there is a subset of guarantees G' s.t. a makes the difference between realizability and unrealizability for that subset (w.r.t. the given set of assumptions A). Similarly, a guarantee $g \in G$ is helpful iff there is at least one subset of assumptions $A' \subseteq A$ s.t. g make the difference between realizability and unrealizability: $\langle A', G \setminus \{g\}\rangle$ is realizable while $\langle A', G\rangle$ is not. We formalize that intuition below.

Definition 2 ((un-) helpful). *Let $\langle A, G\rangle$ be a specification.*

1. *An assumption $a \in A$ is* unhelpful *if*
   $\forall G' \subseteq G$. ($\langle A, G'\rangle$ *is realizable* $\Leftrightarrow \langle A \setminus \{a\}, G'\rangle$ *is realizable.*)
2. *A guarantee $g \in G$ is* unhelpful *if*
   $\forall A' \subseteq A$. ($\langle A', G\rangle$ *is realizable* $\Leftrightarrow \langle A', G \setminus \{g\}\rangle$ *is realizable.*)
3. *An assumption or a guarantee is* helpful *iff it is not unhelpful.*

The next proposition shows that Def. 2 is well-behaved in the following sense. If $\langle A, G\rangle$ is an explanation and $A' \subset A$ is obtained from A by a sequence of removals of unhelpful assumptions A'' (by a sequence of applications of Def. 2), then each of the removed assumptions in A'' is unhelpful also in A. Moreover, the assumptions in A'' could have been removed from A in any order. The case for guarantees is similar.

Proposition 1

1. *Let* $\langle A_0 = A, G\rangle, \langle A_1 = A_0 \setminus \{a_0\}, G\rangle, \langle A_2 = A_1 \setminus \{a_1\}, G\rangle, \ldots, \langle A' = A_k \setminus \{a_k\}, G\rangle$ *be a sequence of explanations s.t., for all* $0 \leq i \leq k$*, each* a_i *is unhelpful in* $\langle A_i, G\rangle$*. Let* $A'' \subseteq A \setminus A'$*, let* $a \in A''$*. Then* a *is unhelpful in* $\langle A' \cup A'', G\rangle$*.*
2. *Let* $\langle A, G_0 = G\rangle, \langle A, G_1 = G_0 \setminus \{g_0\}\rangle, \langle A, G_2 = G_1 \setminus \{g_1\}\rangle, \ldots, \langle A, G' = G_k \setminus \{g_k\}\rangle$ *be a sequence of explanations s.t., for all* $0 \leq i \leq k$*, each* g_i *is unhelpful in* $\langle A, G_i\rangle$*. Let* $G'' \subseteq G \setminus G'$*, let* $g \in G''$*. Then* g *is unhelpful in* $\langle A, G' \cup G''\rangle$*.*

The following proposition yields a practical way to remove unhelpful constraints from the remaining side of an explanation. Given $\langle A, G\rangle$ minimally unfulfillable, we suggest to remove assumptions from A until the result is not *minimally* unfulfillable any more (resp., if $\langle A, G\rangle$ is minimally sufficient, remove guarantees from G until the result is not *minimally* sufficient).

Proposition 2

1. *Let* $\langle A, G\rangle$ *be a minimally unfulfillable specification.* $a \in A$ *is unhelpful in* A *iff* $\langle A \setminus \{a\}, G\rangle$ *is minimally unfulfillable.*
2. *Let* $\langle A, G\rangle$ *be a minimally sufficient specification.* $g \in G$ *is unhelpful in* G *iff* $\langle A, G \setminus \{g\}\rangle$ *is minimally sufficient.*

Our next proposition shows the coincidence between Def.s 1 and 2. In particular, it shows that an unrealizable specification $\langle A, G\rangle$ contains no unhelpful guarantees iff it is minimally unfulfillable and a realizable specification $\langle A, G\rangle$ contains no unhelpful assumptions iff it is minimally sufficient.

Proposition 3

1. *Let* $\langle A, G\rangle$ *be unrealizable.* $g \in G$ *is unhelpful in* G *iff* $\langle A, G \setminus \{g\}\rangle$ *is unrealizable.*
2. *Let* $\langle A, G\rangle$ *be unrealizable.* G *is minimally unfulfillable iff all* $g \in G$ *are helpful.*
3. *Let* $\langle A, G\rangle$ *be realizable.* $a \in A$ *is unhelpful in* A *iff* $\langle A \setminus \{a\}, G\rangle$ *is realizable.*
4. *Let* $\langle A, G\rangle$ *be realizable.* A *is minimally sufficient iff all* $a \in A$ *are helpful.*

Thus Def. 2 can be used to obtain minimally unfulfillable explanations from unrealizable specifications (by removing unhelpful guarantees) and minimally sufficient explanations from realizable specifications (by removing unhelpful assumptions).

Putting the Pieces Together. In Fig. 1 we show the approach that applies the previous results according to the types of explanations suggested in Sect. 3.1.

3.3 Methodology

Unsatisfiable Assumptions and Guarantees. Sometimes subsets of assumptions or guarantees may be temporally unsatisfiable. Such situations should be pointed out to

Explaining Unrealizability — a Minimal Conflict

1. Assume $\langle A, G\rangle$ unrealizable.
2. Find some $G' \subseteq G$ s.t. $\langle A, G'\rangle$ is minimally unfulfillable.
3. Find a minimal $A' \subseteq A$ s.t. $\langle A', G'\rangle$ is minimally unfulfillable.
4. Return $\langle A', G'\rangle$.

Start with an unrealizable specification. First, remove unhelpful guarantees, then remove unhelpful assumptions. Now, every single guarantee in G' is required for a conflict; moreover, removing any assumption from A' leads to additional conflict(s), each involving fewer guarantees.

Explaining Unrealizability — a Minimal Fix

1. Assume $\langle A, G\rangle$ unrealizable.
2. Find some $G' \subseteq G$ s.t. $\langle A, G'\rangle$ is maximally fulfillable.
3. Find some $A' \subseteq A$ s.t. $\langle A', G'\rangle$ is minimally sufficient.
4. Return $\langle A', G'\rangle$.

Start with an unrealizable specification. First, remove just enough guarantees to make the specification realizable, then remove unhelpful assumptions. Now, adding any guarantee or removing any assumption leads to unrealizability. Moreover, $G \setminus G'$ is a minimal fix to make the original specification $\langle A, G\rangle$ realizable.

Explaining Realizability

1. Assume $\langle A, G\rangle$ realizable.
2. Find some $A' \subseteq A$ s.t. $\langle A', G\rangle$ is minimally sufficient.
3. Find a minimal $G' \subseteq G$ s.t. $\langle A', G'\rangle$ is minimally sufficient.
4. Return $\langle A', G'\rangle$.

Start with a realizable specification. First, remove unhelpful assumptions, then remove unhelpful guarantees. Now, every single assumption in A' is required for realizability; removing any guarantee in G' makes one or more assumptions unnecessary for realizability.

Fig. 1. A summary of our approach

the developer; however, as these situations may not be uniquely identifiable from the explanations suggested above, a separate check has to be performed. Satisfiability can be checked in various ways. A detailed treatment is out of the scope of this work. We therefore assume that the specification has been checked for satisfiability (in particular, the checks suggested in [20]) before applying our method.

Removing Unhelpful Constraints. When a specification is checked for unhelpful constraints, it is important to note that several constraints that have been found unhelpful cannot be removed at once. For example, if for a specification $\langle A, G\rangle$ the individual assumptions $a_1, a_2 \in A$ are found to be unhelpful, they should not be removed at once. Rather, it is necessary to remove one of them (e.g., a_1) and then recheck the second assumption a_2 for being unhelpful in $\langle A \setminus \{a_1\}, G\rangle$. Otherwise, the result can be incorrect. For example, if a_1 and a_2 are equivalent, they will always be unhelpful. Nevertheless, removing *both* of them can change realizability of the specification. Therefore, constraints can be checked and removed (if found unhelpful) only one by one.

3.4 Examples

Let us consider the following example with the assumptions on the left and guarantees on the right and e and s being an environment and a system variable (there may be other constraints that do not speak about e and s):

$$\begin{array}{ll} a_1 \doteq e & g_1 \doteq s \\ a_2 \doteq \mathbf{G}((\mathbf{X}\, e) \leftrightarrow e) & g_2 \doteq \mathbf{G}((\mathbf{X}\, s) \leftrightarrow e) \\ a_3 \doteq \mathbf{G}\,\mathbf{F}\, e & g_3 \doteq \mathbf{G}\,\mathbf{F}(\neg s \wedge e) \\ \ldots & \ldots \end{array}$$

The specification is unrealizable with a minimal conflict explanation $\langle A, G\rangle = \langle\{a_3\}, \{g_2, g_3\}\rangle$. A minimally unfulfillable guarantee set G shows the reason of unrealizability (i.e., the system cannot simultaneously make s equal to the previous value of e and at the same time reach $\neg s \wedge e$) and that the initial condition g_1 does not influence this conflict. The presence of the assumption a_3 is enough to show this conflict. By removing a_3 the explanation will not be minimally unfulfillable any more since it will remain unrealizable even without g_2 thereby losing the information about the original conflict among guarantees g_2 and g_3. Thus a_3 is required for the explanation.

4 Computing Explanations

In this section we describe our approach to computing explanations for specifications. First we explain explicit algorithms that are aimed to compute one explanation for a given specification, and we estimate their complexity. Then, we outline an alternative approach to computing explanations based on the use of activation variables [13,9].

4.1 Explicit Algorithms

In the most simplistic setting we assume that there is an external realizability checker considered as a black box and available for us as a function *Realizable*$(\langle A, G\rangle)$, which takes a specification $\langle A, G\rangle$ and returns $true$ iff the specification is realizable.

Among the possible kinds of explanations summarized in Fig. 1 let us begin with an unrealizable specification and its explanation in the form of a minimal conflict. The first step of the computation is to obtain a minimally unfulfillable set of guarantees. For that it is enough to identify which guarantees after removal keep the specification unrealizable. Propositions 1 and 3 establish that guarantees can be removed in any order.[4] In Sect. 3.3 we noticed that a check for realizability has to be done after each removal of any individual guarantee. As a result a simple algorithm to compute a minimally unfulfillable explanation for a given specification $\langle A, G\rangle$ is:

[4] Note, though, that while the order of removal of guarantees in a particular set of unhelpful guarantees $G' \subseteq G$ from G does not matter, it is still possible that there are different sets of unhelpful guarantees $G' \neq G''$ such that both $G \setminus G'$ and $G \setminus G''$ contain no unhelpful guarantees anymore (and similarly for assumptions). As a consequence, the algorithms presented here find minimal but not necessarily minimum explanations.

function *ComputeMinUnfulfil*$(\langle A, G\rangle)$
  $G' := G$;
  foreach $g \in G$
    if $\neg$*Realizable*$(\langle A, G' \setminus \{g\}\rangle)$ **then** $G' := G' \setminus \{g\}$;
  return $\langle A, G'\rangle$;

The second step of obtaining a "good" explanation is to remove unhelpful assumptions. Proposition 2 shows that it is enough to detect and remove assumptions whose removal keeps the specification minimally unfulfillable. Notice that, similarly to the previous algorithm it is necessary to check and remove only one assumption at every iteration. Thus the simplest algorithm is:

function *ComputeGoodMinUnfulfil*$(\langle A, G\rangle)$
  $A' := A$;
  foreach $a \in A$
    if *MinUnfulfil*$(\langle A' \setminus \{a\}, G\rangle)$ **then** $A' := A' \setminus \{a\}$;
  return $\langle A', G\rangle$;

where the predicate *MinUnfulfil*$(\langle A, G\rangle)$ returns $true$ iff the specification $\langle A, G\rangle$ is minimally unfulfillable:

$$\textit{MinUnfulfil}(\langle A, G\rangle) \doteq \forall g \in G \,.\, \textit{Realizable}(\langle A, G \setminus \{g\}\rangle)$$

Notice that, all above functions *ComputeMinUnfulfil*, *ComputeGoodMinUnfulfil* and *MinUnfulfil* expect as input an unrealizable specification.

In the case of computing explanations for realizable specifications (see Fig. 1) the corresponding algorithms are symmetric to the algorithms for unrealizable specifications explained above. Hence the functions *ComputeMinSuffic*, *ComputeGoodMinSuffic* and the predicate *MinSuffic* are defined similarly as *ComputeMinUnfulfil*, *ComputeGoodMinUnfulfil*, and *MinUnfulfil*, respectively, by switching realizability and unrealizability and the player whose constraints are minimized.

A minimal fix for an unrealizable specification according to Fig. 1 is also computed in two steps. The first step is to identify a maximal set of guarantees making the specification realizable. The simplest algorithm is very similar to *ComputeMinUnfulfil* with the exception that now the aim is to make the specification realizable and maximize the number of guarantees:

function *ComputeMaxFulfil*$(\langle A, G\rangle)$
  $G' := \emptyset$;
  foreach $g \in G$
    if *Realizable*$(\langle A, G' \cup \{g\}\rangle)$ **then** $G' := G' \cup \{g\}$;
  return $\langle A, G'\rangle$;

The second step is to find a minimally sufficient set of assumptions. For that the function *ComputeMinSuffic* defined above can be used.

To summarize, if a specification $\langle A, G\rangle$ is unrealizable and the cause of unrealizability is of interest, then an explanation is computed as

ComputeGoodMinUnfulfil(*ComputeMinUnfulfil*($\langle A, G\rangle$)). If a minimal fix is required, then *ComputeMinSuffic*(*ComputeMaxFulfil*($\langle A, G\rangle$)) is computed. Otherwise, if the specification $\langle A, G\rangle$ is realizable, the minimization of assumptions can be done and *ComputeGoodMinSuffic*(*ComputeMinSuffic*($\langle A, G\rangle$)) is returned.

Complexity. Let us assume that the upper bound on the time of checking the realizability of a specification $\langle A, G\rangle$ is denoted as $[\langle A, G\rangle]$, and that this upper bound cannot increase with the removal of some constraints from either A or G. Let $|A|$ and $|G|$ be the number of assumptions and guarantees, respectively. Then it is easy to see that the upper bound on the time of computing a minimal conflict for an unrealizable specification is $(|G| + |A| * |G|) * [\langle A, G\rangle]$, where $|G| * [\langle A, G\rangle]$ is the upper bound for the first step and $|A| * |G| * [\langle A, G\rangle]$ is for the second one. Similarly, the upper bound on computing an explanation for a realizable specification is $(|A| + |A| * |G|) * [\langle A, G\rangle]$, and $(|A| + |G|) * [\langle A, G\rangle]$ is the upper bound on computing a minimal fix for an unrealizable specification.

Notice that, for both, good minimally unfulfillable explanations and good minimally sufficient explanations, the number of realizability checks for computing a minimally unfulfillable set of guarantees (resp. a minimally sufficient set of assumptions), is linear in the number of constraints. While, for reducing the set of assumptions (resp. guarantees), the number of realizability checks may be quadratic.

4.2 Algorithms with Activation Variables

An alternative approach to computing explanations inspired by [13,9] works as follows. In a specification $\langle A, G\rangle$ for every constraint $c_i \in A \cup G$ a fresh activation variable av_i is created and then c_i is substituted by $av_i \rightarrow c_i$, obtaining in such a way the specification $\langle A^{AV}, G^{AV}\rangle$. Activation variables, differently from usual variables, cannot change their values after their initialization, and they belong neither to the system nor to the environment.

According to Sect. 2.2 the addition of the constraint $true$ to assumptions or guarantees cannot change the realizability of a specification. Thus, setting an activation variable av_i to $false$ disables the corresponding constraint c_i in the specification $\langle A^{AV}, G^{AV}\rangle$, whereas setting av_i to $true$ makes the constraint $av_i \rightarrow c_i$ behave the same as the original one c_i. If a realizability checker is able to find assignments to activation variables that make a specification (un)realizable, then using these assignments we can directly identify which subsets of constraints cause (un)realizability of the specification. The algorithm for the class of GR(1) specifications mentioned in Sect. 2.3 is able to do that without any modifications. Given a modified specification $\langle A^{AV}, G^{AV}\rangle$ after finding winning states W_S and checking it against initial conditions the obtained result is not just a constant $true$ or $false$ but a formula over the activation variables. Each assignment that makes that formula $true$ identifies a subset of the constraints that make the specification realizable.

The major difference from the previously described algorithms is that with activation variables one call to the realizability checker is enough to find all (un)realizable subsets of the constraints. Unfortunately, experimental results showed that introduction of new variables to the game slows down the realizability check considerably. As a result the computation of explanations with activation variables is often much slower than using the explicit algorithms described in Sect. 4.1.

5 Experimental Evaluation

We implemented all the algorithms described in Sect. 4 plus the algorithm for GR(1) synthesis [15] within the framework of the NuSMV system [7]. We applied several optimizations to the algorithm for checking realizability of [15] as to improve the performance. For instance, we compute the set of reachable states of the game structure and we use such set during the realizability check to restrict the search only to reachable states. The different explicit algorithms have been also optimized as to possibly re-use as much as possible results of previous checks. The implementation of activation variables takes into account that they remain constant after the initial state.

We evaluated our algorithms on two real-life specifications parametric on the number of components: the ARM AMBA AHB Arbiter [5] and the IBM GenBuf Controller[6]. We took these specifications from [5]: since that paper is about showing feasibility of synthesis, both specifications are realizable. We remark that, we were not able to find real-life unrealizable specifications in the literature. As we have pointed out before, we can make a GR(1) specification unrealizable by adding constraints to $\varphi_I^{\mathcal{S}}$, $\varphi_R^{\mathcal{S}}$, or $\phi_\psi^{\mathcal{S}}$, or by removing constraints from $\varphi_I^{\mathcal{E}}$, $\varphi_R^{\mathcal{E}}$, or $\varphi_\psi^{\mathcal{E}}$. We simulate the cases of adding to $\varphi_R^{\mathcal{S}}$ (referred to as W-GT), adding to $\varphi_\psi^{\mathcal{S}}$ (referred to as W-GF), and removing from $\varphi_\psi^{\mathcal{E}}$ (referred to as WO-AF).

We ran the experiments on an Intel Xeon 3GHz bi-processor equipped with 4GB of RAM running Linux. We fixed a memory limit to 1.5GB and a time-out to 1 hour. We report "T" and "M", respectively, when a time-out or a memory limit is reached. We used BDD dynamic variable ordering during the search since this resulted in better performances on average. All the experiments and an extended version of this paper [8] are available from `http://es.fbk.eu/people/roveri/tests/vmcai08`.

The table below shows the results of experiments with activation variables:

Specification Name	Assumptions/ Guarantees	Realizable	Time Realizability	Time Step 1	Time Step 1 and 2
AMBA-1	8/52	R	0.14	0.24	212
AMBA-W-GF-1	8/53	U	0.02	587	T

The first three columns show the name of a specification, its size, and its realizability, respectively. The fourth column gives the original realizability checking time (in seconds). The fifth column lists the checking time if only assumptions (for a realizable specification) or only guarantees (for an unrealizable one) are given activation variables — this corresponds to step 1 of the explicit algorithms. The last column shows realizability checking times if all constraints are given activation variables — this corresponds to both steps of the explicit algorithms.

The above results show how significantly activation variables may slow down the realizability check. This is the reason why only two specifications are in the table. We remark that (1) the algorithm using activation variables computes minimum rather than just minimal cores and (2) computing minimum cores by using activation variables has

[5] ARM Ltd. AMBA Specification (Rev. 2). Available from `www.arm.com`, 1999.

[6] `http://www.haifa.ibm.com/projects/verification/RB_Homepage/tutorial3/`

Table 1. Computation of explanations using explicit algorithms

Specification Name	Assumpt/ Guarant	Real- izable	Time Re- alizability	Time Step 1	Reduction Step 1	Time Step 2	Reduction Step 2
AMBA-1	8 / 52	R	0.14	0.25	**75.0%** (2 / 52)	1.93	**65.4%** (2 / 18)
AMBA-1-W-GF	8 / 53	U	0.02	0.24	**92.5%** (8 / 4)	0.02	**87.5%** (1 / 4)
AMBA-1-W-GT	8 / 53	U	0.02	0.21	**96.2%** (8 / 2)	0.01	**100%** (0 / 2)
AMBA-1-WO-AF	5 / 52	U	0.09	0.41	**76.9%** (5 / 12)	0.07	**100%** (0 / 12)
AMBA-2	11 / 80	R	1.22	2.97	**63.6%** (4 / 80)	64.1	**68.8%** (4 / 25)
AMBA-2-W-GF	11 / 81	U	0.19	1.06	**88.9%** (11 / 9)	0.12	**72.7%** (3 / 9)
AMBA-2-W-GT	11 / 81	U	0.17	0.97	**91.4%** (11 / 7)	0.06	**81.8%** (2 / 7)
AMBA-2-WO-AF	10 / 80	U	0.19	1.47	**87.5%** (10 / 10)	0.28	**100%** (0 / 10)
AMBA-3	14 / 108	R	14.3	35.2	**85.7%** (2 / 108)	26.7	**86.1%** (2 / 15)
AMBA-3-W-GF	14 / 109	U	0.51	4.31	**94.5%** (14 / 6)	0.09	**92.9%** (1 / 6)
AMBA-3-W-GT	14 / 109	U	0.39	2.92	**97.2%** (14 / 3)	0.04	**100%** (0 / 3)
AMBA-3-WO-AF	13 / 108	U	1.73	15.8	**90.7%** (13 / 10)	0.54	**100%** (0 / 10)
AMBA-4	17 / 136	R	74.9	292	**64.7%** (6 / 137)	T	-
AMBA-4-W-GF	17 / 137	U	1.17	23.9	**89.8%** (17 / 14)	0.71	**82.4%** (3 / 14)
AMBA-4-W-GT	17 / 137	U	0.86	12.5	**92.0%** (17 / 11)	0.29	**88.2%** (2 / 11)
AMBA-4-WO-AF	16 / 136	U	5.03	163	**92.6%** (16 / 10)	0.75	**100%** (0 / 10)
AMBA-5	20 / 164	R	525	T	-	-	-
AMBA-5-W-GF	20 / 165	U	19.7	188	**92.7%** (20 / 12)	0.50	**85.0%** (3 / 12)
AMBA-5-W-GT	20 / 165	U	11.6	70.1	**93.9%** (20 / 10)	0.26	**90.0%** (2 / 10)
AMBA-5-WO-AF	19 / 164	U	14.9	126	**93.9%** (19 / 10)	0.80	**100%** (0 / 10)
GENBUF-5	28 / 81	R	0.15	1.23	**46.4%** (15 / 81)	39.2	**54.3%** (15 / 37)
GENBUF-5-W-GF	28 / 82	U	0.15	2.38	**87.8%** (28 / 10)	0.60	**88.3%** (3 / 10)
GENBUF-5-W-GT	28 / 82	U	0.22	3.25	**86.6%** (28 / 11)	0.75	**82.1%** (5 / 11)
GENBUF-5-WO-AF	27 / 81	U	0.12	1.48	**87.7%** (27 / 10)	0.63	**96.3%** (1 / 10)
GENBUF-10	43 / 152	R	1.22	12.3	**53.5%** (20 / 152)	522	**62.5%** (20 / 57)
GENBUF-10-W-GF	43 / 153	U	1.26	29.3	**90.2%** (43 / 15)	3.34	**93.0%** (3 / 15)
GENBUF-10-W-GT	43 / 153	U	4.53	56.1	**89.5%** (43 / 16)	3.81	**88.4%** (5 / 16)
GENBUF-10-WO-AF	42 / 152	U	0.44	9.60	**93.4%** (42 / 10)	1.74	**97.6%** (1 / 10)
GENBUF-20	73 / 368	R	3.65	90.7	**58.9%** (30 / 368)	M	-
GENBUF-20-W-GF	73 / 369	U	3.51	470	**93.2%** (73 / 25)	35.5	**95.9%** (3 / 25)
GENBUF-20-W-GT	73 / 369	U	1328	T	-	-	-
GENBUF-20-WO-AF	72 / 368	U	2.21	115	**97.3%** (72 / 10)	7.78	**98.6%** (1 / 10)
GENBUF-30	103 / 683	R	24.4	920	**61.2%** (40 / 683)	M	-
GENBUF-30-W-GF	103 / 684	U	23.9	T	-	-	-
GENBUF-30-W-GT	103 / 684	U	T	T	-	-	-
GENBUF-30-WO-AF	102 / 683	U	7.61	842	**98.5%** (102 / 10)	22.7	**99.0%** (1 / 10)

incurred a significant performance penalty in [13,9], too. For the explicit algorithms the execution time results are considerably better. Table 1 reports all the results obtained with explicit algorithms.

The first column of Table 1 indicates the name of the specification. The original specifications have names AMBA-n and GENBUF-n, where n is the number of components of the described system. The modified ones have suffixes W-GF, W-GT, and WO-AF as explained above. The following three columns list the size (the number of assumptions and guarantees), the realizability and the time in seconds of checking the realizability of a specification. The fifth column is the time required to remove unhelpful guarantees from an unrealizable specification or unhelpful assumptions from a realizable one.

The sixth column shows the percentage of corresponding player's constraints that have been removed and the new size of the specification. The last two columns are similar to the previous two columns but dedicated to the removal of unhelpful constraints of the remaining player.

The experiments show that a considerable number of constraints can be removed from the explanations. For example, for unrealizable specifications the cause of unrealizability is found to be among only 9% (on average) of guarantees. Moreover, removing 92% (on average) of assumptions does not change the realizability of the obtained guarantees or any of their subsets. Thus before trying to understand and fix the problem a designer can decrease the size of a specification more than 10 times thereby decreasing the effort required to detect and comprehend a bug.

For the real-life realizable specifications ARM AMBA AHB Arbiter and IBM GenBuf Controller we found that about 64% of the assumptions are not required for the realizability of the guarantees. This may indicate that the designers over-constrained the environment in fear that the specification may become unrealizable at some state of the design development. Another possible reason is that not all intended guarantees have been added to the specification. In any case showing unnecessary assumptions can be a valuable debugging information for designers. In fact, our approach unexpectedly shows that going from AMBA-2 to AMBA-3 the number of required assumptions decreases from 4 to 2. The analysis of the generated core allowed us to detect a missing constraint in the AMBA-3 aiming to forbid one assignment to the two Boolean signals used to encode a three value variable. (See [8] for additional details.)

In our experiments the first step of explanation computation is on average 20 times slower than the realizability checking of the original specification. The second step is about 25 times slower than the original realizability checking. Though the time required for computation is relatively large, it is not exceedingly large and is very likely to be a good trade-off by potentially decreasing the time required for debugging a specification.

6 Conclusions and Future Works

In this paper we addressed the problem of providing diagnostic information in presence of formal analysis of requirements, and in particular in the case of realizability. We showed that the problem is nontrivial, formally characterized it, and proposed methods to automatically extract explanations, i.e., descriptions of the reasons for (un)realizability. The experimental evaluation shows the potential of the approach.

It is worth noticing that, most of the concepts and algorithms developed in this paper easily extend beyond realizability: given any computable Boolean-valued function r on a couple of finite sets $\langle A, G\rangle$ such that r has the monotonicity properties stated in Sect. 2.2, the definitions, theorems, and algorithms developed in Sect. 3 and 4.1 apply.

In the future, we plan to evaluate the integration of the explicit and the symbolic methods. We will investigate the use of heuristic search in the space subsets of the specification. We will also investigate the integration within a practical framework (e.g., contract-based design) where realizability comes into play (e.g., by composition of contracts). Finally, given that the use of activation variables has proved to be both a

powerful and expensive means to extract minimum cores for several problem classes involving temporal logic, a separate investigation of how to improve this technique (e.g., by carrying over results from the Boolean domain) seems worthwhile to us.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. European Railway Agency. Feasibility study for the formal specification of ETCS functions. Sep, Invitation to tender (2007), `http://www.era.europa.eu`
3. Behrmann, G., et al.: UPPAAL-Tiga: Time for playing games! In: Damm and Hermanns [11], pp. 121–125.
4. Bloem, R., et al.: RAT: Formal analysis of requirements. In: Damm and Hermanns [11], pp. 263–267.
5. Bloem, R., et al.: Interactive presentation: Automatic hardware synthesis from specifications: A case study. In: Lauwereins, R., Madsen, J. (eds.) DATE, pp. 1188–1193. ACM Press, New York (2007)
6. Bontemps, Y., Schobbens, P., Löding, C.: Synthesis of open reactive systems from scenario-based specifications. Fundam. Inform. 62(2), 139–169 (2004)
7. Clarke, E., et al.: NuSMV: A new symbolic model verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
8. Cimatti, A., et al.: Diagnostic information for realizability. Technical Report FBK-092007-01, Fondazione Bruno Kessler (2007), `http://es.fbk.eu/people/roveri/tests/vmcai08`
9. Cimatti, A., et al.: Boolean abstraction for temporal logic satisfiability. In: Damm and Hermanns [11], pp. 532–546
10. Clarke, E., Veith, H.: Counterexamples Revisited: Principles, Algorithms, Applications. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 208–224. Springer, Heidelberg (2004)
11. Damm, W., Hermanns, H. (eds.): CAV 2007. LNCS, vol. 4590. Springer, Heidelberg (2007)
12. Groce, A.: Error Explanation and Fault Localization with Distance Metrics. PhD thesis, Carnegie Mellon University (2005)
13. Lynce, I., Marques Silva, J.: On computing minimum unsatisfiable cores. In: SAT (2004)
14. Pill, I., et al.: Formal analysis of hardware requirements. In: Sentovich, E. (ed.) DAC, pp. 821–826. ACM Press, New York (2006)
15. Pnueli, A., Piterman, N., Sa'ar, Y.: Synthesis of Reactive(1) Designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)
16. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: 16th Annual ACM Symposium on Principles of Programming Languages, pp. 179–190 (1989)
17. Safra, S.: On the complexity of omega-automata. In: FOCS, pp. 319–327. IEEE, Los Alamitos (1988)
18. Tripakis, S., Altisen, K.: On-the-Fly Controller Synthesis for Discrete and Dense-Time Systems. In: Wing, J.M., Woodcock, J.C.P., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, Springer, Heidelberg (1999)
19. `http://www.prosyd.org`
20. Yoshiura, N.: Finding the causes of unrealizability of reactive system formal specifications. In: SEFM, pp. 34–43. IEEE Computer Society Press, Los Alamitos (2004)