Formal Methods in System Design (2021) 57:473–495
https://doi.org/10.1007/s10703-021-00381-5

Vacuity in synthesis

Roderick Bloem[1] · Hana Chockler[2] · Masoud Ebrahimi[1] · Ofer Strichman[3]

Received: 8 July 2019 / Accepted: 11 August 2021 / Published online: 17 September 2021

Abstract
In reactive synthesis, one begins with a temporal specification φ, and automatically synthesizes a system M such that $M \models \varphi$. As many systems can satisfy a given specification, it is natural to seek ways to force the synthesis tool to synthesize systems that are of a higher quality, in some well-defined sense. In this article we focus on a well-known measure of the way in which a system satisfies its specification, namely *vacuity*. Our conjecture is that if the synthesized system M satisfies φ *non-vacuously*, then M is likely to be closer to the user's intent, because it satisfies φ in a more "meaningful" way. Narrowing the gap between the formal specification and the designer's intent in this way, automatically, is the topic of this article. Specifically, we propose a *bounded synthesis* method for achieving this goal. The notion of vacuity as defined in the context of model checking, however, is not necessarily refined enough for the purpose of synthesis. Hence, even when the synthesized system is technically non-vacuous, there are yet more interesting (equivalently, less vacuous) systems, and we would like to be able to synthesize them. To that end, we cope with the problem of synthesizing a system that is *as non-vacuous as possible*, given that the set of interesting behaviours with respect to a given specification induce a partial order on transition systems. On the theoretical side we show examples of specifications for which there is a single maximal element in the partial order (i.e., the most interesting system), a set of equivalent maximal elements, or a number of incomparable maximal elements. We also show examples of specifications that induce infinite chains of increasingly interesting systems. These results have implications on how non-vacuous the synthesized system can be. We implemented the new procedure in our synthesis tool PARTY. For this purpose we added to it the capability to synthesize a system based on a property which is a conjunction of universal and existential LTL formulas.

Keywords Reactive synthesis · Vacuity · Bounded synthesis

This work was supported by the Austrian Research Promotion Agency (FFG) through project TRUSTED (867558), Graz University of Technology's LEAD project "Dependable Internet of Things in Adverse Environments".

Extended author information available on the last page of the article

1 Introduction

Given a temporal specification φ, the goal of *reactive synthesis* [9,17] is to build a transition system M such that $M \models \varphi$. The motivation of synthesis is clear: rather than building a design and then checking whether it adheres to the specification, focus on the specification alone, and generate automatically a design that satisfies it. In recent years, the theory and especially the tools for synthesis have made significant progress [11].

As many systems can satisfy a given specification, it is natural to seek ways to force the synthesis tool to synthesize systems that are of a higher quality, in some well-defined sense. In this article we focus on a well-known measure of the way in which a system satisfies its specification, namely *vacuity* [2]. It is a standard concept in model-checking, for detecting errors in the specification itself, the model, or both. So far it has not been used in the context of synthesis, however. Our conjecture is that if the synthesized system M satisfies φ *non-vacuously*, then M is likely to be closer to the user's intent, because it satisfies φ in a more "meaningful" way. Narrowing the gap between the formal specification and the designer's intent in this way, automatically, is the topic of this article.

Consider, for example, the property

$$\varphi = \mathrm{G}(req \rightarrow \mathrm{F}grant). \tag{1}$$

A system M with one state satisfying $grant$ (regardless of req) satisfies φ, and is indeed a legitimate outcome of synthesising (1). However, M also satisfies stronger properties such as $\mathrm{GF}grant$. When a system satisfies a property regardless of some of its subformulas, as in this example where the behavior of req is immaterial for the satisfaction of φ, the specification is said to be satisfied *vacuously* (see below a formal definition).

It is not likely that M captures the user's intent: the intent is probably that the system also permits a path π in which there are no grants from a point in which there are no requests. Such a path is called an *interesting witness* [2]: it is an execution that demonstrates non-vacuous satisfaction of the original property. However, the requirement that there *exists* an interesting witness *cannot* be expressed in LTL, since it implicitly relies on an existential quantifier.

Previous work addressing the lack of expressibility of LTL in specifying high-quality systems suggested the extension to quantitative specifications, in order to make it easier to specify certain properties [4], and to be able to synthesize systems that are robust against environment errors, even if the way to react to such errors has not been specified explicitly [3,18].

There are multiple definitions of vacuity in the literature [1,2,6,7,14,15]; the vast majority of them is based on the concept of strengthening the specification by changing a part of it and checking whether the strengthened version is still satisfied in the system; if not, then the system is considered to satisfy the specification non-vacuously. While the general method that we will describe in this paper is orthogonal to the chosen definition as long as the analysed formulas are still in LTL, for ease of exposition we will choose one definition and use it throughout the paper. Most commercially used vacuity-detection tools use the generalised definition by Kupferman and Vardi [14], which is what we will follow here: Let ψ be a subformula in φ. The strengthening of φ with respect to ψ is $\varphi[\psi \leftarrow \bot]$.[1] If $M \models \varphi[\psi \leftarrow \bot]$ then ψ is irrelevant for the satisfaction of φ in M, and we say that φ is satisfied in M vacuously with respect to ψ. It follows that M satisfies φ non-vacuously with respect to ψ iff $M \models \mathrm{E}\neg\varphi[\psi \leftarrow \bot]$. As shown in [14], it is sufficient to consider

[1] This means that we swap ψ with falseif ψ is in positive polarity, and with trueotherwise. Hence, e.g., if $\varphi \equiv \psi_1 \Rightarrow \Psi_2$, then $\varphi[\psi_1 \leftarrow \bot] \equiv \psi_2$.

strengthenings of φ with respect to atomic propositions (literals, in fact) rather than all subformulas. We note that the definitions of vacuity in the literature, including [14], did not consider the division of the atomic propositions into inputs and outputs, as such division is immaterial in model-checking. As we argue later, in synthesis this division is in fact important.

Our synthesis method requires systems with at least one interesting witness for every possible strengthening of φ. More formally, if φ is a specification in LTL, a model M satisfies φ non vacuously if it satisfies a formula in a simple fragment of CTL* consisting of a conjunction of universal and existential formulas:

$$(M \models \mathrm{A}\varphi) \wedge \bigwedge_{\psi \in \mathrm{Lit}(\varphi)} (M \models \mathrm{E}\neg\varphi[\psi \leftarrow \bot]) \;, \tag{2}$$

where Lit (φ) denotes the literals of φ. One of the contributions of this article is the extension of the *bounded synthesis* [10] algorithm to handle this fragment, based on a new ranking function (the original bounded synthesis algorithm handles only universal formulas).

Even when the system satisfies the specification non-vacuously, our tool is capable of improving it by synthesizing a system that has additional interesting witnesses. The users decides when the system reflects their intent. In Sect. 3 we define a *partial order of vacuity on transition systems*, stating that system M' is less vacuous than M if it contains all of the interesting witnesses permitted by M and at least one more. This condition can be stated as a formula in the same fragment of CTL* mentioned above. We describe a synthesis procedure for generating increasingly non-vacuous systems for a given number of states.

In Sect. 4, we revisit the *partial order of vacuity* and prove that this partial order can be mapped to a partial order on subsets of input traces. This allows us to formally introduce the concept of equivalent systems w.r.t. their degree of vacuity and discuss cases where there is a single top element to the partial order of vacuity, multiple equivalent top elements, or multiple incomparable top elements. This is also the main addition to the paper compared to its conference version [5].

We have implemented the non-vacuous bounded synthesis algorithm on top of the PARTY synthesizer [13], which is available for download.[2] Given the informal goal we stated ("capturing the user's intent") naturally it is difficult to prove that our approach works, especially since there are no users in the industry that specify real system for the purpose of synthesis. Our experiments were based, then, on starting from previously published complete specifications, removing parts of them, and activating non-vacuous synthesis. In our experiments, which we describe in Sect. 5, the removed parts of the specification were compensated by our tool. In fact, the generated models not only satisfy the original, complete specifications, but they also realize them less vacuously.

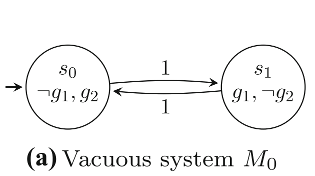
Motivating example

The following example will be used in the rest of the article as a running example. It is a specification of an arbiter with two types of requests and two types of grants (i. e., φ_1 and φ_2) and a mutual exclusion between the grants (i. e., φ_3). The specification φ is a conjunction of the following three properties:

$$\varphi_1 = \mathrm{G}(r_1 \rightarrow \mathrm{F}g_1) \;, \quad \varphi_2 = \mathrm{G}(r_2 \rightarrow \mathrm{F}g_2) \;, \quad \varphi_3 = \mathrm{G}(\neg(g_1 \wedge g_2)) \;, \tag{3}$$

where r_1 and r_2 are inputs (the 'requests') and g_1 and g_2 are outputs (the 'grants'). The smallest system M_0 satisfying φ, synthesised by our tool, is depicted in Fig. 1a. It consists of

[2] https://extgit.iaik.tugraz.at/scos/scos.sources/party-eris.



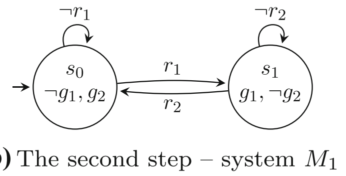


Fig. 1 Systems of the running example

two states, s_0 and s_1, where in each state exactly one of the grants is up. It is easy to see that M_0 satisfies φ vacuously. In particular $M_0 \models \varphi_1[r_1 \leftarrow \bot]$ and $M_0 \models \varphi_2[r_2 \leftarrow \bot]$, where the $\bot$ value for r_1 and r_2 is true in both φ_1 and φ_2, respectively.

The system generated by our tool in the next step is M_1, depicted in Fig. 1b.[3] This system satisfies φ non-vacuously in all its subformulas. Indeed:

1. $M_1 \not\models \varphi_1[r_1 \leftarrow \bot]$, as the path $\pi_1 = s_0^\omega$ corresponding to the output trace $\{\neg g_1, g_2\}^\omega$ falsifies $\mathrm{GF}g_1$;
2. $M_1 \not\models \varphi_2[r_2 \leftarrow \bot]$, as the path $\pi_2 = s_0 s_1^\omega$ corresponding to the output trace $\{\neg g_1, g_2\}, \{g_1, \neg g_2\}^\omega$ falsifies $\mathrm{GF}g_2$;
3. The formulas obtained by replacing one of the grants with false are unrealisable, i. e., there is no system that can satisfy, for example, $\mathrm{G}(\neg r_1)$, because we have no control over the inputs.

□

2 Preliminaries

2.1 Labeled transition systems

For the remainder of the paper, let us fix an input alphabet I and a disjoint output alphabet O, and let us define $\mathrm{AP} = I \cup O$, $\Upsilon = \mathrm{P}(I)$, $\Sigma = \mathrm{P}(O)$, and $\Gamma = \mathrm{P}(\mathrm{AP})$.

Definition 1 (*Labeled Transition System*) A finite, Σ-labeled Υ-transition system is a tuple $M = (S, s_0, \tau, o)$, where S is nonempty set of states, $s_0 \in S$ is the initial state, $\tau : S \times \Upsilon \to S$ is a transition function, and $o : S \to \Sigma$ is a labelling function.

Definition 2 (*An LTS word*) A *word* p of a labeled transition system M, is a sequence $(s_0, \upsilon_0, \sigma_0), (s_1, \upsilon_1, \sigma_1), \ldots$ in $(S \times \Upsilon \times \Sigma)^\omega$ such that $\sigma_i = o(s_i)$ and $s_{i+1} = \tau(s_i, \upsilon_i)$.

Definition 3 (*Path*) A *path* of a word p is the sequence of states induced by p.

Definition 4 (*Trace*) A *trace* of a word p is the sequence of input/output pairs induced by p.

We denote by traces (M) the set of all traces of M. The projection of a trace $\pi \in$ traces (M) on Υ^ω (resp. Σ^ω) is an *input (resp. output) trace* denoted by $\upsilon = (\pi \downharpoonright \Upsilon)$ (resp. $\sigma = (\pi \downharpoonright \Sigma)$). Similarly, for an input trace $\upsilon \in \Upsilon^\omega$, we denote by $M(\upsilon)$ the (unique) trace $\pi \in$ traces (M) s. t. $\upsilon = (\pi \downharpoonright \Upsilon)$.

[3] Note that these are Moore machines. As such the output of a transition leaving a state appears in the state itself. Hence, for example, in the trace $\{g_2, r_1\}, \{g_1, r_1, r_2\}, \{g_2\}^\omega$, the request r_1 on the outgoing edge of s_1 is granted by the label g_1 in s_1.

Definition 5 (*Parallel Composition*) Given input trace $\upsilon \in \Upsilon^\omega$ and output trace $\sigma \in \Sigma^\omega$, their parallel composition denoted by $(\upsilon \parallel \sigma)$ is an infinite word $\omega = \upsilon_0 \cup \sigma_0, \upsilon_1 \cup \sigma_1, \dots$ over Γ.

2.2 Temporal logic

Throughout the paper, we denote by φ an LTL formula in negation normal form (NNF), over the set AP of atomic propositions [16]. The semantics of LTL is defined over AP with respect to infinite paths of M in a standard way. In this paper, we synthesize systems that satisfy the following simple fragment of CTL*:

$$\Phi ::= \mathrm{A}\varphi \mid \mathrm{E}\varphi \mid \Phi \wedge \Phi, \tag{4}$$

where φ is an LTL formula. The semantics of the universal and existential quantifiers over LTL formulas are defined as expected:

So far we denoted by traces (M) the set of traces of M, where a trace corresponds to a run, which in itself must start from an initial state of M (see Definition 2 and 4). We similarly use traces (M, s) to refer to runs that begin at an arbitrary state $s \in S$ of M.

Definition 6 For a state s of a transition system M,

$$s \models \mathrm{A}\varphi \quad \textit{iff} \quad \forall t \in \mathrm{traces}\,(M, s)\,.\; t \models \varphi \;,$$
$$s \models \mathrm{E}\varphi \quad \textit{iff} \quad \exists t \in \mathrm{traces}\,(M, s)\,.\; t \models \varphi \;.$$

A transition system M satisfies a formula φ, written $M \models \varphi$, if all its initial states do.

2.3 Nondeterministic Büchi automata

An LTL formula can be represented by a nondeterministic Büchi automaton [19]: a tuple $\mathcal{A} = (Q, q_0, \rho, \alpha)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\rho : Q \times \Upsilon \times \Sigma \rightarrow \mathrm{P}(Q)$ is the transition relation, and α is the set of accepting states; recall Σ and Υ are defined in Sect. 2.1.

Definition 7 (*Run of a* Büchi *automaton*) Given an infinite word $\omega = \upsilon_0 \cup \sigma_0, \upsilon_1 \cup \sigma_1, \dots$ over Γ, The corresponding *runs* of a Büchi automaton $\mathcal{A}$, denoted by run (w), are the infinite paths $\pi = q_0, q_1, \dots \in Q^\omega$ where for all $i \geq 0$, $q_{i+1} \in \rho(q_i, \upsilon^i, \sigma^i)$.

Definition 8 (*Accepting run*) An *accepting run* of $\mathcal{A}$ is a run that visits some accepting state infinitely often; a trace is accepted by $\mathcal{A}$ if it has a corresponding accepting run, and the language of $\mathcal{A}$ is the set of all accepted traces.

From this point forward, we denote by $\mathcal{A}_\varphi$ the nondeterministic Büchi automata that accepts exactly the traces that satisfy φ.

2.4 Vacuity detection

Informally speaking, a transition system M satisfies a property φ *vacuously* if not all parts of φ are instrumental for the satisfaction of φ in M (in other words, M satisfies φ in an uninteresting way). As proved in [14], it is sufficient to check vacuity with respect to atomic propositions of φ, which, in case of an atomic proposition p appearing in φ in a pure polarity, amounts to model-checking $\varphi[p \leftarrow \bot]$, that is, φ where p is replaced with its bottom value.

Hence, we use the following definition of vacuity that allows for efficient detection algorithm:

Definition 9 (*Vacuity* [2,14]) A transition system M satisfies an LTL property φ *vacuously* iff $M \models \varphi$ and there exists a literal ψ of φ which is of pure polarity and $M \models \varphi[\psi \leftarrow \bot]$.

The formula $\varphi[\psi \leftarrow \bot]$ is a *strengthening* of φ since $\varphi[\psi \leftarrow \bot] \implies \varphi$ and we call the negation $\varphi_\psi = \neg\varphi[\psi \leftarrow \bot]$ of a strengthening a *witness formula*. A trace π of M that satisfies φ_ψ is called an *interesting witness* for ψ, since it demonstrates that ψ is instrumental to the satisfaction of φ in M; π is an *interesting witness* of M if it is an interesting witness for some subformula ψ of φ. We note that if $M \models \bigwedge_i \varphi_i$ then $M \not\models \bigvee_i \varphi_i[\psi \leftarrow \bot]$ is relevant to check only for those φ_i in which ψ appears.

The concept of witnesses and strengthenings is not restricted to Definition 9, and it lends itself, in theory, to other definitions of vacuity [1,7,8]. The framework proposed in this paper is orthogonal to the particular definition of vacuity, as long as the strengthenings are ω-regular.

2.5 Bounded synthesis

Bounded synthesis is a method to construct a finite-state labeled transition system that not only satisfies a given temporal specification φ, but also fulfills a constraint on its size [10]. The idea is to let an SMT solver synthesize a transition system M (i.e., choose the transitions and the labeling of the states), such that $M \times \mathcal{A}_{\neg\varphi}$ has an empty language.

The synchronous product $\mathcal{G}$ of a transition system $M = (S, s_0, \tau, o)$ and a Büchi automaton $\mathcal{A}_{\neg\varphi} = (Q, q_0, \rho, \alpha)$ is called the *run graph* of $\mathcal{A}_{\neg\varphi}$ on M.[4] The states of $\mathcal{G}$ are annotated with two functions: a reachability function $\lambda^{\mathbb{B}} : Q \times S \to \mathbb{B}$ and a ranking function $\lambda^{\#} : Q \times S \to C \subset \mathbb{N}$, where $C = \{0, \ldots, |Q| \times |S| - 1\}$. Annotations of $\mathcal{G}$ (i.e., $\lambda^{\#}$ and $\lambda^{\mathbb{B}}$ functions) are valid if they satisfy the following constraints. First, the initial state is reachable:

$$\lambda^{\mathbb{B}}(q_0, s_0) \ . \tag{5}$$

Second, the reachability predicate and the transition system are compatible:

$$\bigwedge_{\substack{q,q' \in Q \\ s,s' \in S \\ \upsilon \in \Upsilon}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), \upsilon) \wedge s' \in \tau(s, \upsilon) \to \lambda^{\mathbb{B}}(q', s') \ . \tag{6}$$

Finally, the ranking function guarantees that the constraint is satisfiable only if the language of the run graph is empty: For *accepting* states, we require that the labelling on the target state is strictly larger than on the source (accepting) state:

$$\bigwedge_{\substack{q \in \alpha, q' \in Q \\ s,s' \in S \\ \upsilon \in \Upsilon}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), \upsilon) \wedge s' \in \tau(s, \upsilon) \to \lambda^{\#}(q', s') > \lambda^{\#}(q, s) \ ; \tag{7}$$

and for *non-accepting* states the labelling on the target states is larger or equal than on the source state:

$$\bigwedge_{\substack{q \in Q \setminus \alpha, q' \in Q \\ s,s' \in S \\ \upsilon \in \Upsilon}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), \upsilon) \wedge s' \in \tau(s, \upsilon) \to \lambda^{\#}(q', s') \geq \lambda^{\#}(q, s) \ . \tag{8}$$

[4] Since $\mathcal{G}$ is only used for checking emptiness, the labels are immaterial, and it is customary to use a one-letter automaton (i.e., $|\Sigma| = |\Upsilon| = 1$).

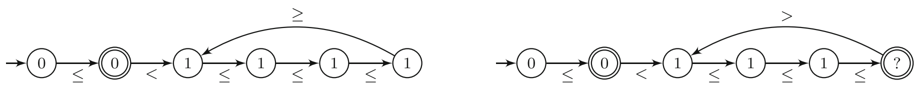


Fig. 2 We can assign a number to each state on the left automaton, that satisfies the inequality constraints, e.g., the 0/1 values labeling the states. Such a labeling is impossible for the automaton on the right, because it has an accepting state in a loop

The intuition behind the ranking function is as follows: if the language is not empty, then there is an accepting path (i.e., a lasso-shaped path in the product automaton that includes an accepting state), and then it is impossible to satisfy these constraints over that path. This is because the ranks of states on the cycle cannot be strictly descending. The two automata in Fig. 2 illustrate this point—see caption. Hence, (5)–(8) are satisfiable if and only if the language of the product automaton is empty. The correctness of this construction was proven in [10].

Theorem 1 (Finkbeiner et al. [10]) *Given a Büchi automaton $\mathcal{A} = (Q, q_0, \rho, \alpha)$ constructed from $\neg\varphi$, transition system $M = (S, s_0, \tau, o)$ satisfies $A\varphi$ iff it corresponds to a solution to the constraints* (5)–(8).

Initially, the LTL specification φ is negated and translated to a Büchi automaton $\mathcal{A}_{\neg\varphi}$. In the next step, (5)–(8) are solved with an SMT solver based on $\mathcal{A}_{\neg\varphi}$. Being unknown, τ, $\lambda^{\mathbb{B}}$, $\lambda^{\#}$ and o (the labeling function) are represented by uninterpreted functions; thus, the quest for finding M is reduced to the problem of satisfiability modulo finite integer arithmetic with uninterpreted functions, which is an NP-complete problem.

3 Non-vacuous bounded synthesis

In this section we describe *non-vacuous bounded synthesis*—our method for constructing a finite-state labeled transition system that fulfils a constraint on its size and satisfies a given temporal specification non-vacuously.

3.1 A specification for non-vacuous satisfaction

A specification φ is satisfied non-vacuously in M if and only if M contains a witness for each strengthening of φ. In other words, as we stated earlier in (2),

$$M \models A\varphi \wedge \bigwedge_{\psi \in \mathrm{Lit}(\varphi)} E\neg\varphi[\psi \leftarrow \bot] \tag{9}$$

(note that (9) is based on our choice of definition for vacuity—see 9). We call $\varphi_\psi = \neg\varphi[\psi \leftarrow \bot]$ the *witness formulas* for non-vacuity of φ with respect to ψ.

Note that not all witness formulas add usable information. For instance, for φ as defined in (3), the witness formula φ_{g_1} (i.e., $\neg\varphi_1[g_1 \leftarrow \bot] = \mathrm{F}r_1$) is clearly satisfied by a trace of any system, and the same holds for any satisfiable witness formula that contains only input signals.

We continue in the next subsection by showing how existentially-quantified formulas can be synthesized. Then, we can use this technique to synthesise formulas of the form defined in (9).

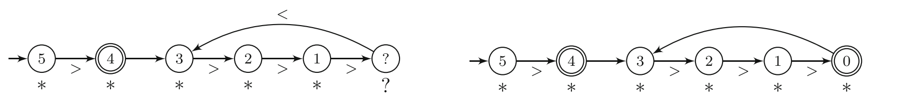


Fig. 3 On the left there is no accepting run, and indeed there is no ranking function that can satisfy the constraints. On the right there is an accepting run (the λ^* predicate is marked with '*'), and the fact that there is no constraint on the outgoing edge of the accepting state allows to find a ranking function, namely the numbers 0,1,2,3,4,5 that are marked inside the states

3.2 Bounded synthesis for existential formulas

Our goal is to synthesize a finite-state labeled transition system with a bound on its size, in which there *exists* an execution path that satisfies a given temporal specification φ. We will define a set of constraints that is different than the case described in Sect. 2.5 to achieve this. Initially, we translate φ to a nondeterministic Büchi automaton $\mathcal{A}_\varphi$ and create the run graph $\mathcal{G}$ of $\mathcal{A}_\varphi$ on M. Then, we use a Boolean marking function $\lambda^* : Q \times S \to \mathbb{B}$ to indicate that a state is on our selected path in $\mathcal{G}$. On that selected path, we impose a ranking function that can only be satisfied if it corresponds to an accepting run.

First, the initial state is marked:

$$\lambda^*(q_0, s_0)\ . \tag{10}$$

Next, if a *non-accepting* state is marked, then at least one of its successors is marked, and the ranking of the destination state is strictly smaller:

$$\bigwedge_{\substack{q\in Q\setminus\alpha\\ s\in S\\ \upsilon\in\Upsilon}} \left(\lambda^*(q,s) \to \bigvee_{\substack{q'\in Q\\ s'\in S}} \begin{pmatrix} q' \in \rho(q, o(s), \upsilon) \wedge s' \in \tau(s,\upsilon) \wedge \\ \lambda^*(q',s') \wedge \lambda^\#(q',s') < \lambda^\#(q,s) \end{pmatrix} \right) . \tag{11}$$

On the other hand if an *accepting state* is marked, then we only require that one of its successors is marked (but in contrast to the previous case, here there is no restriction on the ranking of its successor):

$$\bigwedge_{\substack{q\in \alpha\\ s\in S\\ \upsilon\in\Upsilon}} \left(\lambda^*(q,s) \to \bigvee_{\substack{q'\in Q\\ s'\in S}} \begin{pmatrix} q' \in \rho(q, o(s), \upsilon) \wedge s' \in \tau(s,\upsilon) \wedge \\ \lambda^*(q',s') \end{pmatrix} \right) . \tag{12}$$

The two automata in Fig. 3 illustrate our construction—see caption. The following theorem states that these constraints are correct.

Theorem 2 *Given a Büchi automaton $\mathcal{A} = (Q, q_0, \rho, \alpha)$ constructed from a formula φ', a transition system $M = (S, s_0, \tau, o)$ satisfies $E\varphi'$ iff it corresponds to a solution to constraints* (10)–(12).

Proof ($\Rightarrow$) There is a unique run graph $\mathcal{G} = (G, E)$ for $\mathcal{A}$ on M. Assume M is accepted by $\mathcal{A}$; therefore, $\mathcal{G}$ contains at least one lasso-shaped path $\pi = (q_0, s_0)(q_1, s_1)\ldots[(q_n, s_n)\ldots(q_m, s_m)]^\omega$ such that q_i is accepting for some $i \in [n, m]$. We have to show that in such a case (10)-(12) are satisfiable. Marking all the states on the path clearly satisfies (10), and the λ^* predicate is true along this path as required by constraints (11) and (12). It is left to show that there exists a ranking function that satisfies (11). Indeed the following function, which annotates each state on π by its distance to q_i, is a valid ranking function:

$$\lambda^{\#}(q_j, s_j) = \begin{cases} i - j & \text{if } j \leq i \\ m - j + i - n + 1 & \text{if } i < j. \end{cases}$$

Indeed, $\lambda^{\#}(q_j, s_j) > \lambda^{\#}(q_k, s_k)$ for all $((q_j, s_j)(q_k, s_k)) \in \pi$, unless $j = i$. Recall that only non-accepting states are bound by constraint (11). The figure below demonstrates this

$$\begin{tikzcd} \lambda^{\#}: \arrow[r] & 4 \arrow[r, ">"'] & 3 \arrow[r, ">"'] & 2 \arrow[r, ">"'] & 1 \arrow[r, ">"'] & 0 \arrow[r] & 3 \arrow[llll, bend right, "<"'] \\ j: & 1 & 2 & 3 & 4 & 5 & 6 \\ & & & n & & i & m \end{tikzcd}$$

ranking for $n = 3$, $m = 6$, and $i = 5$.

($\Leftarrow$) Assume that (10)–(12) are satisfiable. The set of marked states must include a lasso-shaped path beginning from the initial state, and the fact that (11) is satisfied means that there exists an accepting state in the loop. Hence the run graph must contain an accepting path. □

Finally, synthesising a non-vacuous system—a system that satisfies (2)—amounts to solving the *conjunction* of the constraints that were described in Sect. 2.5 (for the universal part), and the constraints in Sect. 3.2 for each $\psi \in \text{Lit}(\varphi)$ (for the existential part). A separate discrete ranking function is required for φ and each of its witness formulas.

Corollary 1 *A finite-state transition system $M = (S, s_0, \tau, o)$ satisfies a temporal specification in the form of the CTL* fragment defined in* (4) *iff it corresponds to a solution to constraints* (5)–(8) *and* (10)–(12).

We note that the addition of (10)–(12) to the bounded synthesis formula, does not change the asymptotic complexity of the problem.

4 Beyond vacuity

In the introduction we argued that non-vacuous systems are preferable to vacuous systems because they are more likely to fulfill the designer's intent. This guarantees that for specifications like $\varphi = \text{G}(r \rightarrow \text{F}g)$, there will be at least one path on which GFg does not hold. Intuitively, this corresponds to the idea that an input r should trigger the output g. However, the definition of vacuity is somewhat too coarse for our purpose. We need a more refined notion, which will enable us to distinguish between non-systems that are non-vacuous. In the following subsection we introduce a partial order between systems realizing a given specification. We consider a system A strictly *less vacuous* (or, equivalently, *more interesting*) than another system B if the set of interesting witnesses in A properly contains the set of interesting witnesses in B. For the property above, for example, this corresponds to additional witnesses to $\neg$GFg, i.e., additional traces in which g does not occur without first being 'triggered' by r. For some specifications, there exist *least vacuous* systems, that is, maximal systems in the partial order of vacuity. In Sect. 4.2, however, we show that for some other specifications the partial order gives rise to *infinite vacuity chains*, i.e., infinite chains of ever less vacuous systems.

In Sect. 4.3 we show that given a system, we can use a variant of bounded synthesis to synthesize a less vacuous one, which naturally leads to a most interesting (least-vacuous) system of a given size, when such a system exists. Finally, in Sect. 4.4, we show that for our running example there exists such a least-vacuous system.

4.1 Partial order on non-vacuous systems

Let M_1 and M_2 be transition systems that satisfy φ. Given a witness formula φ_ψ, we define a relation $M_1 \preccurlyeq_\psi M_2$ to indicate that M_2 has at least the same set of interesting witnesses according to φ_ψ as M_1. Formally, given a specification φ and a witness formula φ_ψ of φ, we define

$$M_1 \preccurlyeq_\psi M_2 \text{ iff } \forall \upsilon \in \Upsilon^\omega.\ (M_1(\upsilon) \models \varphi_\psi) \rightarrow (M_2(\upsilon) \models \varphi_\psi). \tag{13}$$

We say that M_2 is *strictly less vacuous* than M_1 if in addition there is at least one input sequence that leads to an interesting witness only in M_2:

$$M_1 \prec_\psi M_2 \text{ iff } M_1 \preccurlyeq_\psi M_2 \text{ and } \exists \upsilon \in \Upsilon^\omega.\ (M_1(\upsilon) \not\models \varphi_\psi) \wedge (M_2(\upsilon) \models \varphi_\psi). \tag{14}$$

If $M_1 \preccurlyeq_\psi M_2$ and $M_2 \preccurlyeq_\psi M_1$, we say that M_1 and M_2 are equivalent in the partial order of vacuity with respect to ψ, denoted $M_1 \equiv_\psi M_2$.

By extending the relation $\prec_\psi$ to the set of all witness formulas, we can compare two transition systems in terms of vacuity. Let Ψ be the set of all witness formulas for φ. We define the *partial order* $\preccurlyeq_\varphi$ as

$$M_1 \preccurlyeq_\varphi M_2 \text{ iff } \forall \varphi_\psi \in \Psi.\ M_1 \preccurlyeq_\psi M_2\ , \tag{15}$$

and the *strict partial order* $\prec_\varphi$ as

$$M_1 \prec_\varphi M_2 \text{ iff } M_1 \preccurlyeq_\varphi M_2 \text{ and } \exists \varphi_\psi \in \Psi.\ \big(M_1 \prec_\psi M_2\big)\,. \tag{16}$$

In other words, M_2 is at least as non-vacuous as M_1 w.r.t. all possible witnesses and is *strictly less vacuous* than M_1 w.r.t. at least one witness formula. Similarly to the above, we say that M_1 is equivalent to M_2 in the partial order of vacuity, denoted $M_1 \equiv_\varphi M_2$, if $M_1 \preccurlyeq_\varphi M_2$ and $M_2 \preccurlyeq_\varphi M_1$.

Since there is a finite number of transition systems of any size N, for a given LTL formula φ, there exists at least one least vacuous system M_N^φ of size N, according to $\prec$, assuming that φ is realizable by a system with N states. This system may not be unique. Moreover, if there are several least-vacuous systems of size N, they can be equivalent or incomparable.

It is easier to reason about equivalent and incomparable systems if we re-define the partial order on systems as follows. We can view the above partial order on systems realizing φ as a partial order on subsets of the (infinite) set of all input traces $\mathcal{V} = \{\upsilon : \upsilon \in \Upsilon^\omega\}$. A system M realizing φ can be mapped to the set of input traces $V(M, \varphi)$ that induce interesting witness traces in M with respect to some subformula of φ. We can similarly denote by $V(M, \varphi_\psi)$ the set of input traces that induce interesting witness traces in M with respect to a particular witness formula φ_ψ. Then, it is easy to see that (13) is equivalent to

$$M_1 \preccurlyeq_\psi M_2 \text{ iff } V(M_1, \varphi_\psi) \subseteq V(M_2, \varphi_\psi), \tag{17}$$

and the strict inequality holds if the set $V(M_1, \varphi_\psi)$ is a strict subset of $V(M_2, \varphi_\psi)$. Also, $M_1 \equiv_\psi M_2$ if $V(M_1, \varphi_\psi) = V(M_2, \varphi_\psi)$.

Similarly, (15) can be rewritten as

$$M_1 \preccurlyeq_\varphi M_2 \text{ iff } \forall \varphi_\psi \in \Psi.\ V(M_1, \varphi_\psi) \subseteq V(M_2, \varphi_\psi), \tag{18}$$

and the strict inequality holds if the set $V(M_1, \varphi_\psi)$ is a strict subset of $V(M_2, \varphi_\psi)$ for some $\varphi_\psi \in \Psi$. Two systems M_1 and M_2 realizing φ are *equivalent* with respect to φ (i.e., $M_1 \equiv_\varphi M_2$) if and only if the following holds:

$$\forall \varphi_\psi \in \Psi.\ V(M_1, \varphi_\psi) = V(M_2, \varphi_\psi). \tag{19}$$

Finally, if $M_1 \not\preccurlyeq_\varphi M_2$ and $M_2 \not\preccurlyeq_\varphi M_1$, we say that M_1 and M_2 are *incomparable*.

The following theorem clarifies the difference between incomparable and equivalent least-vacuous systems using (18) and (19).

Theorem 3 *The following claims hold for any LTL formula φ.*

(a) *If there exists a system M_{max} realizing φ such that*

$$\forall \varphi_\psi \in \Psi.\ V(M_{max}, \varphi_\psi) = \bigcup_{M \text{ realizes } \varphi} V(M, \varphi_\psi), \tag{20}$$

then M_{max} is a least-vacuous system realizing φ, and any other least-vacuous system M' realizing φ is equivalent to M_{max}.

(b) *If φ is realizable and there is no system M_{max} that realizes φ and satisfies (20), then either*

- *there is an infinite vacuity chain; or*
- *there are several incomparable least-vacuous systems.*

Proof We prove the claims separately.

(a) If there exists M_{max} realizing φ such that (20) holds, then by (18), for any other M realizing φ, we have $M \preccurlyeq_\varphi M_{max}$, hence M_{max} is a least-vacuous system realizing φ. Now assume that there is another system M' that is a least-vacuous system realizing φ. Then,

$$\forall \varphi_\psi \in \Psi.\ V(M', \varphi_\psi) \subseteq \bigcup_{M \text{ realizes } \varphi} V(M, \varphi_\psi) \tag{21}$$

(as M' is one of the systems in the union), hence $M' \preccurlyeq_\varphi M_{max}$. Therefore, M' and M_{max} are not incomparable. Since M' is least vacuous, it also holds that $M' \succeq M_{max}$, and hence they are equivalent in the vacuity preorder.

(b) Assume that there is no system that satisfies (20) and let M_1 be a least-vacuous system realizing φ (if there is no such M_1, then there is an infinite vacuity chain, and the claim holds trivially). In particular, since M_1 does not satisfy (20), we have

$$V(M_1, \varphi_\psi) \subset \bigcup_{M \text{ realizes } \varphi} V(M, \varphi_\psi)$$

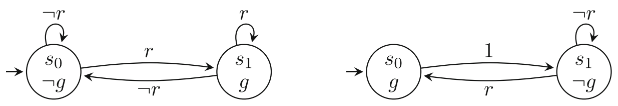
for at least one witness formula φ_ψ of φ. Consider an input trace

$$\upsilon \in \Big(\bigcup_{M \text{ realizes } \varphi} V(M, \varphi_\psi) \Big) \setminus V(M_1, \varphi_\psi).$$

Assume that there exists M_2 realizing φ that contains an interesting witness trace for φ_ψ induced by υ. By construction, M_1 and M_2 are incomparable. Hence, either M_2 is also a least-vacuous system, which is incomparable to M_1 (which concludes the proof), or there exists another system M_3 such that $M_2 \prec_\varphi M_3$ and M_3 is a least-vacuous system realizing φ (recall that we assumed that there are no infinite vacuity chains). Then, M_1 and M_3 are two incomparable least-vacuous systems realizing φ.

□

The following examples illustrate the notion of least-vacuous systems. Example 1 shows a specification for which there exist two *equivalent* least-vacuous systems. Example 2 and the example in "Appendix 1" show specifications for which there are two *incomparable* least-vacuous systems.



(a) M_0 **(b)** M_1

Fig. 4 Least-vacuous systems realizing $\mathrm{G}(r \rightarrow \mathrm{F}g)$

Example 1 [*Equivalent least-vacuous systems*] Consider the following specification:

$$\varphi = \mathrm{G}(r \implies \mathrm{F}g) \ ,$$

where the sole interesting witness formula is $\varphi_r = \mathrm{FG}(\neg g)$. Figure 4 demonstrates two systems of size 2 realizing φ. It is easy to see that

$$V(M_0, \varphi_r) = V(M_1, \varphi_r),$$

that is, all non-vacuous traces of M_0 are still non-vacuous in M_1 and vice-versa. In other words, $M_0 \equiv_\varphi M_1$. Yet, the systems are not isomorphic. As an example of the difference between M_0 and M_1, consider the run of M_0 on the input trace $(\neg r)^\omega$:

$$M_0((\neg r)^\omega) = \{\neg r, \neg g\}^\omega \ .$$

On the other hand, the run of M_1 on the same input trace is:

$$M_1((\neg r)^\omega) = \{\neg r, g\}, \{\neg r, \neg g\}^\omega \ .$$

Proposition 1 *M_0 and M_1 are least-vacuous systems realizing φ.*

Proof As $M_0 \equiv_\varphi M_1$, it suffices to prove that M_0 is a least-vacuous system realizing φ. Let M be an arbitrary system that realizes φ. For an input sequence $\upsilon \in \Upsilon^\omega$, assume that υ induces a path in M that satisfies $\varphi_r = \mathrm{FG}(\neg g)$. Since this path, in particular, satisfies φ, it also satisfies $\mathrm{FG}(\neg r)$ (otherwise there would have been requests that are never granted). Observing Fig. 4, it is easy to see that the same input sequence υ would induce a path in M_0 with an infinite suffix $\{s_0\}^\omega$, hence, in particular, it satisfies $\mathrm{FG}(\neg g)$. Hence, M is not less vacuous than M_0. □

The following is an example of two *incomparable* least-vacuous systems, illustrating the second case in 3(b).

Example 2 [*Incomparable least-vacuous systems*] Consider the specification φ, which is the conjunction of the following three properties:

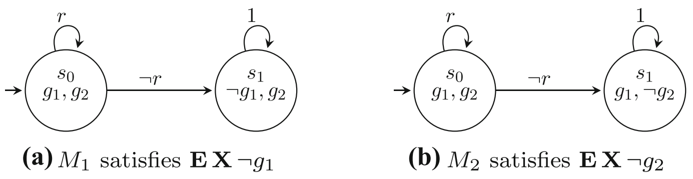
$$\varphi_1 = r \rightarrow \mathrm{X}g_1, \quad \varphi_2 = r \rightarrow \mathrm{X}g_2, \quad \varphi_3 = \mathrm{X}(g_1 \vee g_2) \ , \tag{22}$$

where r is an input, and g_1 and g_2 are outputs. The interesting witness formulas for φ are

$$\neg\varphi_1[r \leftarrow \bot] = \mathrm{X}(\neg g_1), \quad \neg\varphi_2[r \leftarrow \bot] = \mathrm{X}(\neg g_2) \ .$$

In any system that realizes φ, satisfying the witness formula $\mathrm{X}(\neg g_1)$ implies that the second state must be labelled with $\neg g_1 \wedge g_2$ if $\neg r$ is its input (because of φ_3). At the same time in order to satisfy $\mathrm{X}(\neg g_2)$ the second state of this system must be labelled with $g_1 \wedge \neg g_2$, which clearly contradicts the previous requirement.

On the other hand, least-vacuous systems for φ exist. For example, consider the two systems in Fig. 5: the left satisfies $\mathrm{EX}(\neg g_1)$ and the right satisfies $\mathrm{EX}(\neg g_2)$, but neither



r
1
s_0 g_1, g_2
$\neg r$
s_1 $\neg g_1, g_2$
(a) M_1 satisfies $\mathbf{E}\,\mathbf{X}\,\neg g_1$
r
1
s_0 g_1, g_2
$\neg r$
s_1 $g_1, \neg g_2$
(b) M_2 satisfies $\mathbf{E}\,\mathbf{X}\,\neg g_2$

Fig. 5 Both M_1 and M_2 realize φ from (22) least-vacuously, but are incomparable

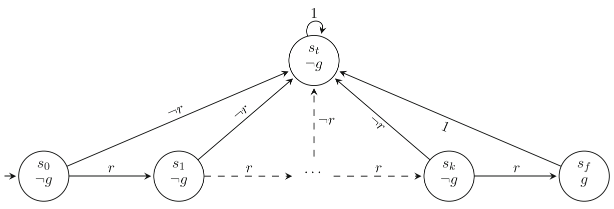


Fig. 6 An example of infinite vacuity chain

satisfies both. Furthermore, it is not hard to see that both M_1 and M_2 are least-vacuous with respect to φ (the proof would be similar to the proof of Proposition 1).

□

4.2 Infinite vacuity chains

For some formulas, there is an infinite chain of ever less vacuous (and ever larger) systems. As an example, consider the following LTL specification:

$$\varphi = (\mathrm{G}r) \implies (\mathrm{F}g) \ . \tag{23}$$

The only useful witness formula for φ is

$$\varphi_r = \mathrm{G}\neg g \ . \tag{24}$$

Figure 6 depicts an abstract transition system M_k of arbitrary size (i. e., $k+3$) that realizes specification φ non-vacuously for any k.

Proposition 2 $\forall k.\ M_k \prec_\psi M_{k+1}$.

Proof We have to show that M_{k+1} is as non-vacuous as M_k and that there exists an input trace that makes M_{k+1} less vacuous w.r.t. φ_r.

First we show $\forall k.\ M_k \preccurlyeq_\psi M_{k+1}$. For each input trace $\pi \in \Upsilon^\omega$, if $M_k(\pi) \models \mathrm{G}\neg g$, then $\pi \models r^j(\neg r)^+(\neg r + r)^\omega$ for some $j \leq k$, so $M_{k+1}(\pi) \models \mathrm{G}\neg g$.

To see that $\forall k.\ M_k \prec_\psi M_{k+1}$ holds, note that the input trace $r^{k+1}(\neg r)^\omega$ leads to an interesting trace in M_{k+1} but not in M_k. □

4.3 Synthesizing a less vacuous system

We now discuss how to synthesize a less vacuous system M_2 given a correct system M_1. We do this by expressing the partial order defined above in the simple fragment of CTL*defined in (4).

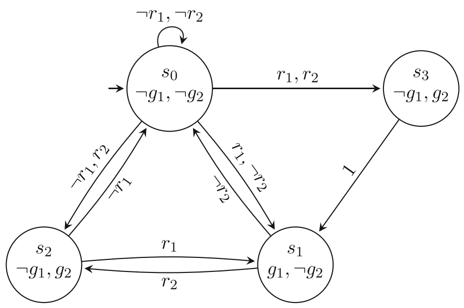


Fig. 7 The final non-vacuous system M_2

Given a formula φ or a system M, we use a primed version (φ' or M', respectively) to denote the formula/system obtained by replacing all output literals by primed versions. Given a system M_1 that satisfies φ, we have $M_1 \prec_\psi M_2$ iff

$$M_1' \times M_2 \models \mathrm{A}\varphi \wedge \mathrm{A}(\varphi_\psi' \to \varphi_\psi) \wedge \mathrm{E}(\neg\varphi_\psi' \wedge \varphi_\psi)\ . \tag{25}$$

Equation (25) follows directly from Eq. (14) after renaming the variables of M_1 with their primed versions. Note that φ and φ_ψ refer to the outputs of M_2 and φ_ψ' refers to the outputs of M_1, while both systems receive the same inputs. The following theorem generalizes (25) to all subformulas of φ.

Theorem 4 *M_1 is strictly less vacuous than M_2 iff*

$$M_1' \times M_2 \models A(\varphi \wedge \bigwedge_{\varphi_\psi \in \Psi} (\varphi_\psi' \to \varphi_\psi)) \wedge E(\bigvee_{\varphi_\psi \in \Psi} (\neg\varphi_\psi' \wedge \varphi_\psi))\ . \tag{26}$$

Note that this equation has the form of Eq. (2) and can thus be solved as described in Sect. 3.

We iteratively synthesize less and less vacuous systems by applying (26) where M_2 is the existing system. The result is M_1', which then becomes M_2 of the next iteration. We repeat this process until (26) becomes unrealizable with the current bound on the number of states. We can then either stop or increase the bound. Every such step adds a linear number of variables. Since the asymptotic complexity has an exponential dependence on the number of variables, eventually this formula may become too hard to solve in practice.

4.4 A least-vacuous system for the running example

Consider once again our running example from the introduction. Figure 7 shows a least-vacuous system M_2 with the bound 4 on the number of states (one of the intermediate iterations resulted in M_1 depicted in Fig. 7).

System M_2 is strictly less vacuous than M_1. Recall that the two witness formulas are $\varphi_{r_1} = \mathrm{FG}\neg g_1$ and $\varphi_{r_2} = \mathrm{FG}\neg g_2$. It is not hard to verify that all interesting paths in M_1 w.r.t. to φ_{r_1} (w.r.t. to φ_{r_2}) are also interesting w.r.t. to φ_{r_1} (w.r.t. to φ_{r_2}, resp.) in M_2. Also, the trace that results from leaving r_1 and r_2 low all the time is interesting w.r.t. φ_{r_2} in M_2 but not in M_1.

Proposition 3 *M_2 is a least-vacuous system with respect to $\{\varphi_{r_1}, \varphi_{r_2}\}$.*

Proof Let M be an arbitrary system that realizes φ. For an input sequence $\upsilon \in \Upsilon^\omega$, assume that υ induces a path in M that satisfies $\varphi_1[r_1 \leftarrow \bot] = \mathrm{FG}\neg g_1$. Since this path, in particular,

Fig. 8 LTL specification for the 'next' arbiter of two clients. Note that the incomplete specification excludes the right-to-left implications in the guarantee

assume
$\mathbf{G}\,\neg(r_1 \wedge r_2)$
guarantee
$\mathbf{G}(r_1 \longleftrightarrow \mathbf{X}\, g_1) \wedge$
$\mathbf{G}(r_2 \longleftrightarrow \mathbf{X}\, g_2) \wedge$
$\mathbf{G}\,\neg(g_1 \wedge g_2)$

(a) Complete Specification

assume
$\mathbf{G}\,\neg(r_1 \wedge r_2)$
guarantee
$\mathbf{G}(r_1 \rightarrow \mathbf{X}\, g_1) \wedge$
$\mathbf{G}(r_2 \rightarrow \mathbf{X}\, g_2) \wedge$
$\mathbf{G}\,\neg(g_1 \wedge g_2)$

(b) Incomplete Specification

satisfies φ, it also satisfies FG$\neg r_1$ (otherwise there would have been requests that are never granted). Observing Fig. 7, it is easy to see that the same input sequence υ would induce a path in M_2 with an infinite suffix $\{s_0, s_2\}^\omega$, hence, in particular, it satisfies FG$\neg g_1$. A similar argument holds for $\varphi_2[r_2 \leftarrow \bot]$. Hence, M is not less vacuous than M_2. □

The question whether a given system is a least-vacuous one (again, such systems may not be unique) is equivalent to asking whether a less vacuous one exists, which, by (26) can be reduced to a CTL*realizability question.

5 Experimental evaluation

We implemented the described technique in the PARTY synthesizer [13]. We cannot check directly whether non-vacuous synthesis leads to a system which is closer to 'the user intent'. However, we can check something with a similar flavor: the ability of this technique to guess missing parts of a specification. Hence, we conducted the following experiment: first, we synthesized models for three complete and correct specifications; then, we made them incomplete by removing some of the conjuncts in the specification; finally, we ran non-vacuous synthesis on these incomplete specifications, and checked whether the resulting system satisfies the *original*, complete specification; A positive answer indicates that non-vacuous synthesis can accelerate the convergence towards the desired system.

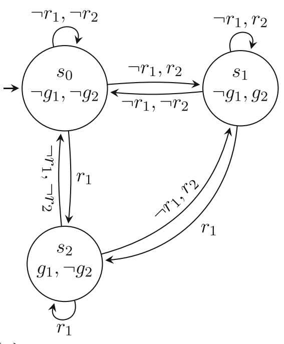
Indeed, in the three experiments that we conducted, non-vacuous synthesis was able to synthesize a system that satisfies the original, full specification, although we emphasize that this is not guaranteed in general. The synthesized system in all three cases is not identical to the one synthesized according to the full specification, which reflects the fact that many systems can satisfy the same specification. It is up to the user to choose between them.

More details about the experimental setting, as well as a performance comparison, is given in 1. It is evident from the comparison that, as expected, on average non-vacuous bounded synthesis takes more time than bounded synthesis.

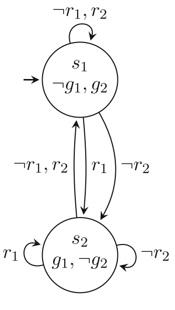
5.1 A 'next' arbiter

The 'next' arbiter of two clients issues a grant for each client in the next step if and only if the client sends a request. The assumption is that clients never send requests simultaneously; thus, issued grants should be mutually exclusive. The complete and incomplete specification of this arbiter for two clients is shown in Fig. 8. The specification should be interpreted as 'every run that satisfies the assume predicates should also satisfy the guarantee predicates'.

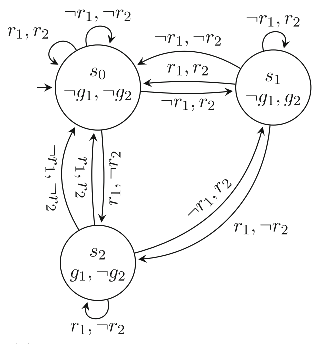
As depicted in Fig. 9a, b, even a slight modification in the specification results in a large gap in the behaviors of the synthesized systems. On the other hand starting from the system depicted in Fig. 9b, three iterations of the non-vacuous synthesis process result in the system shown in Fig. 9c, which satisfies the original, full specification. Figure 10 depicts the runtime



(a) Bounded synthesis of complete specification

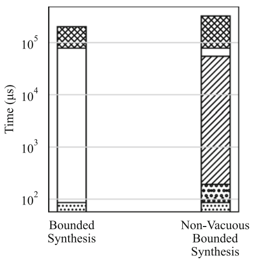


(b) Bounded synthesis of incomplete specification



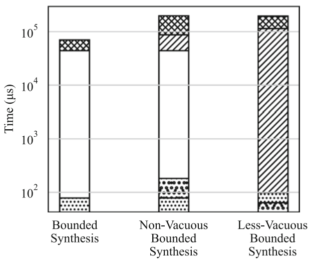
(c) Non-vacuous synthesis of incomplete specification

Fig. 9 Synthesized arbiters of the complete and incomplete specifications of the 'next' arbiter that appeared in Fig. 8



		Bounded Synthesis	Non-Vacuous Bounded Synthesis
☐	Co-Buchi Translation	78360	78360
▨	Buchi Translation	0	54641
░	Co-Buchi Encoding	86	86
●	Buchi Encoding	0	193
▦	SMT Solver	202536	325270
	Sum	280982	458550

(a) Runtime breakdown of the complete specification of the 'next' arbiter. Since the complete specification already gives a least vacuous arbiter, runtime breakdown of less-vacuous bounded synthesis is not applicable.



		Bounded Synthesis	Non-Vacuous Bounded Synthesis	Less-Vacuous Bounded Synthesis
☐	Co-Buchi Translation	43991	43991	195379
▨	Buchi Translation	0	87428	113927
░	Co-Buchi Encoding	78	78	99
●	Buchi Encoding	0	182	64
▦	SMT Solver	69915	198081	193115
	Sum	113985	329760	502583

(b) Runtime breakdown of the incomplete specification of the 'next' arbiter.

Fig. 10 Runtime breakdown of synthesis of the 'next' arbiter

breakdown of bounded synthesis and non-vacuous bounded synthesis for both the complete and incomplete versions of the 'next' arbiter.

5.2 A 'full' arbiter

A 'full' arbiter of two clients eventually issues a grant for each client if the client sends a request. The complete specification appears in Fig. 11 (left), and a partial specification

guarantee

$$\begin{aligned}
&\neg(\neg r_1 \wedge \neg g_1)\,\mathbf{U}(\neg r_1 \wedge g_1) \wedge \\
&\neg(\neg r_2 \wedge \neg g_2)\,\mathbf{U}(\neg r_2 \wedge g_2) \wedge \\
&\neg\,\mathbf{F}(g_1 \wedge X(\neg r_1 \wedge \neg g_1) \wedge \mathbf{X}(\neg r_1 \wedge g_1)\,\mathbf{U}(\neg r_1 \wedge g_1)) \wedge \\
&\neg\,\mathbf{F}(g_2 \wedge X(\neg r_2 \wedge \neg g_2) \wedge \mathbf{X}(\neg r_2 \wedge g_2)\,\mathbf{U}(\neg r_2 \wedge g_2)) \wedge \\
&\mathbf{G}((\neg r_1 \wedge g_1) \rightarrow \mathbf{F}((r_1 \wedge g_1) \vee \neg g_1)) \wedge \\
&\mathbf{G}((\neg r_2 \wedge g_2) \rightarrow \mathbf{F}((r_2 \wedge g_2) \vee \neg g_2)) \wedge \\
&\mathbf{G}(r_1 \rightarrow \mathbf{F}\,g_1) \wedge \\
&\mathbf{G}(r_2 \rightarrow \mathbf{F}\,g_2) \wedge \\
&\mathbf{G}\,\neg(g_1 \wedge g_2)
\end{aligned}$$

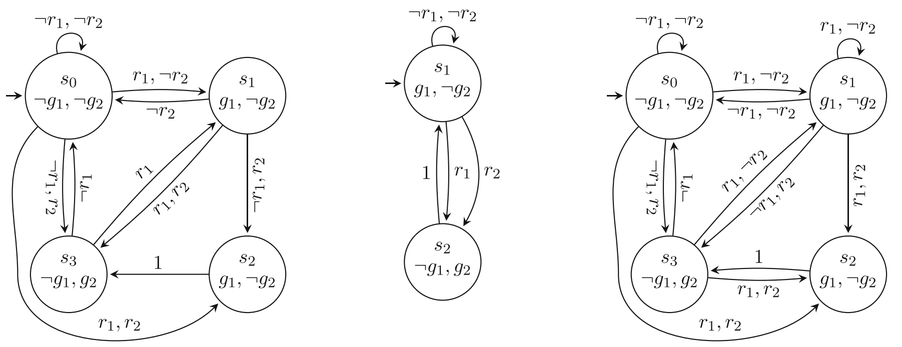
(a) Complete Specification

guarantee

$$\begin{aligned}
&\mathbf{G}(r_1 \rightarrow \mathbf{F}\,g_1) \wedge \\
&\mathbf{G}(r_2 \rightarrow \mathbf{F}\,g_2) \wedge \\
&\mathbf{G}\,\neg(g_1 \wedge g_2)
\end{aligned}$$

(b) Incomplete Specification

Fig. 11 LTL specification for 'full' arbiter of 2 clients



(a) Bounded synthesis of complete specification

(b) Bounded synthesis of incomplete specification

(c) Non-vacuous synthesis of incomplete specification

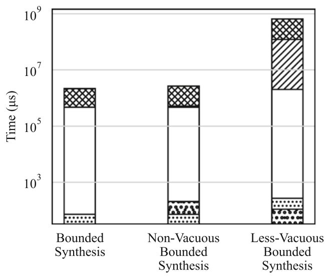
Fig. 12 Synthesized arbiters of complete and incomplete specifications of full arbiter as read in Fig. 11

appears in Fig. 11 (right). The properties that are removed in the partial specification state that grants are never given *"unnecessarily"*.

The transition systems that are synthesized for the full and partial specification appear in Fig. 12a, b respectively. On the other hand, starting from the partial specification, after four iterations of the non-vacuous synthesis the system we get is as shown in Fig. 12c, which again satisfies the full specification. Figure 10 depicts the runtime breakdown of bounded synthesis and non-vacuous bounded synthesis for both the complete and incomplete versions of the 'full' arbiter (Fig. 13).

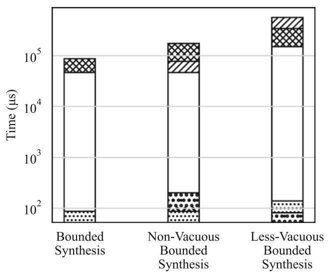
5.3 A 'Pnueli' arbiter

A 'Pnueli' arbiter of two clients is a handshake mechanism such that whenever a client sets a request the arbiter will set and keep the corresponding grant high as long as the request is high [12]. The complete and incomplete specification of a 'Pnueli' arbiter of two clients is shown in Fig. 14. The incomplete specification allows the arbiter to set a grant and never unset it; therefore, the synthesized system may issue vacuous grants for each client infinitely often unless the other client sends a request—see Fig. 15b.



		Bounded Synthesis	Non-Vacuous Bounded Synthesis	Less-Vacuous Bounded Synthesis
□	Co-Buchi Translation	464694	464694	1994477
▨	Buchi Translation	0	505440	121539721
░	Co-Buchi Encoding	72	72	268
▓	Buchi Encoding	0	206	109
▩	SMT Solver	2137801	2623794	652392861
	Sum	2602567	3594206	775927437

(a) Runtime breakdown of the complete specification of the 'full' arbiter.



		Bounded Synthesis	Non-Vacuous Bounded Synthesis	Less-Vacuous Bounded Synthesis
□	Co-Buchi Translation	46561	46561	150155
▨	Buchi Translation	0	77427	566389
░	Co-Buchi Encoding	87	87	139
▓	Buchi Encoding	0	201	82
▩	SMT Solver	87179	174226	342458
	Sum	133828	298502	1059222

(b) Runtime breakdown of the incomplete specification of the 'full' arbiter.

Fig. 13 Runtime breakdown of synthesis of the 'full' arbiter

assume

$$\begin{aligned}
&\neg r_1 \wedge \neg r_2 \wedge \\
&\mathbf{G}((r1 \wedge \neg g1 \rightarrow \mathbf{X}\, r_1) \wedge (\neg r_1 \wedge g_1 \rightarrow \mathbf{X}\, \neg r1)) \wedge \\
&\mathbf{G}((r2 \wedge \neg g2 \rightarrow \mathbf{X}\, r_2) \wedge (\neg r_2 \wedge g_2 \rightarrow \mathbf{X}\, \neg r_2)) \wedge \\
&\mathbf{G}\,\mathbf{F}(\neg r_1 \vee \neg g_1) \wedge \\
&\mathbf{G}\,\mathbf{F}(\neg r_2 \vee \neg g_2)
\end{aligned}$$

guarantee

$$\begin{aligned}
&\neg g_1 \wedge \neg g_2 \wedge \\
&\mathbf{G}(((\neg r_1 \wedge \neg g_1) \rightarrow \mathbf{X}\, \neg g_1) \wedge ((r_1 \wedge g_1) \rightarrow \mathbf{X}\, g_1)) \wedge \\
&\mathbf{G}(((\neg r_2 \wedge \neg g_2) \rightarrow \mathbf{X}\, \neg g_2) \wedge ((r_2 \wedge g_2) \rightarrow \mathbf{X}\, g_2)) \wedge \\
&\mathbf{G}\,\mathbf{F}(r_1 \longleftrightarrow g_1) \wedge \\
&\mathbf{G}\,\mathbf{F}(r_2 \longleftrightarrow g_2) \wedge \\
&\mathbf{G}\,\neg(g_1 \wedge g_2)
\end{aligned}$$

(a) Complete Specification

assume

$$\begin{aligned}
&\neg r_1 \wedge \neg r_2 \wedge \\
&\mathbf{G}((r1 \wedge \neg g1 \rightarrow \mathbf{X}\, r_1) \wedge (\neg r_1 \wedge g_1 \rightarrow \mathbf{X}\, \neg r1)) \wedge \\
&\mathbf{G}((r2 \wedge \neg g2 \rightarrow \mathbf{X}\, r_2) \wedge (\neg r_2 \wedge g_2 \rightarrow \mathbf{X}\, \neg r_2)) \wedge \\
&\mathbf{G}\,\mathbf{F}(\neg r_1 \vee \neg g_1) \wedge \\
&\mathbf{G}\,\mathbf{F}(\neg r_2 \vee \neg g_2)
\end{aligned}$$

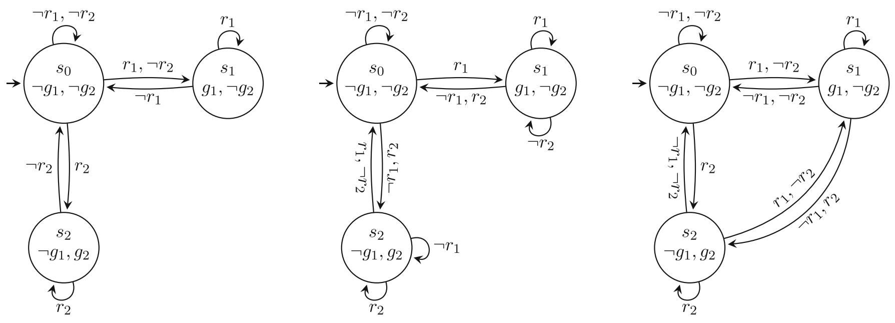
guarantee

$$\begin{aligned}
&\neg g_1 \wedge \neg g_2 \wedge \\
&\mathbf{G}(((\neg r_1 \wedge \neg g_1) \rightarrow \mathbf{X}\, \neg g_1) \wedge ((r_1 \wedge g_1) \rightarrow \mathbf{X}\, g_1)) \wedge \\
&\mathbf{G}(((\neg r_2 \wedge \neg g_2) \rightarrow \mathbf{X}\, \neg g_2) \wedge ((r_2 \wedge g_2) \rightarrow \mathbf{X}\, g_2)) \wedge \\
&\mathbf{G}\,\mathbf{F}(r_1 \rightarrow g_1) \wedge \\
&\mathbf{G}\,\mathbf{F}(r_2 \rightarrow g_2) \wedge \\
&\mathbf{G}\,\neg(g_1 \wedge g_2)
\end{aligned}$$

(b) Incomplete Specification

Fig. 14 LTL specification for a 'Pnueli' arbiter of two clients. The incomplete specification lacks the right-to-left implication in the 4th and 5th lines of the guarantee

The result of our non-vacuous synthesis from the partial specification again satisfies the full specification, as shown in Fig. 15c, and is synthesized in one step. This system also satisfies the specification in a less vacuous way than the system synthesized from the complete specification using the previous version of PARTY, without the new functionality of non-vacuity. Figure 10 depicts runtime breakdown of both synthesis methods for complete and incomplete versions of the 'Pneuli' arbiter (Fig. 16).

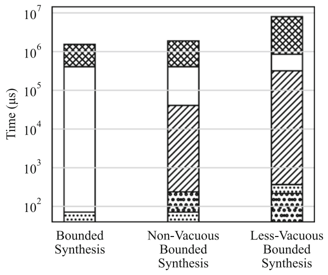


(a) Bounded synthesis of complete specification

(b) Bounded synthesis of incomplete specification

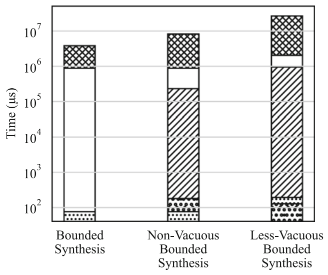
(c) Non-vacuous synthesis of incomplete specification

Fig. 15 Synthesized arbiters of complete and incomplete specifications of a 'Pnueli' arbiter as read in Fig. 14



		Bounded Synthesis	Non-Vacuous Bounded Synthesis	Less-Vacuous Bounded Synthesis
□	Co-Buchi Translation	407917	407917	856014
▨	Buchi Translation	0	40710	318516
░	Co-Buchi Encoding	71	71	366
▓	Buchi Encoding	0	238	217
▦	SMT Solver	1536610	1892877	7996624
	Sum	1944598	2341812	9171737

(a) Runtime breakdown of the complete specification of the 'Pnueli' arbiter.



		Bounded Synthesis	Non-Vacuous Bounded Synthesis	Less-Vacuous Bounded Synthesis
□	Co-Buchi Translation	873798	873798	2051211
▨	Buchi Translation	0	232198	953098
░	Co-Buchi Encoding	77	77	194
▓	Buchi Encoding	0	180	128
▦	SMT Solver	3837228	8171791	26652266
	Sum	4711103	9278045	29656898

(b) Runtime breakdown of the incomplete specification of the 'Pnueli' arbiter.

Fig. 16 Runtime breakdown of synthesis of the 'Pnueli' arbiter

6 Conclusion

In synthesis, it is hard to expect the designer to think of a complete specification. As a result, the large range of possible systems that satisfy the specification permits designs that stand in contrast to the designer's intent. We proposed in this article to apply the concept of vacuity to address this problem. Our method narrows down the range of legitimate synthesised system to those that satisfy the (partial) specification in a *meaningful* way, a well-known concept from using vacuity in model-checking. But as we argued, we do not have to commit to the Boolean nature of the classical definition of vacuity: we showed how a system can be made *less* vacuous, even if it already satisfies the specification non-vacuously. Our experiments showed that our method is capable of synthesising better designs, in the sense that they even satisfy parts of the specification that we deliberately removed and were hence inaccessible to the synthesis algorithm. Perhaps in the future synthesis will be used in the industry, and then our conjecture that this process can save time to the designer will be tested with a user-study.

Our solution is based on a novel bounded synthesis technique that combines universal and existential properties; It paves the way for generalizing our technique to full CTL*. Our tool PARTY is available on the web to try and improve.

Acknowledgements We would like to thank Nir Piterman for his insights on infinite chains of ever less vacuous systems and Ayrat Khalimov for his comments on existential bounded synthesis and his valuable assistance with the implementation.

Funding Open access funding provided by Graz University of Technology.

A: An example of incomparable least-vacuous systems

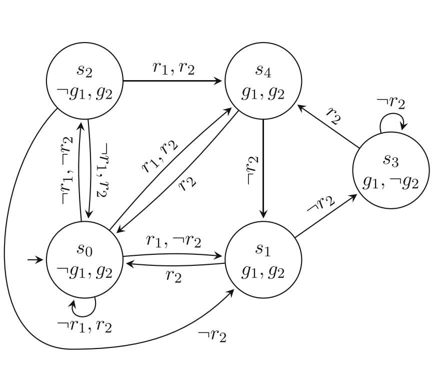
Consider the specification φ that is a conjunction of the following four properties:

$$\varphi_1 = \mathrm{G}(r_1 \rightarrow \mathrm{F}g_1)\ , \quad \varphi_2 = \mathrm{G}(\neg g_1 \rightarrow \mathrm{X}g_2)\ ,$$
$$\varphi_3 = \mathrm{G}(r_2 \rightarrow \mathrm{F}g_2)\ , \quad \varphi_4 = \mathrm{G}(\neg g_2 \rightarrow \mathrm{X}g_1)\ ,$$

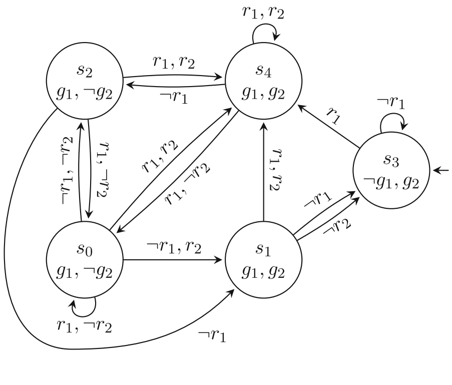
where r_i is an input and g_i is an output. The following interesting witness formulas

$$\neg\varphi_1[r_1 \leftarrow \bot] = \mathrm{FG}(\neg g_1)\ ,$$
$$\neg\varphi_3[r_2 \leftarrow \bot] = \mathrm{FG}(\neg g_2)\ .$$

give rise to incomparable least-vacuous systems as described in Fig. 17. Indeed, the witness formula $\mathrm{FG}(\neg g_1)$ (resp. $\mathrm{FG}(\neg g_2)$) requires a trace that satisfies it to have a suffix $\neg g_1^\omega$ (resp. $\neg g_2^\omega$), hence in order to satisfy the property φ_2 (resp. φ_4), the same suffix should satisfy $\mathrm{X}(g_2^\omega)$ (resp. $\mathrm{X}(g_1^\omega)$). Thus, each trace either satisfies the interesting witness $\mathrm{FG}(\neg g_1)$ or $\mathrm{FG}(\neg g_2)$, but not both, hence only one of the two sets of witness traces can be maximized. The proof of the least vacuity is similar to the proofs of Propositions 1 and 3.



(a) M_0



(b) M_1

Fig. 17 Incomparable least-vacuous systems realizing φ

B: Performance comparison

Here we compare the performance of non-vacuous bounded synthesis to 'normal' bounded synthesis, using a benchmark set of 164 specifications. The set includes the `LTL-Real` benchmarks of *synthcomp 2020*, `lily`, `genbuf`, `acacia+`, and `party-elli`. Our implementation uses Z3 4.8.10 and SPOT 2.9. It is worth mentioning that our implementation uses integer encoding of co-büchi automata for both methods.

We ran each experiment twice and each time for a total of 60 minutes. In both runs, 70 experiments timed out, 85 were realizable, and 9 were unrealizable. We averaged the run-time profile of both runs and computed the performance loss in making the synthesis non-vacuous.

Of all realizable experiments the non-vacuous bounded synthesis gained negligible performance in three cases. Meanwhile, non-vacuous bounded synthesis timed out on 3 experiments that were realizable through bounded synthesis. Table 1 gives a more detailed comparison.

The biggest factor in the performance loss is the solver time. This is evident in the plots shown in Figs. 18 and 19. Other than the solver time, much of the run-time is spent on

Table 1 Performance comparison

Description	Number of experiments
Non-vacuous bounded synthesis is up to 2x faster	3
Bounded synthesis is up to 2x faster	24
Bounded synthesis is (2x, 3x] faster	11
Bounded synthesis is (3x, 17x] faster	23
Bounded synthesis is (20x, 1000x] faster	21
Non-vacuous bounded synthesis timed out	3
Realizable	85
Unrealizable	9
Both timed out	70
Total	164

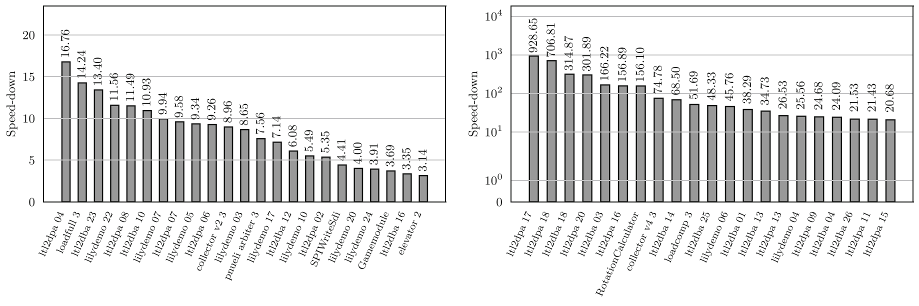


Fig. 18 Performance loss on total synthesis time

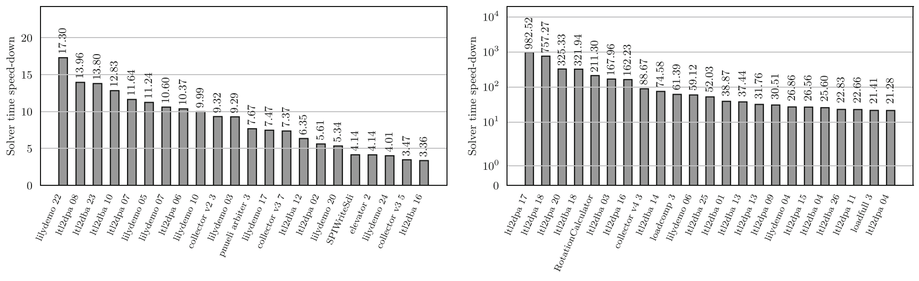


Fig. 19 Performance loss on solver time

translating each specification to a co-büchi automaton, which is shared by both synthesis methods; thus, the total performance loss is typically less than the solver performance loss.

References

1. Armoni R, Fix L, Flaisher A, Grumberg O, Piterman N, Tiemeyer A, Vardi MY (2003) Enhanced vacuity detection in linear temporal logic. In: CAV, LNCS, Springer, vol 2725, pp 368–380
2. Beer I, Ben-David S, Eisner C, Rodeh Y (2001) Efficient detection of vacuity in ACTL formulas. Formal Methods Syst Des 18(2):279–290
3. Bloem R, Chatterjee K, Greimel K, Henzinger TA, Hofferek G, Jobstmann B, Könighofer B, Könighofer R (2014) Synthesizing robust systems. Acta Inf 51:193–220
4. Bloem R, Chatterjee K, Henzinger TA, Jobstmann B (2009) Better quality in synthesis through quantitative objectives. In: CAV, LNCS, Springer, vol 5643, pp 140–156
5. Bloem R, Chockler H, Ebrahimi M, Strichman O (2017) Synthesizing non-vacuous systems. In: Proceedings of the 18th international conference in verification, model checking, and abstract interpretation (VMCAI), Lecture notes in computer science, Springer, vol 10145, pp 55–72
6. Bustan D, Flaisher A, Grumberg O, Kupferman O, Vardi M (2005) Regular vacuity. In: CHARME, LNCS, Springer, vol 3725, pp 191–206
7. Chechik M, Gurfinkel A (2004) Extending extended vacuity. In: FMCAD, LNCS, vol 3312
8. Chockler H, Gurfinkel A, Strichman O (2013) Beyond vacuity: towards the strongest passing formula. Formal Methods Syst Des 43(3):1–8
9. Church A (1963) Logic, arithmetics, and automata. In: ICM, institut Mittag-Leffler, pp 23–35
10. Finkbeiner B, Schewe S (2012) Bounded synthesis. Int J Softw Tools Technol Transf 15(5):519–539
11. Jacobs S, Bloem R, Brenguier R, Könighofer R, Pérez GA, Raskin J, Ryzhyk L, Sankur O, Seidl M, Tentrup L, Walker A (2015) The second reactive synthesis competition. In: SYNT
12. Jobstmann B, Staber S, Griesmayer A, Bloem R (2012) Finding and fixing faults. J Comput Syst Sci 78(2):35–49

13. Khalimov A, Jacobs S, Bloem R (2013) PARTY parameterized synthesis of token rings. In: CAV, pp 928–933
14. Kupferman O, Vardi M (2003) Vacuity detection in temporal model checking. J Softw Tools Technol Transf 4(2):224–233
15. Namjoshi KS (2004) An efficiently checkable, proof-based formulation of vacuity in model checking. In: CAV, LNCS, Springer, vol 3114, pp 57–69
16. Pnueli A (1977) The temporal logic of programs. In: FOCS, pp 46–57
17. Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: POPL, Austin, pp 179–190
18. Samanta R, Deshmukh JV, Chaudhuri S (2013) Robustness analysis of networked systems. In: VMCAI, pp 229–247
19. Vardi M, Wolper P (1994) Reasoning about infinite computations. Inf Comput 115(1):1–37

Affiliations

Roderick Bloem[1] · Hana Chockler[2] · Masoud Ebrahimi[1] · Ofer Strichman[3]

✉ Ofer Strichman
ofers@ie.technion.ac.il

Roderick Bloem
roderick.bloem@iaik.tugraz.at

Hana Chockler
hana.chockler@kcl.ac.uk

Masoud Ebrahimi
masoud.ebrahimi@iaik.tugraz.at

1 Graz University of Technology, Graz, Austria

2 King's College London, London, UK

3 Information Systems Engineering, IE, Technion , Haifa, Israel