King's Research Portal

DOI:

[10.1007/978-3-319-52234-0_4](https://doi.org/10.1007/978-3-319-52234-0_4)

Document Version

Peer reviewed version

[Link to publication record in King's Research Portal](#)

Citation for published version (APA):

Bloem, R., Chockler, H., Ebrahimi, M., & Strichman, O. (2017). Synthesizing non-vacuous systems. In *Proceedings of 18th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI): Lecture Notes in Computer Science* (Vol. 10145, pp. 55-72). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 10145 LNCS). SpringerVerlag Berlin Heidelberg. https://doi.org/10.1007/978-3-319-52234-0_4

Citing this paper

Please note that where the full-text provided on King's Research Portal is the Author Accepted Manuscript or Post-Print version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version for pagination, volume/issue, and date of publication details. And where the final published version is provided on the Research Portal, if citing you are again advised to check the publisher's website for any subsequent corrections.

General rights

Copyright and moral rights for the publications made accessible in the Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the Research Portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the Research Portal

Take down policy

If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Synthesizing Non-Vacuous Systems

Roderick Bloem¹, Hana Chockler², Masoud Ebrahimi¹, and Ofer Strichman³

¹ Graz University of Technology

² King's College London

³ Information Systems Engineering, IE, Technion

Abstract. Vacuity detection is a common practice accompanying model checking of hardware designs. Roughly speaking, a system satisfies a specification vacuously if it can satisfy a stronger specification obtained by replacing some of its subformulas with stronger expressions. If this happens then part of the specification is immaterial, which typically indicates that there is a problem in the model or the specification itself. We propose to apply the concept of vacuity to the synthesis problem. In synthesis, there is often a problem that the specifications are incomplete, hence under-specifying the desired behaviour, which may lead to a situation in which the synthesised system is different than the one intended by the designer. To address this problem we suggest an algorithm and a tool for *non-vacuous bounded synthesis*. It combines synthesis for universal and existential properties; the latter stems from the requirement to have at least one interesting witness for each strengthening of the specification. 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 user decides when the system reflects their intent.

1 Introduction

Given a temporal specification φ , the goal of *reactive synthesis* [8, 16] 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 [10].

Along with the greater applicability of synthesis has come significant attention to the quality of the synthesized systems. Often, systems are underspecified, i.e., their specifications do not include certain desirable properties of the system. Hence, we can add an informal element to the definition of the synthesis problem, namely that it is to build a transition system M that in addition to satisfying the specification φ , it also captures the *designer's intent*. Automatically bridging this gap between the formal specification and the designer's intent is the topic of this article. Previous proposals to tackle incomplete specifications include quantitative specifications to make it easier to specify certain properties [4]

and synthesis of systems that are robust against environment errors, even if the way to react to such errors has not been specified explicitly [3, 17].

In this paper we suggest a different approach, based on leveraging the notion of *vacuity* [2]. Our conjecture is that if the synthesized system M satisfies φ *non-vacuously*, then M is likely closer to the user’s intent, because it satisfies φ in a more “meaningful” way. If our conjecture is right, then this can save some of the effort that is required from the user to complete and refine his/her specification. Consider, for example, the property

$$\varphi = \mathbf{G}(req \rightarrow \mathbf{F} grant). \quad (1)$$

A system M with one state satisfying *grant* (regardless of *req*) satisfies φ , and is indeed a legitimate outcome of synthesising (1). However this system also satisfies stronger properties such as $\mathbf{G} \mathbf{F} grant$, and indeed 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. 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 , we say that the specification is satisfied vacuously (see below a formal definition); in order for a system M to satisfy a property φ non-vacuously we need it to include desired paths like π that are called *interesting witnesses* [2]. These are executions that demonstrate non-vacuous satisfaction of the original property.

There are multiple definitions of vacuity in the literature [1, 2, 5, 6, 13, 14], but the method that we will describe in this paper is independent of the chosen definition. Most commercially used vacuity-detection tools use the generalised definition by Kupferman and Vardi [13], which is what we will follow here: Let ψ be a subformula in φ . The *strengthening of φ with respect to ψ* is $\varphi[\psi \leftarrow \perp]$.⁴ If $M \models \varphi[\psi \leftarrow \perp]$ 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 $M \models \mathbf{E} \neg \varphi[\psi \leftarrow \perp]$. As shown in [13], it is sufficient to consider 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 [13], 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:

$$(M \models \mathbf{A} \varphi) \wedge \bigwedge_{\psi \in \text{Lit}(\varphi)} (M \models \mathbf{E} \neg \varphi[\psi \leftarrow \perp]), \quad (2)$$

where $\text{Lit}(\varphi)$ denotes the literals of φ . One of the contributions of this article is the extension of the *bounded synthesis* [9] algorithm to handle this fragment,

⁴ This means that we swap ψ with **false** if ψ is in positive polarity, and with **true** otherwise. Hence, e.g., if $\varphi \equiv \psi_1 \Rightarrow \psi_2$, then $\varphi[\psi_1 \leftarrow \perp] \equiv \psi_2$.

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 user decides when the system reflects their intent. Thus, we define a partial order 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. Thus, we generate decreasingly vacuous systems up to *the least vacuous* system for a given number of states. In Sect. 5 we show that if the number of states is unbounded, then for some specifications the chain of less and less vacuous systems is infinite.

We have implemented the non-vacuous bounded synthesis algorithm on top of the PARTY synthesizer [12] which is available for download⁵. 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.

A Motivating Example

We illustrate our ideas with a running example: a specification for 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:

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

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 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 $M_0 \models \varphi_1[r_1 \leftarrow \perp]$ and $M_0 \models \varphi_2[r_2 \leftarrow \perp]$, where the $\perp$ 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.⁶ This system satisfies φ non-vacuously in all its subformulas. Indeed:

1. $M_1 \not\models \varphi_1[r_1 \leftarrow \perp]$, as the path $\pi_1 = s_0^\omega$ corresponds to the output trace $(\neg g_1, g_2)^\omega$, which falsifies $\mathbf{G} \mathbf{F} g_1$;

⁵ www.iaik.tugraz.at/content/research/opensource/non-vacuous_systems

⁶ Note the unusual semantics of LTL on this figure: 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 on the state s_1 itself.

Fig. 1: Systems of the running example.

2. $M_1 \not\models \varphi_2[r_2 \leftarrow \perp]$, as the path $\pi_1 = s_0, s_1^\omega$ corresponds to the output trace $(\neg g_1, g_2), (g_1, \neg g_2)^\omega$, which falsifies $\mathbf{GF} g_2$;
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, $G(\neg r_1)$ because we have no control over the inputs.

In Sect. 4, we discuss ways to improve the synthesised system by increasing the number of its non-vacuous traces. We illustrate these ideas on the results of the next iterations of the tool on our running example.

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 $\text{AP} = I \cup O$, $\mathcal{I} = 2^I$, $\Sigma = 2^O$, and $\Gamma = 2^{\text{AP}}$. A *finite, Σ -labeled $\mathcal{I}$ -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, $\tau : S \times \mathcal{I} \rightarrow S$ is a transition function, and $o : S \rightarrow \Sigma$ is a labelling function.

Definition 1 (Path). A path of a transition system M , denoted by π , is an infinite sequence of states $s_0, s_1, \dots \in S^\omega$ such that for $i > 0 \exists v \in \mathcal{I}. s_i = \tau(s_{i-1}, v)$.

We denote by $\text{paths}_M(s)$ the set of all paths of M originating at $s \in S$, omitting M when it is clear from the context.

Definition 2 (Trace). A trace corresponding to a path $\pi = s_0, s_1, \dots$ of a transition system M , denoted $\text{trace}(\pi)$, is an infinite word $v_0 \cup \sigma_0, v_1 \cup \sigma_1, \dots$ over Γ , such that for $i \geq 0$, $s_{i+1} = \tau(s_i, v^i)$, and $\sigma^i = o(s_i)$.

We denote by $\text{traces}(M)$ the set of all traces of M . For an input trace $\pi \in \mathcal{I}^\omega$, we denote by $M(\pi)$ the (unique) trace of M whose projection to $\mathcal{I}^\omega$ equals π .

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 [15]. The semantics of LTL is

defined over AP with respect to infinite paths of M in a standard way. In this paper, we synthesise systems that satisfy the following simple fragment of CTL*:

$$\Phi ::= \mathbf{A} \varphi \mid \mathbf{E} \varphi \mid \Phi \wedge \Phi, \quad (4)$$

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

Definition 3. *For a state s of a transition system M ,*

$$\begin{aligned} s \models \mathbf{A} \varphi & \text{ iff } \forall \pi \in \text{paths}_M(s). \pi \models \varphi \\ s \models \mathbf{E} \varphi & \text{ iff } \exists \pi \in \text{paths}_M(s). \pi \models \varphi. \end{aligned}$$

A transition system M satisfies a formula ϕ , written $M \models \phi$, if its initial state s_0 does.

2.3 Nondeterministic Büchi Automata

An LTL formula can be represented by a nondeterministic Büchi automata [18]: 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, $\rho : Q \times \mathcal{I} \times \Sigma \rightarrow \mathcal{P}(Q)$ is the transition relation, and α is the set of accepting states; recall Σ and $\mathcal{I}$ are defined in Sect. 2.1.

Definition 4 (run). *Given an infinite word $\omega = v_0 \cup \sigma_0, v_1 \cup \sigma_1, \dots$ over $2^{I \cup O}$, a corresponding run of an automaton $\mathcal{A}$, denoted by $\text{run}(w)$, is an infinite path $\pi = q_0, q_1, \dots \in Q^\omega$ where for all $i \geq 0$, $q_{i+1} \in \rho(q_i, v^i, \sigma^i)$.*

Definition 5 (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 a uninteresting way). As proved in [13], for subformulas that occur in the property only once (or multiple time with the same polarity), this is equivalent to checking the effect of replacing a subformula with $\perp$. Furthermore, if the property is in NNF, it is enough to check the effect of replacing atomic propositions with $\perp$. Hence, we use the following definition of vacuity that allows for efficient detection algorithm:

Definition 6 (Vacuity [2,13]). *A transition system M satisfies an LTL property φ vacuously iff $M \models \varphi$ and there exists a literal ψ (an atomic proposition or its negation) of φ such that $M \models \varphi[\psi \leftarrow \perp]$, where $\varphi[\psi \leftarrow \perp]$ denotes φ with ψ replaced by $\perp$.*

The formula $\varphi[\psi \leftarrow \perp]$ is a *strengthening* of φ since $\varphi[\psi \leftarrow \perp] \rightarrow \varphi$ and we call the negation $\varphi_\psi = \neg\varphi[\psi \leftarrow \perp]$ of a strengthening a *witness formula*. An 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 .

The concept of witnesses and strengthenings is not restricted to Def. 6, and it lends itself, in theory, to other definitions of vacuity [1, 6, 7]. 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 [9]. The idea is to let an SMT solver synthesize a transition system M (i.e., choose the transitions between and the labeling of the given number of 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 .⁷ The states of $\mathcal{G}$ are annotated with two functions: a reachability function $\lambda^{\mathbb{B}} : Q \times S \rightarrow \mathbb{B}$ and a ranking function $\lambda^{\#} : Q \times S \rightarrow C \subset \mathbb{N}$, where $C = \{0, \dots, |Q| \times |S| - 1\}$. 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:

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

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

$$\bigwedge_{\substack{q, q' \in Q \\ s, s' \in S \\ v \in T}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), v) \wedge s' \in \tau(s, v) \rightarrow \lambda^{\mathbb{B}}(q', s') . \quad (6)$$

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:

$$\bigwedge_{\substack{q \in \alpha, q' \in Q \\ s, s' \in S \\ v \in T}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), v) \wedge s' \in \tau(s, v) \rightarrow \lambda^{\#}(q', s') > \lambda^{\#}(q, s) ; \quad (7)$$

⁷ 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., $|\Sigma| = |\mathcal{T}| = 1$).

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

$$\bigwedge_{\substack{q \in Q \setminus \alpha, q' \in Q \\ s, s' \in S \\ v \in T}} \lambda^{\mathbb{B}}(q, s) \wedge q' \in \rho(q, o(s), v) \wedge s' \in \tau(s, v) \rightarrow \lambda^{\#}(q', s') \geq \lambda^{\#}(q, s) . \quad (8)$$

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 [9].

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.

Theorem 1 ([9]). *Given a Büchi automaton $\mathcal{A} = (Q, q_0, \rho, \alpha)$ constructed from $\neg\varphi$, transition system $M = (S, s_0, \tau, o)$ satisfies $\mathbf{A}\varphi$ 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.

3 Non-vacuous Bounded Synthesis

In this section we describe *non-vacuous bounded synthesis* – a 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),

$$M \models \mathbf{A} \varphi \wedge \bigwedge_{\psi \in \text{Lit}(\varphi)} \mathbf{E} \neg \varphi[\psi \leftarrow \perp]$$

(note that (2) is based on our choice of definition for vacuity). We call $\neg \varphi[\psi \leftarrow \perp]$ the *witness formulas* for non-vacuity of φ .

Note that not all witness formulas add interesting information. For instance, for φ as defined in (3), the witness formula $\neg \varphi_1[g_1 \leftarrow \perp] = \mathbf{F} r_1$ 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 (2).

3.2 Bounded Synthesis for Existential Formulae

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 $\lambda^* : Q \times S \rightarrow \mathbb{B}$ 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:

$$\lambda^*(q_0, s_0) . \quad (9)$$

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:

$$\bigwedge_{\substack{q \in Q \setminus \alpha \\ s \in S \\ v \in \mathcal{T}}} \left(\lambda^*(q, s) \rightarrow \bigvee_{\substack{q' \in Q \\ s' \in S}} \left(\begin{array}{l} q' \in \rho(q, o(s), v) \wedge s' \in \tau(s, v) \wedge \\ \lambda^*(q', s') \wedge \lambda^\#(q', s') < \lambda^\#(q, s) \end{array} \right) \right) . \quad (10)$$

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):

$$\bigwedge_{\substack{q \in \alpha \\ s \in S \\ v \in \mathcal{T}}} \left(\lambda^*(q, s) \rightarrow \bigvee_{\substack{q' \in Q \\ s' \in S}} \left(\begin{array}{l} q' \in \rho(q, o(s), v) \wedge s' \in \tau(s, v) \wedge \\ \lambda^*(q', s') \end{array} \right) \right) . \quad (11)$$

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

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.

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 $\mathbf{E} \varphi'$ iff it corresponds to a solution to constraints (9)-(11).*

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 $\pi = (q_0, s_0)(q_1, s_1) \dots [(q_n, s_n) \dots (q_m, s_m)]^\omega$ such that q_i is accepting for some $i \in [n, m]$. We have to show that in such a case (9)-(11) are satisfiable. Marking all the states on the path clearly satisfies (9), and the λ^* predicate is true along this path as required by constraints (10) and (11). It is left to show that there exists a ranking function that satisfies (10). 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 accepting states are bound by constraint (10). The figure below demonstrates this ranking for $n = 3, m = 6$, and $i = 5$.

($\Leftarrow$) Assume that (9)-(11) are satisfiable. The set of marked states must include a lasso-shaped path beginning from the initial state, and the fact that (10) is satisfied means that there exists an accepting state in the loop. Hence the run graph must contain an accepting path. $\square$

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 (9)-(11).*

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 $\varphi = \mathbf{G}(r \rightarrow \mathbf{F}g)$, there will be at least one path on which $\mathbf{G} \mathbf{F}g$ 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 systems that are non-vacuous. To that end, in this section we introduce a partial order between such systems. We consider a system *less vacuous* than another if more input traces yield interesting witnesses. For the property above, for example, this corresponds to more witnesses to $\neg \mathbf{G} \mathbf{F}g$. Intuitively, this approximates the idea of a *trigger*, where g is triggered by r , and should preferably not occur without r .

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 system of a given size. If the size is unbounded, however, we show that for some specifications, this order gives rise to infinite chains of ever less vacuous systems.

4.1 A 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 $M_1 \preceq_\psi M_2$ 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

$$M_1 \preceq_\psi M_2 \text{ iff } \forall \pi \in \mathcal{Y}^\omega. (M_1(\pi) \models \varphi_\psi) \rightarrow (M_2(\pi) \models \varphi_\psi). \quad (12)$$

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 :

$$M_1 \prec_\psi M_2 \text{ iff } M_1 \preceq_\psi M_2 \text{ and } \exists \pi \in \mathcal{Y}^\omega. (M_1(\pi) \not\models \varphi_\psi) \wedge (M_2(\pi) \models \varphi_\psi). \quad (13)$$

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 preorder $\preceq$ as

$$M_1 \preceq M_2 = \forall \varphi_\psi \in \Psi. M_1 \preceq_\psi M_2, \quad (14)$$

and the strict partial order $\prec$ as

$$M_1 \prec M_2 = M_1 \preceq M_2 \text{ and } \exists \varphi_\psi \in \Psi. (M_1 \prec_\psi M_2). \quad (15)$$

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.

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 , according to $\prec$. This system may not be unique.

4.2 An Infinite Vacuity Chain

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

$$\varphi = (\mathbf{G} r) \rightarrow (\mathbf{F} g). \quad (16)$$

The only witness formula for φ is

$$\varphi_r = \mathbf{G} \neg g. \quad (17)$$

Fig. 4 depicts an abstract transition system M_k of arbitrary size (i.e., $k + 3$) that realizes specification φ non-vacuously for any k .

Fig. 4: An example of infinite vacuity chain

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

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

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 . $\square$

Fig. 5: The final non-vacuous system M_2 .

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).

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

$$M'_1 \times M_2 \models \mathbf{A} \varphi \wedge \mathbf{A}(\varphi'_\psi \rightarrow \varphi_\psi) \wedge \mathbf{E}(\neg \varphi'_\psi \wedge \varphi_\psi).$$

Note that φ and φ_ψ consider the outputs of M_2 and φ'_ψ considers the outputs of M_1 , while both systems receive the same inputs.

Theorem 3. M_1 is strictly less vacuous than M_2 iff

$$M'_1 \times M_2 \models \mathbf{A}(\varphi \wedge \bigwedge_{\varphi_\psi \in \Psi} (\varphi'_\psi \rightarrow \varphi_\psi)) \wedge \mathbf{E}(\bigvee_{\varphi_\psi \in \Psi} (\neg \varphi'_\psi \wedge \varphi_\psi)). \quad (18)$$

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

If we fix a maximal size for the system, it implies that we can synthesise a maximally non-vacuous one (i.e., least vacuous) by repeated application of this procedure.

4.4 A Least Vacuous System for our Running Example

Consider once again our running example from the introduction. Fig. 5 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. 1b).

System M_2 is strictly less vacuous than M_1 . Recall that the two witness formulas are $\varphi_{r_1} = \mathbf{F} \mathbf{G} \neg g_1$ and $\varphi_{r_2} = \mathbf{F} \mathbf{G} \neg g_2$. 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 in 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 2. *M_2 is a least vacuous system with respect to $\{\varphi_{r_1}, \varphi_{r_2}\}$.*

Proof. Let M be an arbitrary system that satisfies φ . For an input sequence $\pi \in \mathcal{T}^\omega$, assume that π induces a path in M that satisfies $\varphi_1[r_1 \leftarrow \perp] = \mathbf{F} \mathbf{G} \neg g_1$. Since this path, in particular, satisfies φ , it also satisfies $\mathbf{F} \mathbf{G} \neg r_1$ (otherwise there would have been requests that are never granted). Observing Fig. 5, it is easy to see that the same input sequence π would induce a path in M_2 with an infinite suffix $\{s_0, s_2\}^\omega$, hence, in particular, it satisfies $\mathbf{F} \mathbf{G} \neg g_1$. A similar argument holds for $\varphi_2[r_2 \leftarrow \perp]$. 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 (18) can be reduced to CTL* realizability question.

5 Experimental Evaluation

We implemented the described technique in the PARTY synthesizer [12] and 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 and ran synthesis again; our motivation was to see whether starting with a partial specification, with non-vacuous synthesis we can synthesize a system that satisfies the full specification. Clearly this highly depends on the properties that we choose to remove, but recall that this is not the scenario that we are aiming at anyway. We aim at a scenario in which there is no full specification, and non-vacuous synthesis accelerates the convergence towards the desired system. Since we cannot run such an experiment, the experiments below only give us a certain indication for the power of this technique.

In the three experiments that we conducted, non-vacuous synthesis was able to synthesize a system that satisfies the original, full specification, although we emphasise 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.

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. 6. The specification should be interpreted as ‘every run that satisfies the *assume* predicates should also satisfy the *guarantee* predicates’.

Complete Specification	Incomplete Specification
assume $\mathbf{G} \neg(r_1 \wedge r_2)$ guarantee $\mathbf{G}(r_1 \leftrightarrow \mathbf{X} g_1) \wedge$ $\mathbf{G}(r_2 \leftrightarrow \mathbf{X} g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$	assume $\mathbf{G} \neg(r_1 \wedge r_2)$ guarantee $\mathbf{G}(r_1 \rightarrow \mathbf{X} g_1) \wedge$ $\mathbf{G}(r_2 \rightarrow \mathbf{X} g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$

Fig. 6: LTL specification for the ‘next’ arbiter of two clients. Note that the incomplete specification on the right excludes the right-to-left implications in the guarantee.

As depicted in Figs. 7a and 7b, 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. 7b, three iterations of the non-vacuous synthesis process result in the system shown in Fig. 7c, which satisfies the original, full specification.

Fig. 7: Synthesized arbiters of the complete and incomplete specifications of the ‘next’ arbiter that appeared in Fig. 6.

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. 8 (left), and a

partial specification appears in Fig. 8 (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

Complete Specification	Incomplete Specification
guarantee $\neg(\neg r_1 \wedge \neg g_1) \mathbf{U}(\neg r_1 \wedge g_1) \wedge$ $\neg(\neg r_1 \wedge \neg g_1) \mathbf{U}(\neg r_1 \wedge g_1) \wedge$ $\neg \mathbf{F}(g_1 \wedge X(\neg r_1 \wedge \neg g_1) \wedge \mathbf{X}(\neg r_1 \wedge g_1) \mathbf{U}(\neg r_1 \wedge g_1)) \wedge$ $\neg \mathbf{F}(g_2 \wedge X(\neg r_2 \wedge \neg g_2) \wedge \mathbf{X}(\neg r_2 \wedge g_2) \mathbf{U}(\neg r_2 \wedge g_2)) \wedge$ $\mathbf{G}((\neg r_1 \wedge g_1) \rightarrow \mathbf{F}((r_1 \wedge g_1) \vee \neg g_1)) \wedge$ $\mathbf{G}((\neg r_2 \wedge g_2) \rightarrow \mathbf{F}((r_2 \wedge g_2) \vee \neg g_2)) \wedge$ $\mathbf{G}(r_1 \rightarrow \mathbf{F} g_1) \wedge$ $\mathbf{G}(r_2 \rightarrow \mathbf{F} g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$	guarantee $\mathbf{G}(r_1 \rightarrow \mathbf{F} g_1) \wedge$ $\mathbf{G}(r_2 \rightarrow \mathbf{F} g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$

Fig. 8: LTL specification for full arbiter of 2 clients.

appear in Figs. 9a and 9b respectively. On the other hand, starting from the partial specification, after four iterations the non-vacuous synthesis we get is as shown in Fig. 9c, which again satisfies the full specification.

Fig. 9: Synthesized arbiters of complete and incomplete specifications of full arbiter as read in Fig. 8.

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 [11]. The complete and incomplete specification of a ‘Pnueli’ arbiter of two clients is shown in Fig. 10. 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. 11b. The result of our non-vacuous synthesis from

Complete Specification	Incomplete Specification
assume $\neg r_1 \wedge \neg r_2 \wedge$ $\mathbf{G}((r_1 \wedge \neg g_1 \rightarrow \mathbf{X} r_1) \wedge (\neg r_1 \wedge g_1 \rightarrow \mathbf{X} \neg r_1)) \wedge$ $\mathbf{G}((r_2 \wedge \neg g_2 \rightarrow \mathbf{X} r_2) \wedge (\neg r_2 \wedge g_2 \rightarrow \mathbf{X} \neg r_2)) \wedge$ $\mathbf{G} \mathbf{F}(\neg r_1 \vee \neg g_1) \wedge$ $\mathbf{G} \mathbf{F}(\neg r_2 \vee \neg g_2)$	assume $\neg r_1 \wedge \neg r_2 \wedge$ $\mathbf{G}((r_1 \wedge \neg g_1 \rightarrow \mathbf{X} r_1) \wedge (\neg r_1 \wedge g_1 \rightarrow \mathbf{X} \neg r_1)) \wedge$ $\mathbf{G}((r_2 \wedge \neg g_2 \rightarrow \mathbf{X} r_2) \wedge (\neg r_2 \wedge g_2 \rightarrow \mathbf{X} \neg r_2)) \wedge$ $\mathbf{G} \mathbf{F}(\neg r_1 \vee \neg g_1) \wedge$ $\mathbf{G} \mathbf{F}(\neg r_2 \vee \neg g_2)$
guarantee $\neg g_1 \wedge \neg g_2 \wedge$ $\mathbf{G}(((\neg r_1 \wedge \neg g_1) \rightarrow \mathbf{X} \neg g_1) \wedge ((r_1 \wedge g_1) \rightarrow \mathbf{X} g_1)) \wedge$ $\mathbf{G}(((\neg r_2 \wedge \neg g_2) \rightarrow \mathbf{X} \neg g_2) \wedge ((r_2 \wedge g_2) \rightarrow \mathbf{X} g_2)) \wedge$ $\mathbf{G} \mathbf{F}(r_1 \leftrightarrow g_1) \wedge$ $\mathbf{G} \mathbf{F}(r_2 \leftrightarrow g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$	guarantee $\neg g_1 \wedge \neg g_2 \wedge$ $\mathbf{G}(((\neg r_1 \wedge \neg g_1) \rightarrow \mathbf{X} \neg g_1) \wedge ((r_1 \wedge g_1) \rightarrow \mathbf{X} g_1)) \wedge$ $\mathbf{G}(((\neg r_2 \wedge \neg g_2) \rightarrow \mathbf{X} \neg g_2) \wedge ((r_2 \wedge g_2) \rightarrow \mathbf{X} g_2)) \wedge$ $\mathbf{G} \mathbf{F}(r_1 \rightarrow g_1) \wedge$ $\mathbf{G} \mathbf{F}(r_2 \rightarrow g_2) \wedge$ $\mathbf{G} \neg(g_1 \wedge g_2)$

Fig. 10: LTL specification for a ‘Pnueli’ arbiter of two clients. The partial specification on the right lacks the right-to-left implication in the 4th and 5th lines of the guarantee.

the partial specification again satisfies the full specification, as shown in Fig. 11c, and is synthesised in one step. This system also satisfies the specification in a less vacuous way than the system synthesised from the complete specification. In fact, in this case, if the input to our tool is a *complete* specification, the result is also the system in Fig. 11c.

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

(a) Complete specification (b) Partial specification (c) Non-vacuous synthesis from partial specification

Fig. 11: Synthesized arbiters of complete and incomplete specifications of a ‘Pnueli’ arbiter as read in Fig. 10.

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 for others to try and improve.

Acknowledgments This work was supported by the TU Graz LEAD project “Dependable Internet of Things in Adverse Environments” and the Austrian Science Fund (FWF) under the RiSE National Research Network (S11406). 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.

References

1. Armoni, R., Fix, L., Flaisher, A., Grumberg, O., Piterman, N., Tiemeyer, A., Vardi, M.Y.: Enhanced vacuity detection in linear temporal logic. In: CAV (2003)
2. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in ACTL formulas. Formal Methods in System Design 18(2) (2001)

3. Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T.A., Hofferek, G., Jobstmann, B., Könighofer, B., Könighofer, R.: Synthesizing robust systems. *Acta Inf.* 51 (2014)
4. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: *CAV* (2009)
5. Bustan, D., Flaisher, A., Grumberg, O., Kupferman, O., Vardi, M.: Regular vacuity. In: *CHARME* (2005)
6. Chechik, M., Gurfinkel, A.: Extending extended vacuity. In: *FMCAD* (2004)
7. Chockler, H., Gurfinkel, A., Strichman, O.: Beyond vacuity: towards the strongest passing formula. *Formal Methods in System Design* 43(3) (2013)
8. Church, A.: Logic, arithmetics, and automata. In: *ICM* (1963)
9. Finkbeiner, B., Schewe, S.: Bounded synthesis. *International Journal on Software Tools for Technology Transfer* 15(5) (2012)
10. Jacobs, S., Bloem, R., Brenguier, R., Könighofer, R., Pérez, G.A., Raskin, J., Ryzhyk, L., Sankur, O., Seidl, M., Tentrup, L., Walker, A.: The second reactive synthesis competition. In: *SYNT* (2015)
11. Jobstmann, B., Staber, S., Griesmayer, A., Bloem, R.: Finding and fixing faults. *Journal of Computer and System Sciences* 78(2) (2012)
12. Khalimov, A., Jacobs, S., Bloem, R.: PARTY parameterized synthesis of token rings. In: *CAV* (2013)
13. Kupferman, O., Vardi, M.: Vacuity detection in temporal model checking. *Journal on Software Tools For Technology Transfer* 4(2) (2003)
14. Namjoshi, K.S.: An efficiently checkable, proof-based formulation of vacuity in model checking. In: *CAV* (2004)
15. Pnueli, A.: The temporal logic of programs. In: *FOCS* (1977)
16. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: *POPL* (1989)
17. Samanta, R., Deshmukh, J.V., Chaudhuri, S.: Robustness analysis of networked systems. In: *VMCAI* (2013)
18. Vardi, M., Wolper, P.: Reasoning about infinite computations. *Information and Computation* 115(1) (1994)