Behavior Minimization Feedback for Unrealizable Specifications

ABSTRACT

At their early stages system specifications are usually unrealizable, in other words, there is no system that can be built to satisfy its goals under the assumed environment conditions. The cause for unrealizability can be boiled down to the fact that stated goals are too strong, the assumptions are too weak, or (commonly) a subtle combination of both. Providing engineers feedback that allows to understand the cause for unrealizability is necessary if specifications are to be evolved into realizable ones.

In this paper, we propose a technique that minimizes the behavior of an unrealizable GR(1) specification while preserving non-realizability, thus allowing engineers to focus on core behavior that causes unrealizability. Our technique *differs* from existing ones in two ways. The approach produces a semantic minimization, reducing allowable behavior while preserving unrealizability. The result is a model that preserves the cause for unrealizability in the sense that the environment's strategy for beating the system in the minimised model is also a winning strategy in the original model. Second, it assumes a automata-based description of the environment rather than an LTL specification.

ACM Reference Format:

. 2018. Behavior Minimization Feedback for Unrealizable Specifications . In *Proceedings of International Conference on Software Engineering, Gothenburg, Sweden, May 27 - 3 June 2018 (ICSE'18),* 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Requirements are naturally split between goals the system-to-be is required to achieve and assumptions that the system-to-be can rely on to fulfil its goals[24, 35]. The question to be asked then is one of realizability: Is it possible to build a system that can monitor its environment and react through its actuators in order to guarantee its goals as long as the environment fulfils the assumptions?

At early stages specifications are usually unrealizable [35]. There can be multiple causes for unrealizability including lack of monitorability and controllability, and over-idealised goals and assumptions. Goals in their first formulations tend to be stronger than what can be reasonably required and assumptions tend to be too weak, failing to rule out exceptional circumstances that the system cannot deal with [50].

Unfortunately, the cause for unrealizability tends to be the result of a combination of issues and is not easy to detect or understand. Providing engineers feedback that allows them to understand the cause for unrealizability is highly desired if specifications are to be evolved into realizable ones that can then be implemented.

In general the process of producing a running system from a specification is a manual and laborious process. However, in some settings this process can be done automatically, ensuring a correct-by-construction system. This is the case for systems studied by supervisory control [44], FOND planning [12] and controller synthesis [6, 39]. In these settings, given a specification in the form of assumptions and goals, an algorithm produces a strategy (that can be encoded as an automaton) that by monitoring and acting over its environment can guarantee the goals as long as the environment satisfies the assumptions.

Synthesis algorithms only produce a system strategy if the original specification is realizable. When the specification is unrealizable, the engineer is left with the onerous task of understanding why no such strategy exists and, only after that, with the task of changing the specification. *In this paper, we propose a novel technique that provides feedback on unrealizable specifications.*

Techniques for providing feedback on unrealizability have been proposed [2, 30, 47]. These assume that the specification is given in an expressive subset of Linear Temporal Logic (LTL), called GR(1), and return a minimal subset of these that is still unrealizable. In other words, and in the spirit of unsatisfiable cores [48], the technique provides a *syntactic minimisation that preserves unrealizability*, allowing engineers to focus on a portion of the original specification to identify the causes for unrealizability. Syntactic minimisation of specifications is not adequate when the specification is not provided (in its entirety) in a declarative form. For instance, specifications using automata-based descriptions of the environment (using for instance labelled transition systems [25] as in [37], statecharts [21] as in [49], and process algebra [23, 41] as in [17]) would require a non-trivial property extraction and/or a very large sets of formulae encoding every state transition.

The technique we propose in this paper *differs* from existing ones in two ways. First and foremost, the approach produces a semantic minimization on the search space, reducing allowable behavior while preserving unrealizability. The result is a semantically minimized model in which the cause for unrealizability also holds in the original specification. Second, it assumes an automata-based description of the environment rather than an LTL specification.

Our intent is to find semantic minimization that preserves the cause of unrealizability. *What does this mean?* A specification is one that dictates a game in which one player, the environment, tries to satisfy the assumptions while preventing the other player, the system-to-be, from achieving its goals. If the specification is unrealizable, it means that the environment has a playing strategy that always beats its opponent. In other words, no matter what the system-to-be does, by monitoring the environment and controlling its outputs, it cannot achieve its goals even if the environment behaves according to its assumptions. Reducing the specification

ICSE'18, May 27 - 3 June 2018, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

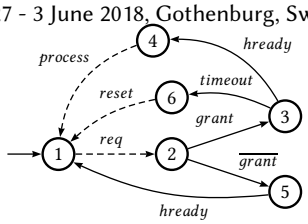

Figure 1: Bus Access example (E)

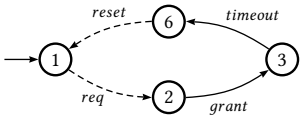

Figure 2: Minimized Bus Access (E_1)

while preserving its causes of unrealizability leads to a new specification that describes less behavior and in which the winning strategy of the environment also works in the original specification.

The technique has also the potential to complement existing techniques that assume a specification only in LTL form and that work by syntactic, rather than semantic, minimization.

The paper is structured as follows. We first provide a motivating example as an informal overview of the proposed technique, we then present some preliminary definitions, namely to define formally a control problem and realizability, and then go on to define minimization and non-realizability preservation. In Section 5 we present the minimization algorithm and then discuss validation of our technique. We then proceed with a discussion on related work and conclusions.

2 MOTIVATION

We motivate and informally introduce relevant concepts by discussing a small example inspired by [42]. Suppose an engineer is writing the specification for an industrial controller that needs to coordinate communication between various devices that consume data from different sensors via a shared bus according to the scheduling policy defined by an arbiter. A device acting as a master will raise a bus access requirement (*req*) to communicate with a specific sensor that will act as a slave. The arbiter can then either grant access to the device (*grant*) or give the bus to another master ($\overline{grant}$). For the latter case, the device denied permission will have to wait for the slave currently using the bus to raise a ready signal (*hready*) indicating that it has finished operation. Only then will the device issue another requirement. If access is granted then the device will wait either for a slave confirmation (*hready*) or a *timeout* event. After the slave has finished operation the device will start to process the received information (*process*). Should the slave issue a timeout message, a reset will be triggered (*reset*) to try and restore normal operation.

The initial environment specification E, including safety assumptions and safety goals, is presented in Figure 1, where solid lines indicate monitored events and dotted lines indicate controlled events. Monitored events are those that the environment can trigger while controlled events are triggered by the system.

The goal of continually processing data from the board is captured by the LTL liveness formula $\square\diamond process$. The assumption that access to the bus will always eventually be granted to the device is captured by the liveness formula $\square\diamond grant$. Note that without further assumptions the environment is able to systematically produce *timeout* events even after granting the device access to the bus. These two choices, picking *grant* at state 2 and then *timeout* at state 3 conform a winning strategy for the environment. In other words, no matter what the system does, the environment, by making these two choices always prevents the system goals from being achieved.

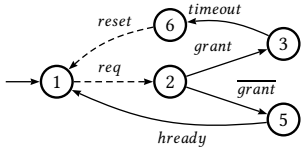

Figure 3: Minimized Bus Access (E_2)

The specification E_1 in Figure 2 is a subgraph of E, describing an environment in which strictly less behavior than that of E can occur. Note that the environment strategy for specification E_1 is the same strategy the environment uses in E to prevent the system from achieving its goals. Indeed, E_1 cannot be further reduced while satisfying this (environment strategy preserving) property. Thus, E_1 can be thought of as a slice of the original environment specification that has removed behavior that is irrelevant with respect to a cause for unrealizability.

Imagine that after analyzing this first unrealizability cause (captured by E_1) the engineer adds another assumption forcing the environment to produce *hready* infinitely often ($\square\diamond hready$) hoping to reach state 4, thus realizing the goal. It turns out that even after adding this new assumption the specification remains unrealizable: The environment can avoid state 4 if it keeps track of the most recently satisfied assumption. It can first grant access to the device and then produce a *timeout*. This will be the first cycle of its strategy, then it will deny access ($\overline{grant}$) and start over again. This alternating strategy will allow the environment to fulfill its assumptions (producing *grant* in the first cycle and *hready* in the second) while avoiding to produce the expected outcome. The sub LTS capturing this behavior is presented in Figure 3 and represents a minimal slice of the original specification E that captures the winning strategy of the environment.

Note that an LTS representation of the winning strategy for the environment (see Figure 4) is not a sub-LTS of E as the strategy must remember which assumption was the last to be satisfied, to focus on the next. In general, the worst case is that the strategy of the environment will have n times more states than the minimal subLTS that can be used to generate it, where n is the number of liveness assumptions that the environment must satisfy.

In the next sections we will show how to automatically build a minimal sub-LTS that preserves a winning strategy for the environment (which is a cause of unrealizability).

3 PRELIMINARIES

Definition 3.1. (Labelled Transition Systems) A *Labelled Transition System* (LTS) is $E = (S, \Sigma, \Delta, s_0)$, where S is a finite set of

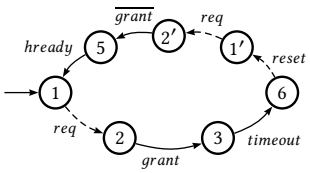

Figure 4: Bus Access counter-strategy.

states, $\Sigma \subseteq Act$ is its *communicating alphabet*, $\Delta \subseteq (S \times \Sigma \times S)$ is a transition relation, and $s_0 \in S$ is the initial state. We denote $\Delta(s) = \{a \mid (s,a,s') \in \Delta\}$. An LTS is deterministic if (s,a,s') and (s,a,s'') are in Δ implies $s' = s''$. An execution of E is a word $\varepsilon = s_0, a_0, s_1, \ldots$ where $(s_i, a_i, s_{i+1}) \in \Delta$. A word π is a trace (induced by ε) of E if its the result of removing every s_i from an execution ε of E. We denote the set of infinite traces of E by $Tr(E)$.

We use linear temporal logics of fluents (FLTL) is that it provides a uniform framework for specifying state-based temporal properties in event-based models [19]. A *fluent fl*is defined by a pair of sets and a Boolean value: $\text{fl} = \langle I_{\text{fl}}, T_{\text{fl}}, Init_{\text{fl}} \rangle$, where $I_{\text{fl}} \subseteq Act$ is the set of initiating actions, $T_{\text{fl}} \subseteq Act$ is the set of terminating actions and $I_{\text{fl}} \cap T_{\text{fl}} = \emptyset$. A fluent may be initially *true* or *false* as indicated by $Init_{\text{fl}}$. Every action $\ell \in Act$ induces a fluent, namely $\dot{\ell} = \langle \ell, Act \setminus \{\ell\}, false \rangle$. Finally, the alphabet of a fluent is the union of its terminating and initiating actions.

Let $\mathcal{F}$ be the set of all possible fluents over Act. An FLTL formula is defined inductively using the standard Boolean connectives and temporal operators X (next), U (strong until) as follows: $\varphi ::= fl \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi U \psi$, where $fl \in \mathcal{F}$. As usual we introduce $\wedge$, $\Diamond$ (eventually), and $\Box$ (always) as syntactic sugar. Let Π be the set of infinite traces over Act. The trace $\pi = \ell_0, \ell_1, \ldots$ satisfies a fluent Fl at position i, denoted $\pi, i \models Fl$, if and only if one of the following conditions holds:

- $Init_{Fl} \wedge (\forall j \in \mathbb{N} \cdot 0 \leq j \leq i \rightarrow \ell_j \notin T_{fl})$
  $\exists j \in \mathbb{N} \cdot (j \leq i \wedge \ell_j \in I_{fl}) \wedge (\forall k \in \mathbb{N} \cdot j < k \leq i \ \rightarrow \ell_k \notin T_{fl})$

We will assume that FLTL formulae are equipped with the corresponding fluent definition triples. Formula satisfaction for FLTL is standard and omitted.

In this paper we restrict attention to GR(1) [7] formulas, that is, properties of the form $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond\phi_i \implies \bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$, where $\{\phi_1 \ldots \phi_n\}$ and $\{\gamma_1 \ldots \gamma_m\}$ are propositional FLTL formulae.

We now provide a standard definition of parallel composition of LTSs inspired from [22].

Definition 3.2. (Parallel Composition) Let $P = (S_P, \Sigma_P, \Delta_P, s_{0_P})$ y $Q = (S_Q, \Sigma_Q, \Delta_Q, s_{0_Q})$ be two LTSs, then the parallel composition $P \| Q$ is defined as $P \| Q = (S_P \times S_Q, \Sigma_P \cup \Sigma_Q, \Delta_{P\|Q}, (s_{0_P}, s_{0_Q}))$ where $\Delta_{P\|Q}$ is the smallest relation satisfying:

$$((s,s'),a,(t,s')) \in \Delta_{P\|Q} \iff (s,a,t) \in \Delta_P \wedge a \notin \Sigma_Q$$
$$((s,s'),b,(s,t')) \in \Delta_{P\|Q} \iff (s',b,t') \in \Delta_Q \wedge b \notin \Sigma_P$$
$$((s,s'),c,(t,t')) \in \Delta_{P\|Q} \iff (s,c,t) \in \Delta_P \wedge (s',c,t') \in \Delta_Q$$

The distinction between what an LTS can control and what it can monitor is enforced through the notion of *legal environment* taken from [17], and inspired in that of Interface Automata [13]. Intuitively, a controller does not block the actions that it does not control, and dually, the environment does not restrict controllable actions.

Definition 3.3. (Legal Environment) Given $M = (S_M, \Sigma_{M_c} \uplus \Sigma_{M_u}, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma_{P_c} \uplus \Sigma_{P_u}, \Delta_P, s_{P_0})$ LTSs. We say that M is a legal environment for P if the composition of the interface automata $M' = (S_M, s_{M_0}, \Sigma_{M_u}, \Sigma_{M_c}, \emptyset, \Delta_M)$ and $P' = (S_P, s_{P_0}, \Sigma_{P_u}, \Sigma_{P_c}, \emptyset, \Delta_P)$ has no illegal states [13].

We now formally define the control problem for GR(1) formulas.

Definition 3.4. (LTS GR(1) Control) Given a specification for a problem domain in the form of an environment LTS $E = (S, \Sigma, \Delta, s_0)$, a set of controllable actions C, and a GR(1) formula φ, the solution for the LTS control problem $\mathcal{I} = \langle E, C, \varphi \rangle$ is to find an LTS M such that M is a legal environment for E , $E \| M$ is deadlock free, and for φ and for every trace π in $M \| E$ the following holds: if $\pi \models \varphi$.

Definition 3.5. (Two-player Game) A two player game G is defined as the tuple $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, \varphi)$ where S_g is a finite set of states, Γ^-,$\Gamma^+ \subseteq S \times S$ are transition relations defined for the uncontrollable and controllable transtiions respectively, $s_{g0} \in S_g$ is the initial condition and $\varphi \subseteq S_g^{\omega}$ is the winning condition. We denote $\Gamma^-(s) = \{s' | (s,s') \in \Gamma^-\}$ and in a similar fashion for $\Gamma^+(s)$. A state s is uncontrollable if $\Gamma^-(s) \neq \emptyset$ and controllable otherwise. A play on G is a sequence $p = s_0, s_1, \ldots$ an a play p ending in s_{g_n} is extended by the controller choosing a subset $\gamma \subseteq \Gamma^+(s_{g_n})$. Then the environment chooses a state $s_{g_{n+1}} \in \gamma \cup \Gamma^-(s_{g_n})$ and adds $s_{g_{n+1}}$ to p.

Definition 3.6. (Strategy with memory) A strategy with memory Ω for the controller is a pair of functions (δ, u) where Ω is some memory domain with designate start value ω_0, $\delta : \Omega \times S \rightarrow 2^S$ such that $\delta(\omega, s) \subseteq \Gamma^+(s)$ and $u : \Omega \times S \rightarrow \Omega$.

Definition 3.7. (Consistency under Controllability and Winning Strategy) A finite or infinite play $p = s_0, s_1, \ldots$ is consistent under controllability if for every n we have $s_{n+1} \in \delta(\omega_n, s_n)$, where $\omega_{i+1} = u(\omega_i, s_{i+1})$ for all $i \geq 0$. A strategy (δ, u) for controller from state s is winning if every maximal play starting in s and consistent under controllability with (δ, u) is infinite and remains within φ. We say that the controller wins the game G if it has a winning strategy from the initial state.

Definition 3.8. (Generalized Reactivity(1)) Given an infinite sequence of states p, let $inf(p)$ denote the states that occur infinitely often in p, let $\phi_1, \ldots, \phi_n$ and $\gamma_1, \ldots, \gamma_m$ be subsets of S. Let $gr((\phi_1, \ldots, \phi_n), (\gamma_1, \ldots, \gamma_m))$ denote the set of infinite sequences p such that either for some i we have $inf(p) \cap \phi_i = \emptyset$ or for all j we have $inf(p) \cap \gamma_j \neq \emptyset$. A GR(1) game is a game where the winning condition φ is $gr((\phi_1, \ldots, \phi_n), (\gamma_1, \ldots, \gamma_m))$.

Definition 3.9. (Counter-strategy with memory) A counter strategy with memory ∇ for the environment is a pair of functions (κ, υ) where ∇ is some memory domain with designate start value ∇_0, $\kappa : \nabla \times S \rightarrow 2^S$ such that $\kappa(\nabla, s) \subseteq \Gamma^-(s)$ and $\upsilon : \nabla \times S \rightarrow \nabla$.

Definition 3.10. (Consistency under Non-Controllability and ~~Winning Strategy for the Environment~~) A finite or in play $p = s_0, s_1, \ldots$ is consistent under non-controllability if for every n we have $s_{n+1} \in \kappa(\nabla_n, s_n)$, where $\nabla_{i+1} = \upsilon(\nabla_i, s_{i+1})$ for all $i \geq 0$. A strategy (κ, υ) for environment from state s is winning if every maximal play starting in s and consistent under non-controllability with (κ, υ) is infinite and remains exits φ at least at some point. We

say that the environment wins the game G if it has a winning strategy from the initial state.

We convert the GR(1) LTS control problem $\mathcal{I} = \langle E, C, \varphi \rangle$ we construct a GR(1) game $G = (S_g, \Gamma-, \Gamma+, s_{g0}, \varphi)$ such that every state in S_g encodes a state in E and a valuation of all fluents appearing in φ. We build S_g from E in such a way that states in the game encode a state in E and truth values for all fluents in φ, let $S_g = S_e \times \Pi_{i=0}^{k}\{true, false\}$. Consider state $s_g = (s_e, \alpha_1, \ldots, \alpha_k)$, given fluent fl_i we say that s_g satisfies fl_i if and only if α_i is $true$. We generalize satisfaction to boolean combination fo fluents in the natural way. We build the transition relations $\Gamma-,\Gamma+$ through the following rules. Consider state $s_g = (s_e, \alpha_1, \ldots, \alpha_k)$, for every transition $(s_e, l, s'_e) \in \Delta$ we include $(s_g, (s'_e, \alpha'_1, \ldots, \alpha'_k))$ in Γ^{β} where β is $+$ if $l \in C$ and β is $-$ otherwise. α'_i is α_i if $l\neg \in I_{fl_i} \cup T_{fl_i}$, α'_i is $true$ if $l \in I_{fl_i}$ and $false$ if $l \in T_{fl_i}$. s_{g0} is $(s_0, Init_{fl_1}, \ldots, Init_{fl_k})$. Let $\varphi_g \subseteq S_g^{\omega}$ be the set of sequences that satisfy $gr((\phi_1, \ldots, \phi_n), (\gamma_1, \ldots, \gamma_m))$, it follows that $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, \varphi_g)$ is a GR(1) game.
It can be shown that if there is a solution to an SGR(1) LTS control problem then there is a winning strategy for a controller in the constructed GR(1) game, refer to Theorem 3.3 in [15]. We will refer to the game constructed from control problem $\mathcal{I} = \langle E, C, \varphi \rangle$ as $G(\mathcal{I})$.

4 BEHAVIOR MINIMIZATION FEEDBACK

We first formalize the problem we aim to solve and then discuss a solution for it. Assume an unrealizable specification of the form $< E, C, \varphi >$ where E is an LTS that describes safety assumptions and guarantees, C describes the set of events that the system can control, and where φ is a GR(1) property that encodes liveness assumptions and guarantees. Our aim, informally, is to produce automatically a reduced specification that preserves a cause for unrealizability of the original specification. In this sections, we first describe more formally what it means for a specification to preserve unrealizability and then explain how such a specification can be produced automatically. If $\mathcal{I} =< E, C, \varphi >$ is a non realizable control problem then there is no LTS M that is a legal and deadlock-free environment with respect to E such that $E||X \models \varphi$.

In [15] the author shows that a control problem is realizable if and only if there is a winning strategy for the system in the game constructed following his proposed conversion. The same work shows that GR(1) games (as a subset of LTL games) are determined. Thus, we know that a control problem is unrealizable if and only if there exists a winning strategy for the environment. We will refer to the latter as the counter strategy. For the sake of simplicity and following the assumption also introduced in [15] we will work with plants with no mixed state (i.e. states where monitored and controllable transitions are both enabled). When mentioning the game related to a control problem we will be referring to the conversion presented there [15] in section 3.5.1 and it will noted as $G(\mathcal{I})$. We aim to build a minimal unrealizable specification $\mathcal{I}' =< E', C, \varphi >$ such that any winning strategy for the environment in $G(\mathcal{I}')$ is also a winning strategy for the environment in $G(\mathcal{I})$. We define minimality over the partial order defined by a notion inspired from sub-graph.

Definition 4.1. (Sub-LTS) Given $M = (S_M, \Sigma_M, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma_P, \Delta_P, s_{P_0})$ LTSs, we say that P is a sub-LTS of M (noted $P \subseteq M$) if $S_P \subseteq S_M$, $s_{M_0} = s_{P_0}$, $\Sigma_P \subseteq \Sigma_M$ and $\Delta_P \subseteq \Delta_M$.

When referring to the plant and its sub-LTSs an analogous relation holds between the games constructed following [15] conversion. Since the game is built as the product of the plant with the valuations of the fluents if $S_P \subseteq S_M, \Delta_P \subseteq \Delta_M$ then $S_{G(\mathcal{I}_P)} \subseteq S_{G(\mathcal{I}_M)}$ and $\Delta_{G(\mathcal{I}_P)} \subseteq \Delta_{G(\mathcal{I}_M)}$ should hold. We formally define an unrealizability preserving control problem as follows:

Definition 4.2. (Counter strategy Preservation) Given a control problem $\mathcal{I} =< E, C, \varphi >$ we say that $\mathcal{I}' =< E', C, \varphi >$ preserves counter strategy if for every winning strategy for the environment in $G(\mathcal{I}')$ then it is also winning in $G(\mathcal{I})$.

We first show that to preserve unrealizability it is sufficient to satisfy inclusion of E' in E and require a notion of controllability preservation for E' w.r.t. E and C. We will use the term alternation when referring to this notion. The inclusion requirement is self explanatory since we want to stick with the behavior as originally specified, alternation retains the asymmetry of choice when computing the counter strategy, not restricting system choices that would allow the environment to falsify φ under conditions not necessarily present in E. No new deadlock should be introduced by the minimization, since in the control problem scenario deadlocks are winning for the environment. These conditions are captured in the Alternating Sub-LTS definition (4.3).

Definition 4.3. (Alternating Sub-LTS) Given $M = (S_M, \Sigma, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma, \Delta_P, s_{P_0})$ LTSs, where $\Sigma = C \cup \mathcal{U}, C \cap \mathcal{U} = \emptyset$ and $P \subseteq M$. We say that P is a alternating sub-LTS of M, noted as $P \sqsubseteq_{(C)} M$, if $\forall s_P \in S_P$ the following holds: $\Delta_P(s_P) \cap C = \Delta_M(s_P) \cap C$ and $(\Delta_M(s_P) \neq \emptyset \rightarrow \Delta_P(s_P) \neq \emptyset)$.

The Alternating Sub-LTS relation defines a partial order inducing a semi-lattice where the original plant E lies at the top and the minimal (non-empty) Alternating Sub-LTSs appear as the bottom leaves.

Since we want to show that a counter strategy is preserved by our minimization we need to prove that alternating plants correlate with a similar notion for games.

Definition 4.4. (Sub-Game) Given $G = (S_g, \Gamma^-, \Gamma^+, s_{g_0}, \varphi)$ and $G' = (S'_g, \Gamma'^-, \Gamma'^+, s'_{g_0}, \psi)$ two-player games, we say that G' is a sub-game of G, noted $G' \subseteq G$, if $S'_g \subseteq S_g$, $s'_{g_0} = s_{g_0}$, $\psi = \varphi$, $\Gamma'^- \subseteq \Gamma^-$ and $\Gamma'^+ \subseteq \Gamma^+$.

Definition 4.5. (Alternating Sub-Game) Given $G = (S_g, \Gamma^-, \Gamma^+, s_{g_0}, \varphi)$ and $G' = (S'_g, \Gamma'^-, \Gamma'^+, s'_{g_0}, \psi)$ both two-player games such that $G' \subseteq G$, we say that G' is a alternating sub-game of G, noted $G' \sqsubseteq G$, if the following holds: $\forall s'_g \in S'_g : \Gamma'^+(s'_g) = \Gamma^+(s'_g)$ and $\Gamma^-(s'_g) \cup \Gamma^+(s'_g) \neq \emptyset \rightarrow \Gamma'^-(s'_g) \cup \Gamma'^+(s'_g) \neq \emptyset$.

Lemma 4.6. (Alternating Sub-LTS implies Alternating Sub-Ga
Let $\mathcal{I}_1 = \langle E_1, C, \varphi \rangle$ and $\mathcal{I}_2 = \langle E_2, C, \varphi \rangle$ be two control problems, then if $E_2 \sqsubseteq_{(C)} E_1$ it holds that $G(\mathcal{I}_2) \sqsubseteq G(\mathcal{I}_1)$.

Proof. $\mathcal{I}_1$ and $\mathcal{I}_2$ share winning conditions so they also share fluents definition. Since $E_2 \sqsubseteq_{(C)} E_1$ implies $E_2 \subseteq E_1$ we know that $S_2 \subseteq S_1$ and then for every state in the game constructed

from the control problem holds that $S_2 \times \Pi_{i=0}^{k}\{true, false\} \subseteq S_1 \times \Pi_{i=0}^{k}\{true, false\}$ implying that $S_{g_2} \subseteq S_{g_1}$. Following $\Delta_2 \subseteq \Delta_1$ we can see that following the definition of the game $G(\mathcal{I}_1)$ constructed from $\mathcal{I}_1$ if $s_{g_1} = (s_e, \alpha_1, \ldots, \alpha_k) \in S_{g_1}$ for all $(s_e, l, s'_e) \in \Delta_1$ that transition $(s_{g_1}, (s'_e, \alpha'_1, \ldots, \alpha'_k))$ will be added to Γ_1^+ if $l \in C$ or to Γ_1^- otherwise. Since $S_{g_2} \subseteq S_{g_1}$ and $\Delta_2 \subseteq \Delta_1$ it holds that $\Gamma_2^+ \cup \Gamma_2^- \subseteq \Gamma_1^+ \cup \Gamma_1^-$. This far we have proven $G(\mathcal{I}_2) \subseteq G(\mathcal{I}_1)$. Now, if $E_2 \sqsubseteq_{(C)} E_1$ it holds that for all $s \in S_2, l \in C$ if $(s, l, s') \in \Delta_1$ then $(s, l, s') \in \Delta_2$ and also if every state in the game related to s, $s_{g_1} \in s \times \Pi_{i=0}^{k}\{true, false\}$ for all $l \in C$ such that $(s, l, s') \in \Delta_1$ we know $(s, l, s') \in \Delta_2$ and if $s'_{g_1} = (s', \alpha'_1 \ldots, \alpha'_k)$, then $(s_{g_1}, s'_{g_1}) \in \Gamma_1^+$ will hold, as will $(s_{g_1}, s'_{g_1}) \in \Gamma_2^+$, thus implying $\Gamma_1^+(s_{g_1}) = \Gamma_2^+(s_{g_1})$. For the environment actions, for all $s \in S_2$ if there exists an $l \in \Sigma$ such that $(s, l, s') \in \Delta_1$ then from $E_2 \sqsubseteq_{(C)} E_1$ it must exists an $l' \in \Sigma$ such $(s, l', s'') \in \Delta_2$. It follows that with $s_{g_1} \in s \times \Pi_{i=0}^{k}\{true, false\}$, l, and $s'_{g_1} = (s', \alpha'_1 \ldots, \alpha'_k)$ we have $(s_{g_1}, s'_{g_1}) \in \Gamma_1^+ \cup \Gamma_1^-$ and for l' in Δ_2, let $s''_{g_1} = (s'', \alpha''_1 \ldots, \alpha''_k)$, we have $(s_{g_1}, s''_{g_1}) \in \Gamma_2^+ \cup \Gamma_2^-$ which implies $\Gamma_1^-(s_{g_1}) \cup \Gamma_1^+(s_{g_1}) \neq \emptyset \rightarrow \Gamma_2^-(s_{g_1}) \cup \Gamma_2^+(s_{g_1}) \neq \emptyset$.

□

We present the following lemma to show that if a plant E' satisfies the inclusion and alternating properties for the original plant E then a non realizability cause is preserved.

Lemma 4.7. (Alternating Sub-LTS preserves counter strategy) *Let P and M be two LTSs s.t. $P \sqsubseteq_{(C)} M$, if there exists $f_1^{\neg\varphi}$ a winning strategy for the environment in the game constructed from $G(\mathcal{I}_P)$ then $f_1^{\neg\varphi}$ is winning for the environment in $G(\mathcal{I}_M)$.*

Proof sketch. In terms of available options, the difference between M and P (remeber that $P \sqsubseteq_{(C)} M$) is that in M the environment has potentially more options to choose from. So, if it already had a strategy within a subset of such options, it is clear that the same strategy can be applied over M, since P did not take any viable option away from the system (as an opposite player), so it can not win in M if it could not win in P. □

Proof. By way of contradiction suppose that P is Alternating Sub-LTS of M and C and that there exists a counter strategy in $G(\mathcal{I}_P)$ but not winning in $G(\mathcal{I}_M)$.
Let π be a play $s_0 s_1 \ldots$ consistent under non controllability with the counter strategy that wins in $G(\mathcal{I}_P)$ but loses in $G(\mathcal{I}_M)$. π leading to a finite win in $G(\mathcal{I}_P)$ at $s_\perp$ (not a finite state in M) will be prevented by the alternation requirement where $\forall s_P \in S_P : |\Delta_M(s_p)| > 0 \rightarrow |\Delta_P(s_p)| > 0$ (implying $\forall s_P\prime = (s_P, \alpha_1, \ldots, \alpha_k) \in S_{G_P} : |\Gamma_M^\beta(s_P\prime)| > 0 \rightarrow |\Gamma_P^\beta(s_P\prime)| > 0$). If the environment has an infinite win in $G(\mathcal{I}_P)$ not feasible in $G(\mathcal{I}_M)$ and since $G(\mathcal{I}_P) \subseteq G(\mathcal{I}_M)$ it follows that the system was able to take the play out of the domain of the counter strategy. In order to accomplish this a state s_a must be reached in $\pi = s_0 \ldots s_a \ldots$ where the environment can no longer play according to the counter strategy. The offending move has to be performed by the system, since the environment would otherwise play according to his winning strategy. If $s_a \in S_P$ and s_a is controllable we know that under alternation $\Gamma_P^+(s_a) = \Gamma_M^+(s_a)$, i.e., all controllable options are preserved at s_a. If this is the case the strategy should hold since it wins in P against every system choice for the preserved states. If $s_a \in S_{G_M} \setminus S_{G_P}$ there should exist s_b the last state before s_a ($\pi = s_0 \ldots s_b \ldots s_a \ldots$) where the play stepped out of S_{G_P}. If s_b is monitored it would keep playing according to the counter strategy, so we can assume that s_b is controllable, but if this is the case, again, since all the controllable options are preserved in $G(\mathcal{I}_P)$ and the winning strategy holds, by definition, against every system choice, the environment should be able to keep playing accordingly, not stepping out of $G(\mathcal{I}_P)$. □

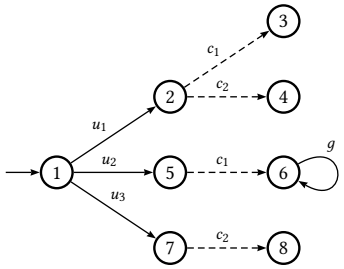

Figure 5: Several conflicts example (E).

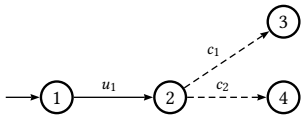

Figure 6: Several conflicts example minimization(E_1).

We can now formally define the problem we aim to resolve:

Definition 4.8. (Problem Statement) Given a unrealizable control problem $\mathcal{I} =< E, C, \varphi >$, find an unrealizable control problem $\mathcal{I}' =< E', C, \varphi >$ such that $E' \sqsubseteq_{(C)} E$, $\mathcal{I}'$ is unrealizable $\mathcal{I}$, and that there is no $\mathcal{I}'' =< E'', C, \varphi >$ such that $E' \subseteq E''$ and $\mathcal{I}''$ is unrealizable.

Since following lemma 4.7 alternating sub-LTSs preserve counter strategies we know that any counter strategy in $G(\mathcal{I}')$ will hold in $G(\mathcal{I})$.

5 A BEHAVIOR MINIMIZATION PROCEDURE

In order to find a minimal representation of the non realizability problem in terms of $\mathcal{I}$, we expect our procedure to incrementally reduce a plant while checking if a local minimum has been reached. We introduce the notion of closest alternating sub LTS:

Definition 5.1. (Closest Alternating Sub-LTS) Given $P = (S_P, \Sigma, \Delta_P, s_{P_0})$ and $M = (S_M, \Sigma, \Delta_M, s_{M_0})$ LTSs, we say that P is a closest Alternating Sub-LTS of M if $P \sqsubseteq_{(C)} M$ and the following holds: $\nexists Q : P \neq Q, P \sqsubseteq_{(C)} Q \sqsubseteq_{(C)} M$.

We will explore the semi-lattice defined by the partial order induced by the alternating sub-LTS to find potential minimizations. We will start from a given plant and move towards its closest Alternating Sub-LTS neighbors.

If we take the example of the figure 5 where the system has to satisfy the liveness goal of achieving g infinitely often (expressed by the LTL formula $\square\diamond g$) with $\Sigma = \{u_1, u_2, u_3, c_1, c_2, g\}$ and $C = \{c_1, c_2, g\}$, the original plant E will stay at the top of the semi lattice

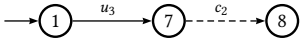

Figure 7: Several conflicts example minimization(E_2).

while E_1 (depicted in figure 6) and E_2 (depicted in figure 7) will be at the bottom. In between these we will find the Alternating Sub-LTSs obtained by removing just one monitored transition. The automaton obtained by removing $\{u_1, u_3\}$ is not shown since it is an Alternating Sub-LTS but does not preserve non realizability.

We introduce the following lemma showing that Alternating Sub-LTSs are closed under realizability.

Lemma 5.2. *(*Alternating Sub-LTSs preserve realizability*) Let E be the plant for the non realizable control problem $\mathcal{I} =< E, C, \varphi >$, and E_1, E_2 two LTSs s.t. $E_2 \sqsubseteq_{(C)} E_1 \sqsubseteq_{(C)} E$ and $\mathcal{I}_1 =< E_1, C, \varphi >$, $\mathcal{I}_2 =< E_2, C, \varphi >$, if $\mathcal{I}_1$ is realizable then $\mathcal{I}_2$ is also realizable.*

Proof sketch. By way of contradiction, if $\mathcal{I}_1$ is realizable but $\mathcal{I}_2$ is not, then the strategy for the environment on E_2 should not hold on E_1. But since we know that $E_2 \sqsubseteq_{(C)} E_1$, this does not hold as a consequence of lemma 4.7, hence proving the contradiction. □

Proof. If the system has a winning strategy in $G(\mathcal{I}_1)$ but no winning strategy in $G(\mathcal{I}_2)$, this would imply that the environment is able to either take the play into a deadlock state or a σ-trap that always falsifies the property φ. Suppose that the system is able to play according to the strategy up to state s_i in $G(\mathcal{I}_2)$, after this point, for every choice the system makes the environment has a way to construct a play π that is winning for him, either finitely or infinitely. The departure at state s_i must have been introduced by restricting the system choice, removing a controllable transition which is not allowed for Alternating Sub-Games, or giving the environment more power, by adding a monitored transition or removing all of them at a purely monitored state thus introducing a deadlock which is also forbidden since by Alternating Sub-Game definition inclusion is satisfied. Is proven by contradiction that realizability is preserved under plant induction. □

We need to construct an algorithm that will yield at least one minimal E' Alternating Sub-LTS for E and C. The following implementation eagerly looks for a single conflict in the search space conformed by the aforementioned semi-lattice. The code can be explained as follows: Monitored transitions will be progressively removed from the plant under minimization (this is what we call $\mathcal{T}_u$ or candidate set), the closest Alternating Sub-LTS will be computed from the resulting partial structure and a one step back track will be fired every time a realizable Alternating Sub-LTS is found. A minimal Alternating Sub-LTS will be found when no further reduction is available.

Lemma 5.2 shows that we can keep eagerly removing elements from a set $\mathcal{T}_u$ of candidate monitored transitions, reducing E to its closest Alternating Sub-LTS. Once we can not, while preserving non-realizability, remove anything else we have reached an Alternating Sub-LTS that is also minimal. This follows from the fact that no closest Alternating Sub-LTS exists bellow it that preserves

```
1  algorithm dfs_min is
2        input: E the LTS representing the original plant
3        output: E' the LTS that satisfies the problem statement
4        E_new = E' = E
5        T_u = E.monitored_transitions() (A)
6        to_remove = [] (B)
7        while(|T_u| > 0)
8              t = T_u.pop()
9              while(¬E'.contains(t) or d_out(E',t) == 1)
10                   if (|T_u| == 0)
11                         return E_K
12                   t = T_u.pop() (C)
13             to_remove.add(t)
14             E_new = E.remove(to_remove) (D)
15
16             is_realizable = E_new.realizable() (E)
17
18             if (¬ is_realizable):
19                   E' = E_new
20             else
21                   to_remove.remove(t) (F)
22       return E' (G)
```

Figure 8: Eager single-conflict exploration

non realizability. If all the closest Alternating Sub-LTSs in its neighborhood are realizable the current plant is minimal. We can now describe the eager algorithm as presented in picture 8.

The algorithm initializes the set of candidate transitions for removal at (A) and the effective list to be removed at each iteration `to_remove` at (B). The latter keeps track of the transitions removed along our search down the space of Alternating Sub-LTSs for E. The exploration will continue until the candidate set is empty. At each cycle the set is updated (C) by removing from it those transitions that are discarded either because they were already removed by the reduction at (D) or because they would have induced a deadlock. It is important to note that such transitions will not appear later on since we are consistently reducing the plant and only performing a one step back track on the search space but never adding a transition back. The reduction, at (D), looks for the closest Alternating Sub-LTS obtained after removing transitions in `to_remove` from E. The oracle is then asked for realizability of the plant under evaluation (E_{new}) (E). If the check indicates that the plant is non realizable the candidate set and the last Alternating Sub-LTS E' are updated (since further minimization is possible), otherwise a one step back track is triggered by removing t the transition under test from the effective set `to_remove` (F). This is actually the backtracking point where the algorithm shifts the search sideways instead of downward. Once the candidate set has been throughly explored the last non realizable plant E' is returned as a minimal Alternating Sub-LTS for E and C (G).

The set $\mathcal{T}_u$ is monotonically reduced, notice that in the step (F) the transition t was removed from the candidate set but was retained in E'. What happens if it could have been removed later on by the algorithm? In such case removing t from E' and reducing the Alternating Sub-LTS E_{new} accordingly will render the control problem realizable. Suppose that the search continued preserving t and removing other candidates $t_1, \ldots, t_n$ from E' and getting the Alternating Sub-LTS $E_{1,\ldots,n}$ that still contains t while preserving

non realizability. If we were to remove t from $E_{1,\ldots,n}$ after reducing it to the closest Alternating Sub-LTS we would get a new plant E'_{new} that can be Alternating Sub-LTS for E_{new}, and from lemma 5.2 it will render the control problem realizable. We showed this way that if we are looking for just one conflict we can immediately remove the transition that yielded realizability from the candidate set without sacrificing minimality. The procedure is complete w.r.t to the problem statement because if a solution for the problem exists (for the sake of simplicity suppose that there is only one possible minimal alternating sub-LTS E' satisfying unrealizability), then the algorithm will effectively, and progressively, reduce it to E', if the problem has multiple solutions the algorithm will yield one of those. The procedure is sound w.r.t to the problem statement since at every step it holds that $E' \sqsubseteq_{(C)} E$ so the alternating condition is met. The realizability check ensures that the output keeps the control problem unrealizable, and the minimality is satisfied by checking that every closest alternating sub-LTS is realizable at the final iteration in conjunction with lemma 5.2.

We introduced two minor modifications to the algorithm in order to try and improve the time taken to compute the minimization and to potentially reduce its size. The first optimization transforms the candidate set $\mathcal{T}_u$ into a list and orders it according to the distance of its elements with respect to the initial state. The idea is to try and cut as aggressively as possible first and see if a big cut still preserves the non realizability cause. The second optimization looks for non controllable strips within the the input automaton E, that is, (linear) sequences of non controllable actions according to Δ_E. For each of these sequences only one action is added to $\mathcal{T}_u$ since, if removed from E' when looking for a minimal sub LTS all the others have to be removed as well to preserve alternation.

6 VALIDATION

The validation reported in this section is guided by two main concerns. This first is the degree to which the initial control problem is reduced and the computational cost involved. The second is if the minimization offers insight into non realizability causes.

In order to avoid bias, we prioritized using third-party developed unrealizable specifications that include explanations for the causes of unrealizability. Unfortunately, third party controller synthesis problems that are found in the literature are typically realizable. The only sources we found were from [2, 14, 31]. More specifically, we used Generalized Buffer case study from [31], the Lift Controller case study from [2] and the Mine Pump case study from [14].

The case studies taken from [2, 14, 31] all have the specification of the control problem provided as a set of LTL formulas rather than automata. Hence, they required a translation of the safety part of these specifications to LTS. To complete our validation, we also included a series of case studies that were provided in their original formulation as a composition of automata: the t-strong fairness case study from [16], a game of tic-tac-toe and and the CTvPlatform Controller [51]. For these we were forced to manually seed unrealizability causes.

In the following we report quantitatively on all case studies to show the degree to which the initial control problem is reduced and the computational cost involved. We also report qualitatively on the case studies taken from [2, 14, 31] as they provide an unbiased basis against to which it is possible to evaluate the insight into non realizability causes that our technique provides.

All case studies were run using an extension of the MTSA tool [18]. MTSA natively supports specification of LTS [26] and properties using a textual, process algebraic notation and linear temporal logic. The tool supports synthesis of controllers for SGR(1) control problems where the safety part of the problem can specified as an composition of LTS, and the liveness goals and assumptions are provided using temporal logic. The extension builds a minimized unrealizable version of the environment which can then be inspected through various visualization methods including animations and visualization of the state space.

The quantitative results of our experiments are shown in table 1 and were run on an `Intel® Core™ i7-4790` CPU with 8 processors of 3.60GHz frequency 16 Gb of RAM memory and `ubuntu 14.04` as the operating system. Total time taken by the algorithm is measured in milliseconds and labeled as *Time*, $|S_e|$ and $|S_{e\kappa}|$ stand for the size of the original and minimized plants in terms of states, *State Reduction* is the reduction percentage of these. $|\Delta_e|$ and $|\Delta_{e\kappa}|$ stand for the size of the original and minimized plants in terms of edges, *Transition Reduction* is the reduction percentage of the later two. *steps* reports how many iterations the algorithm needed before finding the first minimised sub-LTS. *Edges Explored* reports the percentage of edges the algorithm needed to explore before finding the first conflict.

From the results it is possible to argue that the minimization technique can be applied to control problems of similar size to those reported in the literature. In addition, it can be seen that the technique can achieve a significant reduction in terms of states and transitions.

6.1 Mine Pump Controller (MPC)

The Mine Pump Controller (MPC) originally presented in [32] describes a controller that should drain water from a mine whenever its level surpassed a certain threshold, with the restriction that it should stop the pump if the sensors indicated that a high level of methane was present due to the risk of ignition.

We used the variant specification of the problem provided in [14] where they highlight that two safety goals ($\Box(methane.hi \implies \bigcirc(\neg running))$ and $\Box(water.hi \implies \bigcirc(running))$) are logically consistent but there is a *boundary condition* that if true, makes the conjunction of the two safety goals impossible. The GR(1) liveness requirement states that the water level to be kept within reasonable limits infinitely often ($\Box\Diamond\neg water.Hi$).

We applied our minimization on the translated specification from [14] to get the plant depicted in Figure 9. We can see here that the minimization properly slices the plant to expose what is characterized as the boundary condition ($\Diamond hw \wedge m$) and in our case is preventing the system from achieving its goal.

This slice should lead the engineer towards adding the missing assumption $\Box\Diamond(\neg methane.hi \wedge \neg water.hi)$. If the original problem and the implementation proposed in the appendix from [32]

Case	Time (ms)	$\|S_e\|$	$\|S_{e\kappa}\|$	State Reduction	$\|\Delta_e\|$	$\|\Delta_{e\kappa}\|$	Transition Reduction	steps	Edge Explored
t strong fairness	10.65	7	4	57.14%	9	4	44.44%	3	33.33%
mine pump	8.7	14	5	35.71%	20	5	25.00%	6	30.00%
lift controller	121	95	43	45.26%	130	50	38.46%	35	26.92%
Tv controller	11031	218	8	3.67%	1416	21	1.48%	43	3.04%
tictactoe	75279	4696	236	5.03%	18647	472	2.53%	529	2.84%
gen buf (miss. assumpt.)	6688142	10964	2235	20.38%	14894	2963	19.89%	1093	7.34%
gen buf (added goal)	7882862	10964	850	7.75%	14894	1139	7.64%	646	4.33%
gen buf (added restr.)	73	39	26	66.66%	47	28	59.57%	10	21.27%

Table 1: Quantitative results for minimized plants

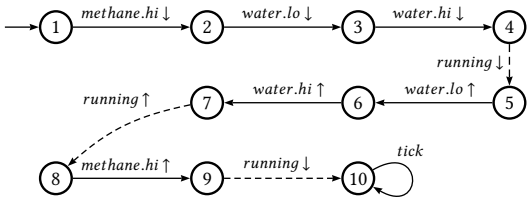

Figure 9: Mine Pump Controller (MPC) minimization.

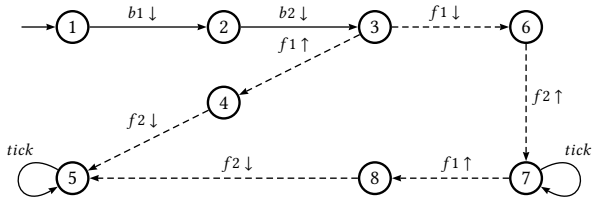

Figure 10: Modified Lift Controller minimization.

are expected to be deployed, this assumption should be added for the problem to be realizable.

Now we explain the behavior depicted in Figure 9. After the initial condition is met (between states 1 and 5) the environment can then raise the water level (transition from state 6 to 7) forcing the system to start the pump (transition from state 7 to 8) but then raise the methane level (transition from 8 to 9) giving to option to the system but to turn off the pump (transition 9 to 10). After this the environment can keep the signals constant denying the system the opportunity to satisfy its goal.

6.2 Modified Lift Controller

Another example that has been evaluated is that of the Modified Lift Controller from [2] first presented in its realizable version in [7]. In the original version a lift controller should serve several floors. It has a button sensor for each floor controlled by the environment. There is an additional restriction that allows the lift to move up only if there is a pending request in one of the floors ($\square\diamond(b_j \implies f_j)$), and another that forces the system to visit the first floor infinitely often if no button was pressed.
In the modified version from [2] a new set of system liveness requirements is added, where the system should visit each floor infinitely often ($\square\diamond f_j$).

The problem here is that this requirement collides with the restriction that the lift will not move up if there is no pending request.
We run the minimization over a simplified two floors specifications and produced the plant depicted in Figure 10. It shows that after initial variable setting on behalf of the environment (states 1 to 3) the system is free to choose opposite values for the initial floor (either start at floor 1, states 3 to 5, or start at the second floor, states 3 to 8), if it chooses to start at the first floor the environment leaves the $b1, b2$ signals down, falsifying the $\square\diamond f_j$ requirement, since it cannot go up to visit the second floor unless $b2$ is up. On the other hand if the system chooses the second floor as the initial configuration the environment keeps waiting until the lift descends to win by letting the signals down. The system is forced to descend eventually to satisfy the restriction that it should visit the first floor when $b1$ and $b2$ are down. In this example we have to do some more interpretation after the plant has been minimized, but we think this is a reasonable expectation. The minimization removed 84 states out of the 95 in the original specification, so even if the engineer has to correlate liveness assumptions with the minimized plant the technique proves itself helpful by hiding irrelevant behavior.
Going back to [2] we see that our slice is consistent with the diagnosis presented by the authors where they say that *(…)A counter-strategy for the environment is to always keep all b_i's low(…).* Indeed, this counter strategy is captured by the minimised plant depicted in Figures 10.

6.3 Generalized Buffer (GenBuf)

Non realizable specifications of the Generalized Buffer Case Study [5] were studied in [30]. The original specification was introduced by IBM with their Rulebase verification tool [4]. It describes a family of buffers that transmit data from n senders $Sender_0, Sender_1, \ldots, Sender_n$ to two receivers $Receiver_0$ and $Receiver_1$. Data is received from senders in a random order and should be delivered to the receivers in round-robin order. It contains a handshake protocol to communicate with each sender/receiver device by exchanging $StoB\_REQ(i) \leftrightarrow BtoS\_ACK(i)$ messages on the senders side and $BtoR\_REQ(j) \leftrightarrow RtoB\_ACK(j)$ on the receivers. A controller should arbitrate between the senders and the receivers to accomplish proper

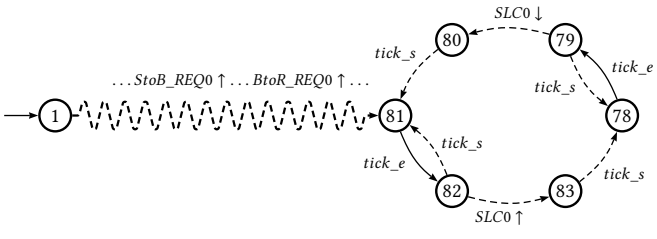

Figure 11: GenBuf (missing assumption) Terminal Set.

communication through a four slot FIFO queue. Several restrictions and assumptions are declared in order to ensure consistent behavior (e.g. *G2: No request should be immediately acknowledged since it is valid at least one step after the assertion of request*).

We studied the three non-realizable specifications of GenBuf discussed in [30]. The goal is captured by the LTL formula $\square\diamond($ $StoB\_REQ0 \leftrightarrow BtoS\_ACK0)$). In the first the authors consider removing the assumption that expresses that always eventually if a request to read data has been sent from the controller to $Receiver_0$ ($BtoR\_REQ0$) then an acknowledge will be issued ($RtoB\_ACK0$). The minimized environment decreased 80% (from 10964 to 2235 states) and, amongst other things, excluded all events related with $Receiver_1$, since they are irrelevant to the environment's strategy. In the minimized plant we identified one terminal set depicted in Figure 11 in which one guarantee is being neglected infinitely often: In the trace leading to the terminal set the buffer sends $BtoR\_REQ0$ message to $Receiver_0$ but never gets the expected acknowledge ($RtoB\_ACK0$), thus not being able to produce an acknowledge for the sender. This diagnosis should point the engineer in the direction to enforce the specification by restricting the environment to always eventually produce an acknowledge message for each requirement sent to the receivers.
The second variation studied in [30] incorporated the goal $\square\diamond$ $(\neg EMPTY \wedge \neg DEQ)$. The minimized environment decreased 92% (from 10964 to 850 states) with only one strongly connected component at which point the buffer remains empty and the environment keeps exchanging the value of the signal *SLC* and never requests the bus ($StoB\_REQ(i)$), thus preventing the controller to achieve the new goal. This tells us that the problem is the lack of an assumption that states that the environment must always eventually achieve *ENQ* thus lowering the *EMTPY* signal.
The final variation analyzed in [30] restricts specification by adding a new safety requirement $\square\neg ENQ$. This already reduces the size of the problem significantly (39 states) leaving little room for minimization (resulting in 26 states). Indeed, in both the original and minimized environment LTS it is straightforward to visualize the deadlock state which is reached after the first $BtoS\_ACK(i)$ message. This acknowledgement is required to be followed by raising the *ENQ* signal which would violate the new requirement.

7 DISCUSSION AND RELATED WORK

The problems an engineer can face when writing the specification of an open system can be characterized in other ways, for instance, when dealing with a design by contract type of specification, a set of assumptions is defined that has to be met if we expect to accomplish our goals, and if they are not, the specification is vacuously satisfied. This has been presented in [33] for CTL* specifications, in [29] for the GR(1)[42] subset, further explored in [38] for the declarative version and in [15] for the generative one. The problem of completeness (where the idea is to find sets of formulas irrelevant to the realization of the goals) has been developed in [10] and [11], amongst others. There are two different although related problems when it comes to the non satisfaction of the expected properties, namely realizability and satisfiability. These are directly related to the difference between open systems and closed systems. Since we are working with open system specifications we will reason about in terms of realizability. Here we try to obtain the satisfaction of a certain set of goals against a potentially antagonistic environment (presented as the Skolem paradigm in [43]), as opposed to satisfiability, where the question to be answered is if it exists a cooperation between the environment and the system that satisfies the goals.

In what we consider to be one the works more closely related to our technique [30] the authors present a debugging framework for non realizable specifications that provides several diagnosis mechanisms, the minimization of the declarative requirements that preserve the non realizability problem as a subset of the initially provided LTL formulas, the elimination of irrelevant output variables and a counter strategy (counter trace) that can be explored interactively. In terms of minimization our technique is semantic as opposed to theirs being syntactic, this allows for a canonical minimization not attached to the granularity of the formulas provided as individual requirements (as already pointed out by [47]). Both minimization techniques are general since they use a realizability oracle to perform the exploration of the search space. Our minimization is also complimentary to counter strategy as feedback for two reasons, it is not bounded to the particular implementation of the counter strategy (potentially providing the user all non realizability causes by exploring the induced semi lattice) and it is not affected by the counter strategy memory that will cause a mismatch with the original plant's states due to unfolding. Our setting (and our search space characterization) allows us to provide conflicts in an enumerative fashion by exhaustively exploring the induced semi-lattice. This differs considerably to the single conflict diagnosis provided by executing a counter-strategy bounded by its implementation to a particular conflict. One could be tempted to translate an operational specification into a set of LTL formulas and then apply the minimization presented in [30]. Such a syntactic minimization will in fact increase the behavior of the plant since removal of constituent subformulas relaxes the transition relation allowing for more interaction that originally specified.
[47] presents a simplification of declarative specifications (expressed through LTL formulas) by computing the unrealizability cores (UCs) a canonical representation of LTL formulas for unsatisfiability and unrealizability computation of a potentially finer grain than [30]. The authors take a syntactical approach when simplifying a formula by over or under approximating of its components, clearly different from our semantical approach.
The idea of [28] is to look for the closest satisfiable specification with respect to the initial specification, which is known as the minimal revision problem (MRP). The search space induced by the relation of closest specification is quite similar, in structural terms, to

ours. Besides looking for the closest satisfiable approximation (as opposed to a minimal non realizability preserving representation in our case) the relaxation in this work is performed over the labels consisting each in a conjunction of literals and their relaxation being weaker sub formulas of these. This concept was revisited later in [27] for weighted transition systems. It is clear that MRP is not a non realizability diagnosis but instead tackles the problem of finding one the closest satisfiable representations.

In [14] the authors compute `boundary conditions` over LTL for specifications that can be initially satisfied but will eventually diverge. A boundary condition is such that, while consistent, when added to a conjunction with the set of domain assumptions and system guarantees can not be satisfied. In [20] strongly unsatisfiable subsets of reactive specifications are defined and computed in this work, strong satisfiability is studied for its simplicity and since it is a necessary precondition for non realizability. These two approaches are different to ours since they do not, nor intend to, treat the non realizability problem.

[2], [36] and [8] deal with the problem of automatically producing (mining) assumptions for non realizable specifications. This is related to out technique since it works with non realizable specification but has a very different intention which is repair-based rather than diagnosis-based. [2] initially tries to correct an unrealizable specification by adding assumptions. They use the counter strategy to build new assumptions following predefined patterns. User interaction is needed to identify underspecified variables. In [36], the authors propose mining assumptions out of an unrealizable GR(1) specification through the use of a counter strategy from [31]. Again, a template based approach is used. In [8] an assumption computing technique is presented based in Craig's interpolants is presented. It obtains refinements by negating plays of the counter-strategy.

In [9] an interactive play-out for scenario-based specifications is presented. If the specifications is unrealizable an interactive game is presented to the user in order to expose the cause of non realizability. It is built following [31] counter-strategy and its restricted to scenario-based specifications domain. In [34] the authors present justice violation transition systems for symbolic specifications as a way to abstract and simplify the winning strategy for the environment. States in the JVTSs are labeled with invariants, since they somehow collapse states of the counter strategy in order to expose relevant environmental decisions. This differs from our technique not only in its type of input which consists of a set of LTL formulas, but because it is about abstracting the strategy instead of minimizing the behavior of the plant. In [40] two-way traceability for AspectLTL specifications is defined in order to link each allowed or forbidden transition in the generated program with the aspect justifying its presence or elimination, for unrealizable specifications an interactive game is provided to explain the cause of non realizability.

Several works have used mobile robotics examples as motivation. Non realizability diagnosis has been applied to this domain in [46] and [45].

In [52] an approach to compute differences between LTSs is introduced in the context of evolving behaviors. This is slightly related with our work in the sense that it copes with the potential divergence between the current model and the initial user intent.

8 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a technique that semantically minimizes behavior while preserving non realizability in order to provide feedback to an engineer. The problem was introduced in the context of operational specifications where the liveness guarantees and assumptions are provided as LTL formulas and safety behavior is expressed as a composition of LTS automata. The characterization of the search space as a semi lattice of sub LTSs derived from the original plant allowed us to introduce an eager algorithm that finds a minimal representative of non realizability preserving behavior. We believe that our work complements pre-existing approaches bases on syntactic minimisation Future work can specialize our approach by defining specific exploration heuristics of the search space for particular sublogics, such as GR(1).

In addition, we believe that our technique can help and provide complimentary value to the specification-repair approaches, and could be used alongside assumption mining tools.

REFERENCES

[1] 2012. *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6215071

[2] Rajeev Alur, Salar Moarref, and Ufuk Topcu. 2013. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. IEEE, 26–33. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6679387

[3] Sharon Barner, Ian G. Harris, Daniel Kroening, and Orna Raz (Eds.). 2011. *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*. Lecture Notes in Computer Science, Vol. 6504. Springer. DOI: http://dx.doi.org/10.1007/978-3-642-19583-9

[4] Ilan Beer, Shoham Ben-David, Cindy Eisner, and Avner Landver. 1996. RuleBase: An industry-oriented formal verification tool. In *Proceedings of the 33rd annual Design Automation Conference*. ACM, 655–660.

[5] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. 2007. Specify, Compile, Run: Hardware from PSL. *Electr. Notes Theor. Comput. Sci.* 190, 4 (2007), 3–16. DOI: http://dx.doi.org/10.1016/j.entcs.2007.09.004

[6] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3 (2012), 911–938. DOI: http://dx.doi.org/10.1016/j.jcss.2011.08.007

[7] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3 (2012), 911–938. DOI: http://dx.doi.org/10.1016/j.jcss.2011.08.007

[8] Davide G. Cavezza and Dalal Alrajeh. 2017. Interpolation-Based GR(1) Assumptions Refinement. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I (Lecture Notes in Computer Science)*, Axel Legay and Tiziana Margaria (Eds.), Vol. 10205. 281–297. DOI: http://dx.doi.org/10.1007/978-3-662-54577-5_16

[9] Pavol Cerný, Sivakanth Gopi, Thomas A. Henzinger, Arjun Radhakrishna, and Nishant Totla. 2012. Synthesis from incompatible specifications. In *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, October 7-12, 2012*, Ahmed Jerraya, Luca P. Carloni, Florence Maraninchi, and John Regehr (Eds.). ACM, 53–62. DOI: http://dx.doi.org/10.1145/2380356.2380371

[10] Hana Chockler, Orna Kupferman, Robert P Kurshan, and Moshe Y Vardi. 2001. A practical approach to coverage in model checking. In *International Conference on Computer Aided Verification*. Springer, 66–78.

[11] Hana Chockler, Orna Kupferman, and Moshe Y Vardi. 2001. Coverage metrics for temporal logic model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 528–542.

[12] M. Daniele, P. Traverso, and M.Y. Vardi. 2000. Strong Cyclic Planning Revisited. *Recent Advances in Ai Planning: 5th European Conference on Planning, Ecp'99, Durham, Uk, September 8-10, 1999: Proceedings* (2000).

[13] Luca de Alfaro and Thomas A. Henzinger. 2001. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, A. Min Tjoa and Volker Gruhn (Eds.). ACM, 109–120. DOI: http://dx.doi.org/10.1145/503209.503226

[14] Renzo Degiovanni, Nicolás Ricci, Dalal Alrajeh, Pablo F. Castro, and Nazareno Aguirre. 2016. Goal-conflict detection based on temporal satisfiability checking. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, David Lo, Sven Apel, and Sarfraz Khurshid (Eds.). ACM, 507–518. DOI: http://dx.doi.org/10.1145/2970276.2970349

[15] Nicolás D'Ippolito. 2013. *Synthesis of event-based controllers for software engineering*. Ph.D. Dissertation. Imperial College London, UK. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.576049

[16] Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. 2011. Synthesis of live behaviour models for fallible domains. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic (Eds.). ACM, 211–220. DOI: http://dx.doi.org/10.1145/1985793.1985823

[17] Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. 2013. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.* 22, 1 (2013), 9:1–9:36. DOI: http://dx.doi.org/10.1145/2430536.2430543

[18] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. 2008. MTSA: The Modal Transition System Analyser. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*. IEEE Computer Society, 475–476. DOI: http://dx.doi.org/10.1109/ASE.2008.78

[19] Dimitra Giannakopoulou and Jeff Magee. 2003. Fluent model checking for event-based systems. In *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 held jointly with 9th European Software Engineering Conference, ESEC/FSE 2003, Helsinki, Finland, September 1-5, 2003*, Jukka Paakki and Paola Inverardi (Eds.). ACM, 257–266. DOI: http://dx.doi.org/10.1145/940071.940106

[20] Shigeki Hagihara, Naoki Egawa, Masaya Shimakawa, and Naoki Yonezaki. 2014. Minimal strongly unsatisfiable subsets of reactive system specifications. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, Ivica Crnkovic, Marsha Chechik, and Paul Grünbacher (Eds.). ACM, 629–634. DOI: http://dx.doi.org/10.1145/2642937.2642968

[21] David Harel. 1987. Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Program.* 8, 3 (1987), 231–274. DOI: http://dx.doi.org/10.1016/0167-6423(87)90035-9

[22] C. A. R. Hoare. 1978. Communicating Sequential Processes. *Commun. ACM* 21, 8 (1978), 666–677. DOI: http://dx.doi.org/10.1145/359576.359585

[23] C. A. R. Hoare. 1983. Communicating Sequential Processes. *Commun. ACM* 26, 1 (Jan. 1983), 100–106. DOI: http://dx.doi.org/10.1145/357980.358021

[24] Michael Jackson. 1995. The world and the machine. In *Proceedings of the 17th international conference on Software engineering (ICSE '95)*. ACM, New York, NY, USA, 283–292. DOI: http://dx.doi.org/10.1145/225014.225041

[25] Robert M. Keller. 1976. Formal verification of parallel programs. *Commun. ACM* 19 (July 1976), 371–384. Issue 7. DOI: http://dx.doi.org/10.1145/360248.360251

[26] Robert M. Keller. 1976. Formal Verification of Parallel Programs. *Commun. ACM* 19, 7 (1976), 371–384. DOI: http://dx.doi.org/10.1145/360248.360251

[27] Kangjin Kim and Georgios E. Fainekos. 2013. Minimal specification revision for weighted transition systems. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. IEEE, 4068–4074. DOI: http://dx.doi.org/10.1109/ICRA.2013.6631151

[28] Kangjin Kim, Georgios E. Fainekos, and Sriram Sankaranarayanan. 2012. On the revision problem of specification automata, See [1], 5171–5176. DOI: http://dx.doi.org/10.1109/ICRA.2012.6224826

[29] Uri Klein and Amir Pnueli. 2010. Revisiting Synthesis of GR(1) Specifications, See [3], 161–181. DOI: http://dx.doi.org/10.1007/978-3-642-19583-9_16

[30] Robert Könighofer, Georg Hofferek, and Roderick Bloem. 2009. Debugging formal specifications using simple counterstrategies. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. IEEE, 152–159. DOI: http://dx.doi.org/10.1109/FMCAD.2009.5351127

[31] Robert Könighofer, Georg Hofferek, and Roderick Bloem. 2010. Debugging Unrealizable Specifications with Model-Based Diagnosis, See [3], 29–45. DOI: http://dx.doi.org/10.1007/978-3-642-19583-9_8

[32] J. Kramer. 1983. CONIC: an integrated approach to distributed computer control systems. *IEE Proceedings E (Computers and Digital Techniques)* 130 (January 1983), 1–10(9). Issue 1. http://digital-library.theiet.org/content/journals/10.1049/ip-e.1983.0001

[33] Orna Kupferman and Moshe Y Vardi. 2003. Vacuity detection in temporal model checking. *International Journal on Software Tools for Technology Transfer (STTT)* 4, 2 (2003), 224–233.

[34] Aviv Kuvent, Shahar Maoz, and Jan Oliver Ringert. 2017. A symbolic justice violations transition system for unrealizable GR(1) specifications. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 362–372. DOI: http://dx.doi.org/10.1145/3106237.3106240

[35] Emmanuel Letier and Axel van Lamsweerde. 2002. Agent-based tactics for goal-oriented requirements elaboration. In *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*. ACM, New York, NY, USA, 83–93. DOI: http://dx.doi.org/10.1145/581339.581353

[36] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. 2011. Mining assumptions for synthesis. In *9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011, Cambridge, UK, 11-13 July, 2011*, Satnam Singh, Barbara Jobstmann, Michael Kishinevsky, and Jens Brandt (Eds.). IEEE, 43–50. DOI: http://dx.doi.org/10.1109/MEMCOD.2011.5970509

[37] J. Magee and J. Kramer. 2006. *Concurrency: state models & Java programs*. Wiley New York.

[38] Shahar Maoz and Jan Oliver Ringert. 2016. On well-separation of GR(1) specifications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 362–372. DOI: http://dx.doi.org/10.1145/2950290.2950300

[39] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2014. Synthesis of Component and Connector Models from Crosscutting Structural Views. In *Software Engineering 2014, Fachtagung des GI-Fachbereichs Softwaretechnik, 25. Februar - 28. Februar 2014, Kiel, Deutschland (LNI)*, Wilhelm Hasselbring and Nils Christian Ehmke (Eds.), Vol. 227. GI, 63–64. http://eprints.uni-kiel.de/23752/

[40] Shahar Maoz and Yaniv Sa'ar. 2013. Two-Way Traceability and Conflict Debugging for AspectLTL Programs. *Trans. Aspect-Oriented Software Development* 10 (2013), 39–72. DOI: http://dx.doi.org/10.1007/978-3-642-36964-3_2

[41] R. Milner. 1982. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[42] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2006. Synthesis of Reactive(1) Designs. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings (Lecture Notes in Computer Science)*, E. Allen Emerson and Kedar S. Namjoshi (Eds.), Vol. 3855. Springer, 364–380. DOI: http://dx.doi.org/10.1007/11609773_24

[43] Amir Pnueli and Roni Rosner. 1989. On the Synthesis of a Reactive Module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989.* ACM Press, 179–190. DOI: http://dx.doi.org/10.1145/75277.75293

[44] P.J.G Ramadge and W.M Wonham. 1989. The control of discrete event systems. *Proc. IEEE* 77, 1 (1989), 81–98. DOI: http://dx.doi.org/10.1109/5.21072

[45] Vasumathi Raman and Hadas Kress-Gazit. 2011. Analyzing Unsynthesizable Specifications for High-Level Robot Behavior Using LTLMoP. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 663–668. DOI: http://dx.doi.org/10.1007/978-3-642-22110-1_54

[46] Vasumathi Raman and Hadas Kress-Gazit. 2012. Automated feedback for unachievable high-level robot behaviors, See [1], 5156–5162. DOI: http://dx.doi.org/10.1109/ICRA.2012.6224807

[47] Viktor Schuppan. 2012. Towards a notion of unsatisfiable and unrealizable cores for LTL. *Sci. Comput. Program.* 77, 7-8 (2012), 908–939. DOI: http://dx.doi.org/10.1016/j.scico.2010.11.004

[48] Emina Torlak, Felix Sheng-Ho Chang, and Daniel Jackson. 2008. Finding Minimal Unsatisfiable Cores of Declarative Specifications. In *Proceedings of the 15th International Symposium on Formal Methods (FM '08)*. Springer-Verlag, Berlin, Heidelberg, 326–341.

[49] Sebastián Uchitel, Jeff Kramer, and Jeff Magee. 2003. Synthesis of Behavioral Models from Scenarios. *IEEE Trans. Software Eng.* 29, 2 (2003), 99–115. DOI: http://dx.doi.org/10.1109/TSE.2003.1178048

[50] Axel van Lamsweerde and Emmanuel Letier. 2000. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering* 26 (October 2000), 978–1005. Issue 10. DOI: http://dx.doi.org/10.1109/32.879820

[51] Rob C. van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. 2000. The Koala Component Model for Consumer Electronics Software. *IEEE Computer* 33, 3 (2000), 78–85. DOI: http://dx.doi.org/10.1109/2.825699

[52] Zhenchang Xing, Jun Sun, Yang Liu, and Jin Song Dong. 2011. Differencing Labeled Transition Systems. In *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings (Lecture Notes in Computer Science)*, Shengchao Qin and Zongyan Qiu (Eds.), Vol. 6991. Springer, 537–552. DOI: http://dx.doi.org/10.1007/978-3-642-24559-6_36