# Behavior Minimization Feedback for Unrealizable Specifications

Anonymous Author(s)

## 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. The technique assumes an automata-based description of the environment and returns a model that preserves the cause for unrealizability in the sense that the environment's strategy for beating the system in the minimized model is also a winning strategy in the original model.

**ACM Reference Format:**
Anonymous Author(s). 2018. Behavior Minimization Feedback for Unrealizable Specifications . In *Proceedings of The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, Florida, United States, 4–9 November, 2018 (ESEC/FSE 2018),* 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Requirements are naturally split between goals the system-to-be is expected to achieve and assumptions that the system-to-be can rely on to fulfill its goals [21, 31]. 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 capabilities in order to guarantee its goals as long as the environment fulfils the assumptions?

At early stages specifications are usually unrealizable [31]. There can be multiple causes for unrealizability including lack of monitorability and controllability, and over-idealization of 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 [42].

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 manual and laborious. 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 [39], FOND planning [11] and controller synthesis [5, 34]. In these settings, given a specification, 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 in [2, 26, 40], these assume that the specification is given in an expressive subset of Linear Temporal Logic (LTL) formulae, called GR(1), and return a minimal subset of these that is still unrealizable. In other words, and in the spirit of unsatisfiable cores [41], the technique produces a minimal subset of the original formulae that is still unrealizable. The intention is to allow engineers to focus on a portion of the original specification to identify the causes for unrealizability.

In this paper we propose an alternative and potentially complementary technique for providing feedback on unrealizability. The technique *differs* from existing ones in two ways. First and foremost, the approach minimizes allowable behavior while preserving unrealizability. The result is a model that exhibits a minimal representation of the original behavior while preserving the unrealizability causes present in the initial specification. In contrast, in existing approaches the reduced number of formulae results in a model that allows more behavior than originally specified. The second difference is that the approach presented herein assumes an automata-based description of the environment rather than an LTL specification. Automata-based specifications of the environment (e.g., labeled transition systems [22], statecharts [18], and process algebra [20, 35]) are common approaches to environment description in software engineering literature. There are a variety of software engineering tasks that have been approached using synthesis and automata-based descriptions (e.g, [15, 30, 36]).

The technique we present minimizes behavior while preserving unrealizability. *What does this mean?* A specification determines 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, it cannot achieve its goals. We refer to a minimization that preserves unrealizability as a process that produces a new model that has less


*ESEC/FSE 2018, 4–9 November, 2018, Lake Buena Vista, Florida, United States*

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

behavior and for which the winning strategy of the environment also works in the original specification. This allows engineers to focus on the behavior of the original specification that causes un realizability.

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 the notion of realizability, and then go on to define minimization and non-realizability preservation. In Section 5 we present the minimization algorithm and then discuss our way of validating the technique. We finally proceed with a discussion on related work and conclusions.

## 2 MOTIVATION

We motivate and informally introduce relevant concepts by discussing a small example while also illustrating what we consider to be the complementary nature of both the presented approach and that of [27]. Suppose that the engineer is working on a specification consisting of two environmental boolean variables $x_1$ and $x_2$ and one system variable $y$. Variable $x_2$ (initially false) is required to swap its value at each step of the execution, while $x_1$ remains free. The value of $y$ is set at the start and kept fixed throughout the execution. It is assumed that $x_1$ will hold true infinitely often while $x_1 \wedge x_2 \wedge y$ is also required to be satisfied infinitely often. This is the set of LTL formulae of the specification, with $\theta$ being the initial condition, $\rho_e$ the environmental safety restriction, $\varphi_e$ the liveness assumption and $\varphi_s$ the liveness guarantee:

$$\begin{aligned} \theta &= \neg x_1 \wedge \neg x_2 \\ \rho_e &= \Box(x_2 \leftrightarrow \bigcirc(\neg x_2)) \\ \rho_s &= \Box(y \leftrightarrow \bigcirc(y)) \\ \varphi_e &= \Box\Diamond x_1 \\ \varphi_s &= \Box\Diamond(x_1 \wedge x_2 \wedge y) \end{aligned}$$

The automaton induced from this set of formulae is depicted in Fig. 1. At this point, the specification is not realizable since the environment can set $x_1$ to true only when $x_2$ is down, thus honoring the environmental assumption while avoiding the system to reach the winning valuation $x_1 \wedge x_2 \wedge y$. The minimization presented in [27] removes $\rho_e$ since it is unnecessary for the environment to swap $x_2$ regularly in order to win the (non) realizability game. This approach can be characterized as a syntactic minimization, whose purpose is to help the engineer when viewing the specification, and in particular the safety part describing the behavior of the plant, as a set of LTL formulae. This approach, nonetheless, yields the automaton depicted in Fig. 2 which is, in fact, more complex in terms of its explicit behavior. Our approach is designed to simplify the semantic view of the specification, producing the automaton depicted in Fig. 3.

The syntactic minimization should help the engineer to focus on a smaller set of formulae by getting rid of the irrelevant element $\rho_e$ while diagnosing the non realizability cause, in this example being that assumption $\varphi_e$ is too weak. Our approach simplifies the specification semantically instead, reducing the relevant behavior and showing that it is possible for the environment to avoid state 2 by never allowing $x_1$ and $x_2$ to happen simultaneously, while at the same time not restricting the system below its original capabilities. Notice that if we were allowed to add the assumption

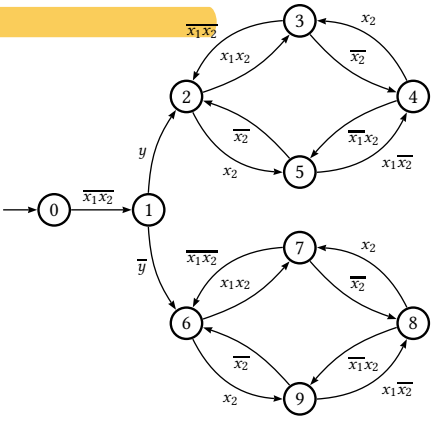


**Figure 1: Original Plant**

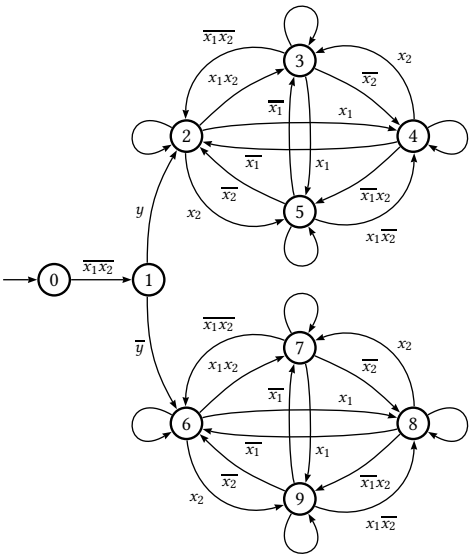


**Figure 2: Minimization according to Könighoefer**

that both environmental variables occur simultaneously infinitely often ($\rho_s' = \Box\Diamond x_1 \wedge x_2$) the specification would be realizable and choosing to raise $y$ at the beginning would be sufficient in order to guarantee our goal, as shown in Fig. 4. After fixing $y$ to true the environment can move freely as long as it visits state 2 infinitely often. If the engineer has this behavior in mind as her design intent, it should be easy to see that in the semantic minimization the desired state is unreachable, hopefully guiding her to the strengthening of the assumption. In Fig. 5 we show the synchronous composition of the aforementioned strategy with our minimization. Note that after letting the system raise $y$ state 1 is visited only once and then the execution keeps moving between states 2 and 3 in order to keep the value of $x_2$ oscillating.

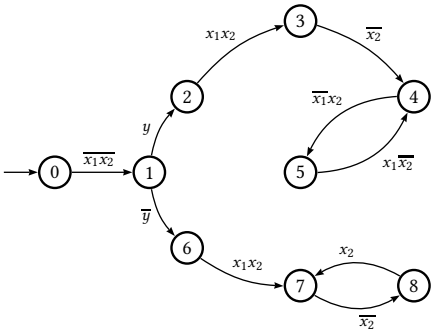


**Figure 3: Behavior minimization**

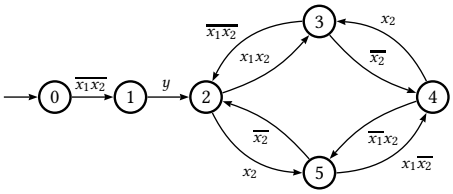


**Figure 4: Realizable version strategy**

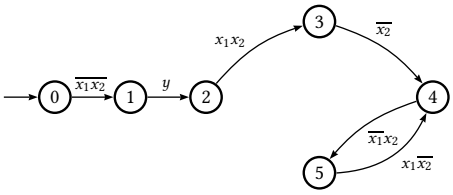


**Figure 5: Composition of the strategy with the minimization**

## 3 PRELIMINARIES

We model safety assumptions on the environment and goals for the system as composition of Concurrent Labeled Transition Systems (CLTS).

*Definition 3.1. (Concurrent Labeled Transition Systems)* A *Concurrent Labeled Transition System* (CLTS) is $E = (S, \Sigma, \Delta, s_0)$, where $S$ is a finite set of states, $\Sigma \subseteq Act$ is its *communicating alphabet*, $\Delta \subseteq (S \times \mathcal{P}(\Sigma) \times S)$ 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 CLTS is deterministic if $(s, a, s')$ and $(s, a, s'')$ are in $\Delta$ 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 $\pi$ is a trace (induced by $\varepsilon$) of $E$ if its the result of removing every $s_i$ from an execution $\varepsilon$ of $E$. We denote the set of infinite traces of $E$ by $Tr(E)$.

*Definition 3.2.* (*Synchronous Composition*) Let $M = (S_M, \Sigma_M, \Delta_M, s_0^M)$, $N = (S_N, \Sigma_N, \Delta_N, s_0^N)$, with $\Delta_M : S_M \times \mathcal{P}(\Sigma_M) \times S_M$, be two CLTS instances, then CLTS synchronous parallel composition is defined as $M \|_s N = (S_M \times S_N, \Sigma_M \cup \Sigma_N, \Delta, (s_0^M, s_0^N))$ where $\Delta$ is the smallest relation s.t:

$$\frac{(s, l_M, s') \in \Delta_M, (t, l_N, t') \in \Delta_N}{((s,t), l_M \cup l_N, (s', t')) \in \Delta} \; l_M \cap \Sigma_N = l_N \cap \Sigma_M \quad (1)$$

The distinction between what an CLTS can control and what it can monitor is enforced through the notion of *legal environment* taken from [15], and inspired in that of Interface Automata [12]. 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. (Legality w.r.t. E and C)* Given CLTS $C = \langle S_c, \Sigma, \Delta_c, s_{c_0}\rangle$ and $E = \langle S_e, \Sigma, \Delta_e, s_{e_0}\rangle$, where $\Sigma$ is partitioned into actions controlled and monitored (non controllable) by $C$ ($\Sigma = \boldsymbol{C} \cup \boldsymbol{\mathcal{U}}$), we say that $C$ is a legal CLTS for $E$ if for all $(s_e, s_c) \in E \|_* C$ it holds that: $\Delta_E(s_E) \cap \mathcal{P}(\boldsymbol{C}) \supseteq \Delta_C(s_C) \cap \mathcal{P}(\boldsymbol{C})$ and also that $\Delta_E(s_E) \cap \mathcal{P}(\boldsymbol{\mathcal{U}}) \subseteq \Delta_C(s_C) \cap \mathcal{P}(\boldsymbol{\mathcal{U}})$.

We use linear temporal logics of fluents (FLTL) over CLTS models. A *fluent fl* is defined by a pair of sets and a Boolean value: $\mathrm{fl} = \langle I_{\mathrm{fl}}, T_{\mathrm{fl}}, Init_{\mathrm{fl}}\rangle$, where $I_{\mathrm{fl}} \subseteq \mathcal{P}(Act)$ is the set of initiating actions, $T_{\mathrm{fl}}\mathcal{P}(Act)$ is the set of terminating actions and $I_{\mathrm{fl}} \cap T_{\mathrm{fl}} = \emptyset$. A fluent may be initially *true* or *false* as indicated by $Init_{\mathrm{fl}}$. Every actions $\ell \in Act$ induces a fluent, namely $\hat{\ell} = \langle\{\ell\}, \{Act \setminus \{\ell\}\}, false\rangle$. 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 $\mathcal{P}(Act)$. 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 $\wedge$, **F** (eventually), and **G** (always) as syntactic sugar. Let $\Pi$ 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:

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

Suppose that $fl \in \mathcal{F}$, in order to avoid having a label $l_i$ in our trace $\pi$ where both the initiation and termination of *fl* are triggered, we will require that our model $E$ satisfies a notion of consistency with respect to a set of fluents, we say that a model $E$ is consistent with respect to a set of fluents $\mathcal{F}$, noted $\cong (E, \mathcal{F})$ if the following safety condition holds:

$$\forall fl \in \mathcal{F}, (s, l, s') \in \Delta_E : \square(\neg(\exists i \in I_{fl} : i \subseteq l) \vee \neg(\exists t \in T_{fl} : t \subseteq l))$$

Given an infinite trace $\pi$, the satisfaction of a formula $\varphi$ at position $i$, denoted $\pi, i \models \varphi$, is defined as follows:

$$\begin{array}{lcl} \pi, i \models_d \neg\varphi & \triangleq & \pi, i \not\models_d \varphi \\ \pi, i \models_d \varphi \vee \psi & \triangleq & (\pi, i \models_d \varphi) \vee (\pi, i \models_d \psi) \\ \pi, i \models_d \mathbf{X}\varphi & \triangleq & \pi, i+1 \models_d \varphi \\ \pi, i \models_d \varphi \mathbf{U} \psi & \triangleq & \exists j \mathbf{G} e q i . \pi, j \models_d \psi \wedge \forall i \leq k \leq k . \pi, k \models_d \varphi \end{array}$$

**Figure 6: Semantics for the satisfaction operator**

We say that $\varphi$ holds in $\pi$, denoted $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula $\varphi \in$ FLTL holds in an CLTS $E$ (denoted $E \models \varphi$) if it holds on every infinite trace produced by $E$. A *fluent fl* is defined by a set of

initiating actions $I_{fl}$, a set of terminating actions $T_{fl}$, and an initial value $Initially_{fl}$.

That is, $fl = \langle I_{fl}, T_{fl} \rangle_{initially_{fl}}$, where $I_{fl}, T_{fl} \subseteq \mathcal{P}(Act)$ and $I_{fl} \cap T_{fl} = \emptyset$.

In this paper we restrict attention to GR(1) [6] formulas as there are effective synthesis algorithms for this sub-logic [6]. GR(1) formulae are of the form $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond\phi_i \implies \bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$, where $\{\phi_1 \dots \phi_n\}$ and $\{\gamma_1 \dots \gamma_m\}$ are propositional FLTL formulae.

We now provide a standard definition of parallel composition of CLTSs to support compositional construction of environment models [19].

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

*Definition 3.4. (GR(1) CLTS control problem)* Let $I = \langle E, C, \mathcal{F}, \varphi \rangle$ be a CLTS control problem, $I$ is a GR(1) CLTS control problem if $\varphi$ satisfies:

$$\varphi = (\bigwedge_{i=1}^{k} \Box\Diamond\gamma_i \implies \bigwedge_{j=1}^{l} \Box\Diamond\psi_j)$$

In the previous definition $\gamma_1, \dots, \gamma_k$, $\psi_1, \dots, \psi_l$ are propositional LTL formulas over fluents that represent a set of assumptions over the environment and a set of guarantees the system should satisfy.

*Definition 3.5. ((Non) Controllable and Mixed States)* Given an CLTS $E = \langle S_e, \Sigma, \Delta_e, s_{e_0} \rangle$, a set of controllable actions $C \subseteq \Sigma$ and its complement $\mathcal{U} = \Sigma \setminus C$, a state $s \in S$ is called controllable if $\Delta(s) \cap C = \Delta(s)$, non controllable if $\Delta(s) \cap \mathcal{U} \neq \emptyset$ and mixed otherwise. We will refer to non mixed states pure states.

*Definition 3.6. (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, $\gamma^-, \gamma^+ \subseteq S \times S$ 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 $p = s_0, s_1, \dots$ 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 3.7. ((Counter)Strategy with memory)* A strategy with memory $\Omega$ for the controller is a pair of functions $(\delta, u)$ where $\Omega$ is some memory domain, $\delta : \Omega \times S \to 2^S$ such that $\delta(\omega, s) \subseteq \gamma^+(s)$ and $u : \Omega \times S \to \Omega$. A counter strategy with memory $\nabla$ for the environment is a pair of functions $(\kappa, \upsilon)$ where $\nabla$ is some memory domain, $\kappa : \nabla \times S \to 2^S$ such that $\kappa(\nabla, s) \subseteq \gamma^-(s)$ and $\upsilon : \nabla \times S \to \nabla$.

*Definition 3.8. (Consistency under (Non)Controllability and Winning (Counter)Strategy)* A finite or infinite play $p = s_0, s_1, \dots$ 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 $(\delta, u)$ for controller from state $s$ is winning if every maximal play starting in $s$ and consistent under controllability with $(\delta, u)$ is infinite and remains within $\varphi$. We say that the controller wins the game $G$ if it has a winning strategy from the initial state. For the non controllable case, a finite or infinite play $p = s_0, s_1, \dots$ 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 $(\kappa, \upsilon)$ for environment from state $s$ is winning if every maximal play starting in $s$ and consistent under non-controllability with $(\kappa, \upsilon)$ is infinite and exits $\varphi$ at some point. We say that the environment wins the game $G$ if it has a winning strategy from the initial state.

*Definition 3.9. (Generalized Reactivity(1))* Given an infinite sequence of states $p$, let $inf(p)$ denote the states that occur infinitely often in $p$, let $\phi_1, \dots, \phi_n$ and $\gamma_1, \dots, \gamma_m$ be subsets of $S$. Let $gr((\phi_1, \dots, \phi_n), (\gamma_1, \dots, \gamma_m))$ 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 $\varphi$ is $gr((\phi_1, \dots, \phi_n), (\gamma_1, \dots, \gamma_m))$.

*Definition 3.10. ($(G_{\mathcal{I}})$ Two-Player Game for $\mathcal{I}$ )* We convert the GR(1) CLTS control problem $\mathcal{I} = \langle E, C, \varphi \rangle$ to a two-player GR(1) game $G(\mathcal{I}) = (S_g, \gamma-, \gamma+, s_{g0}, \varphi)$ as follows: every state in $S_g$ encodes a state in $E$ and a valuation of all fluents appearing in $\varphi$. 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 $\varphi$, let $S_g = S_e \times \Pi_{i=0}^{k} \{true, false\}$. Consider state $s_g = (s_e, \alpha_1, \dots, \alpha_k)$, given fluent $fl_i$ we say that $s_g$ satisfies $fl_i$ if and only if $\alpha_i$ is $true$. We generalize satisfaction to boolean combination of fluents in the natural way. We build the transition relations $\gamma^-, \gamma^+$ through the following rules. Consider state $s_g = (s_e, \alpha_1, \dots, \alpha_k)$, for every transition $(s_e, l, s'_e) \in \Delta$ we include $(s_g, (s'_e, \alpha'_1, \dots, \alpha'_k))$ in $\gamma^\beta$ where $\beta$ is $+$ if $l \in C$ and in $\gamma^-$ otherwise. $\alpha'_i$ is $\alpha_i$ if $l \neg \in I_{fl_i} \cup T_{fl_i}$, $\alpha'_i$ is $true$ if $l \in I_{fl_i}$ and $false$ if $l \in T_{fl_i}$. If $s_e$ is a mixed state (i.e., $\Delta(s_e) \cap C \neq \emptyset \wedge \Delta(s_e) \cap \mathcal{U} \neq \emptyset$) we only include $(s_g, (s'_e, \alpha'_1, \dots, \alpha'_k))$ in $\gamma^-$ if $l \in \mathcal{U}$ since we will consider mixed states as non-controllable (antagonistic) in the game. $s_{g0}$ is $(s_0, Init_{fl_1}, \dots, Init_{fl_k})$. Let $\varphi_g \subseteq S_g^\omega$ be the set of sequences that satisfy $gr((\phi_1, \dots, \phi_n), (\gamma_1, \dots, \gamma_m))$.

## 4 BEHAVIOR MINIMIZATION FEEDBACK

In this section we formalize the problem of minimizing behavior while preserving unrealizability. In the next section we define a minimization procedure.

Assume an unrealizable specification of the form $\mathcal{I} =< E, C, \varphi >$. Our aim is to automatically produce an unrealizable specification $\mathcal{I}' =< E', C, \varphi >$ where $E'$ has less behavior than $E$ and the witnesses for the unrealizablity of $\mathcal{I}'$ (i.e., the counter strategies) are also witnesses for the unrealizability of $\mathcal{I}$. Furthermore, we aim for $E'$ to minimal in the sense that it cannot be further reduced while having its associated counter strategies preserved in $\mathcal{I}$. Thus, an engineer focusing on why $\mathcal{I}'$ is unrealizable can gain insight on why $\mathcal{I}$ is unrealizable.

*Definition 4.1. (Unrealizability Preservation)* Given an unrealizable control problem $\mathcal{I} =< E, C, \varphi >$ we say that $\mathcal{I}' =< E', C, \varphi >$ preserves unrealizability if for all winning strategy $f_1^{\neg\varphi}$ for the environment in $G(\mathcal{I}_P)$ then $f_1^{\neg\varphi}$ is winning for the environment in $G(\mathcal{I}_M)$.

In this section we define a minimization procedure that preserves unrealizability. We do this by defining a notion of alternating sub-CLTS that preserves counter strategies and forms a semilattice that can be used to incrementally minimize a non-realizable specification.

To define alternating sub-CLTSs we first define a sub-CLTS relation. A sub-CLTS is simply achieved by removing states and transitions from a CLTS while keeping its initial state.

*Definition 4.2. (Sub-CLTS)* Given $M = (S_M, \Sigma_M, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma_P, \Delta_P, s_{P_0})$ CLTSs, we say that $P$ is a sub-CLTS of $M$ (noted $P \subseteq M$) if $S_P \subseteq S_M$, $s_{M_0} = s_{P_0}$, $\Sigma_P \subseteq \Sigma_M$ and $\Delta_P \subseteq \Delta_M$.

To preserve unrealizability, the alternating sub-CLTS definition refines that of sub-CLTS by restricting some of the original CLTS transitions. These restrictions consider controllability.

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

- If $s$ is a pure controllable state then $\Delta_P(s) \cap C = \Delta_M(s) \cap C$
- If $s$ is not a pure controllable state then $\Delta_P(s) \cap \mathcal{U} \neq \emptyset$.

We shall omit $C$ and $\mathcal{U}$ in $\sqsubseteq_{C,\mathcal{U}}$ when the context is clear.

The definition above introduces restrictions on transitions of alternating sub-CLTSs. First, in states where all outgoing transitions are controllable, all transitions must be preserved. If this were not the case, then a counter strategy in $P$ with $P \sqsubseteq M$ may not work in $M$ because the controller in $M$ has more controllable actions that the counter strategy in $P$ does not account for. The second restriction states that in all states in which there is at least one uncontrollable action, then at least one uncontrollable action must be preserved. This is because for pure uncontrollable states, if this were not the case, a new deadlock in $P$ would be introduced. This would mean that an environment strategy could win by forcing that deadlock in $P$, however the same strategy when applied in $M$ would reach a non-deadlocking state and hence would not be winning. In mixed states, removing all uncontrolled transitions would allow a counter strategy in $P$ that relies, on reaching such state, to force the controller to move (as not doing so would result in a deadlock). However a strategy in $P$ exploiting this would not work in $M$ because reaching the same state, now a mixed state, the controller may defeat the same environment strategy by not picking any controlled action since the environment is obliged to progress in this case.

An informal justification for the definition of Alternating Sub-CLTSs can be given by showing that for two unrealizable control problems $\mathcal{I} =< E, C, \varphi >$ and $\mathcal{I}' =< E', C, \varphi >$ with $E' \sqsubseteq E$ then any counter strategy for the game $G(\mathcal{I}')$ is also a counter strategy for $G(\mathcal{I})$. For this we first show that there is a corresponding notion of Alternating Sub-Game and then use this to show that, indeed, the Alternating Sub-CLTS relation preserves counter strategies.

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

*Definition 4.5. (Alternating Sub-Game)* Given $G = (S_g, \Gamma^-, \Gamma^+, s_{g_0}, \varphi)$ and $G' = (S'_g, \Gamma^{-\prime}, \Gamma^{+\prime}, s'_{g_0}, \psi)$ 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: $\forall s'_g \in S'_g : \Gamma^+(s'_g) = \Gamma^{+\prime}(s'_g)$ and $\Gamma^-(s'_g) \neq \emptyset \rightarrow \Gamma^{-\prime}(s'_g) \neq \emptyset$.

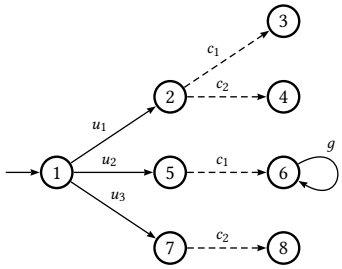


**Figure 7: Several conflicts example ($E$).**

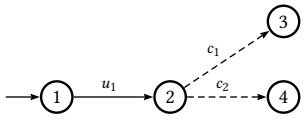


**Figure 8: Several conflicts example minimization($E_1$).**

Lemma 4.6. (Alternating Sub-CLTS implies Alternating Sub-Game) *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 $E_2 \sqsubseteq E_1$ implies $G(\mathcal{I}_2) \sqsubseteq G(\mathcal{I}_1)$.*

The following lemma states that removing transitions in the environment description of an unrealizable control problem, while preserving an Alternating Sub-CLTS relation, yields a new unrealizable control problem in which all its counter strategies are also counter strategies of the original control problem. Thus inspecting the behavior of the smaller environment description provides insights on the causes for unrealizability of the larger one.

Lemma 4.7. (Alternating Sub-CLTS preserves counter strategy) *Let $P$ and $M$ be two CLTSs s.t. $P \sqsubseteq M$, $\mathcal{I}_P = \{P, C, \varphi\}$, $\mathcal{I}_M = \{M, C, \varphi\}$ two control problems, if $f_1^{\neg\varphi}$ is a winning strategy for the environment in $G(\mathcal{I}_P)$ then $f_1^{\neg\varphi}$ is winning for the environment in $G(\mathcal{I}_M)$.*

We formally define the problem we solve as follows.

*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 $E' \sqsubseteq E$, $\mathcal{I}'$ is unrealizable $\mathcal{I}$, and that there is no $\mathcal{I}'' =< E'', C, \varphi >$ such that $E' \sqsubseteq E''$ and $\mathcal{I}''$ is unrealizable. We refer to $\mathcal{I}'$ as a minimal unrealizable control problem.

## 5 TWO BEHAVIOR MINIMIZATION PROCEDURES

The strategy to find a minimal non realizable control problem is to incrementally remove transitions preserving an alternating Sub-CLTS relation until a local minimum is reached. A minimal non realizable control problem is reached when any further reduction of the current Sub-CLTS renders the problem realizable. The first procedure explores the semi-lattice defined by Alternating Sub-CLTS relation starting with the original automaton and moving downwards by performing minimal reductions.

Consider the example of Figure 7 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 $C =$

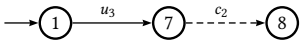


**Figure 9: Several conflicts example minimization($E_2$).**

$\{c_1, c_2, g\}$, the original CLTS $E$ will be at the top of the semi lattice while $E_1$ (depicted in figure 8) and $E_2$ (depicted in figure 9) will be at the bottom. In between there is, for example, the Alternating Sub-CLTSs obtained by removing just one non controllable transition. The automaton obtained by removing $\{u_1, u_3\}$ is not shown since it is realizable.

```
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 = E.prune_mixed_states() (A)
5        E_new = E' = E
6        T_u = E.non_controllable_transitions() (B)
7        to_remove = [] (C)
8        while(|T_u| > 0)
9              t = T_u.pop()
10             while(¬E'.contains(t) or d_out(E',t.from) == 1) (D)
11                   if (|T_u| == 0)
12                         return E
13                   t = T_u.pop() (E)
14             to_remove.add(t)
15             E_new = E.remove(to_remove) (F)
16
17             is_realizable = E_new.realizable() (G)
18
19             if (¬ is_realizable):
20                   E' = E_new
21             else
22                   to_remove.remove(t) (H)
23       return E' (I)
```

**Figure 10: Minimization Algorithm**

We can now describe the algorithm as presented in picture 10. The algorithm first removes all controllable transitions from mixed states (A), this step can be performed since it can be interpreted as the successive removals of controllable transitions with each satisfying the Alternating Sub-CLTS definition and preserving non realizability because if it was non realizable for the mixed state it will remain non realizable when the controllable transitions are removed since int the game built from the automaton at each mixed state the environment wins the race condition, always picking the next move when allowed. The next step initializes the set of candidate transitions for removal at (B) and the effective list to be removed at each iteration `to_remove` at (C). The latter keeps track of the transitions removed along our search down the space of Alternating Sub-CLTSs for $E$. The exploration will continue until the candidate set is empty. At each cycle starting at (D) the set is updated (E) by removing those transitions that were already removed by the reduction at (F) or because they would induce a deadlock ($d_{out}(E', t.from) == 1$ checks if there is only one outgoing transition in the state from which **t** originates). The reduction, at (F), looks for the closest Alternating Sub-CLTS obtained after removing transitions in `to_remove` from $E$. Then it checks for realizability (G). If the control problem is non realizable the candidate set and the last Alternating Sub-CLTS $E'$ are updated (since further minimization is possible), otherwise a one step backtrack is triggered by removing the transition under test from the effective set `to_remove` (H). This is actually the backtracking point where the algorithm shifts the search sideways in the semi-lattice instead of downwards. Once the candidate set has been completely explored the last non realizability preserving CLTS $E'$ is returned as a minimal Alternating Sub-CLTS for $E$ and $C$ (I). Notice that in the step (H) the transition $t$ was removed from the candidate set but was retained in $E'$. What happens if the transition could have been removed further down the semi lattice? Removing $t$ from $E'$ and reducing the Alternating Sub-CLTS $E_{new}$ accordingly would render the control problem realizable. Suppose that the search continued preserving $t$ and removing other candidates $t_1, \ldots, t_n$ from $E'$ getting the Alternating Sub-CLTS $E_{1,\ldots,n}$ 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-CLTS we would get a new CLTS $E'_{new}$ that can be Alternating Sub-CLTS for $E_{new}$, and from lemma 5.1 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 jeopardizing 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-CLTS $E'$ 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 $E' \sqsubseteq E$ 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-CLTS is realizable at the final iteration in conjunction with lemma 5.1.

We introduced two modifications to the algorithm to improve the time taken to compute the minimization. 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 input automaton $E$, that is, sequences of non controllable actions without branches. 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 CLTS all the others have to be removed as well to preserve alternation.

Lemma 5.1 shows that we can keep removing elements from a set $\mathcal{T}_u$ of candidate non controllable transitions, reducing $E$ to its closest Alternating Sub-CLTS. Once we can not, while preserving non-realizability, remove anything else we have reached an Alternating Sub-CLTS that is also minimal. This follows from the fact that no closest Alternating Sub-CLTS exists bellow it that preserves non realizability. If all the closest Alternating Sub-CLTSs in its neighborhood are realizable the current CLTS is minimal.

Lemma 5.1. *(Alternating Sub-CLTSs preserve realizability) Let $E$ be the CLTS for the non realizable control problem $\mathcal{I} =< E, C, \varphi >$,*

*and $E_1$, $E_2$ two CLTSs s.t. $E_2 \sqsubseteq E_1 \sqsubseteq E$ 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.*

The second procedure uses the Delta Debugging technique presented in [43] to accelerate the search space exploration. Instead of reducing the automaton by applying an atomic reduction at a time Delta Debugging tries to achieve a minimal diagnosis by partitioning the problem $\Delta$ into $n$ uniform subsets ($\Delta = \bigcup_{i=1}^{n} \Delta_i$) and then it first checks its elements $\Delta_i$ for a non realizable smaller case, if none is found it proceeds by looking at the complement $\Delta \setminus \Delta_i$ of each element in the partition. If no smaller candidate was found it increases the granularity by updating $n$ as $max(2n, |\Delta|)$. This cycle is repeated until $n$ can not be further increased ($n \geq |\Delta|$). In our application of Delta Debugging the set $\Delta$ is the set of non controllable transitions in the original plant. Instead of descending regularly down the semi lattice, Delta Debugging tries to jump further down by removing a subset of transitions from the current candidate. We can briefly prove that the application of the technique is sound since Delta Debugging ensures 1-minimality, i.e. that removing any single element from the resulting minmization breaks non realizability which in conjunction with lemma 5.1 implies minimality.

## 6 VALIDATION

The extent to which the technique serves it purpose is defined by its soundness, efficiency and, more importantly, information, understood as the ability to guide the engineer while writing a non trivial formal specification. The presented diagnosis should help by pointing out the causes of non realizability (expressed in a behavioral implicit language) that may appear while progressively gathering knowledge and building a complete and precise model.

The measure of information our technique may provide should ultimately be related to the impact it can have on the overall specification process, in particular how clear it is for the engineer to interpret the diagnosis in terms of amount of data presented and its relevance in relation to the non realizability cause. The unavailability of skilled professionals and the amount of time and effort required to prepare a considerable set of external subjects to the point where they would be able to write non trivial formal specifications renders the perspective of a controlled experiment with external engineers writing specifications from scratch unfeasible. On the other hand, it is also impossible to propose a direct comparison with similar techniques, since, for instance, in [27] the minimization is applied on the set of environmental safety formulas, which in fact increments the volume of the plant when presented as an automaton, and in [29] a simplification of the symbolic counter strategy is presented as a LTS composed of attractors and trap states. We think that our technique adds a complimentary view to the non realizability diagnosis problem.

We now explain two of the metrics used to quantitatively evaluate our technique. The first metric is relative minimization volume ($v_{\mathcal{U}}$), defined as the size of the minimized plant against the original instance ($|\Delta_{E'}|/|\Delta_E|$). The number of transitions is used as the representation of absolute volume since it is the defining parameter in the complexity of the synthesis algorithms. The other metric we compute is minimization volume relative to the controller volume ($v_C = |\Delta_{E'}|/|\Delta_C|$). The controller $C$ is the automaton representation of the strategy for the realizable version of each specification. The rationale behind this is that the size of the controller is a good proxy for the volume of the engineer's design intent in terms of concrete behavior. The motivation for taking the controller as a proxy of volume is that sometimes an ongoing specification is under defined, with weaker conditions than it will eventually have when reaching realizability. We expect for our minimization to be at least comparable to the controller in terms of volume.

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 non realizable specifications present in the existing benchmarks and literature.

The Generic Buffer case study was taken from [27], the Lift Controller comes from [2], Collector is part of the SYNTCOMP competition [1] benchmark and the exploration robot case was adapted from [44].

The diagnosis technique was implemented in a new tool that accepts specifications expressed as both LTL formulas or FSP constructs and operates internally on CLTS structures. Each case study was written in its original syntax, for signal based specifications we used LTL formulas (as presented in [5]), and for those initially described using process-like syntax we used CFSP. This tool was written from scratch with the intention to model, verify and synthesize reactive systems over enumerative models in an effective fashion. Bear in mind that for each specification the diagnosis runs a number of synthesis checks (each one having its particular time complexity) linear in time in relation to the number of transitions of the original plant. Prior to this work the only feedback given to the user when specifying an non realizable problem over an enumerative model was binary, either the problem was realizable or it was not. he extension builds a minimized non realizable version of the environment that can then be exported to different formats and inspected through various methods including animations and visualization of the state space.

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 briefly explain the case studies and provide an interpretation of what we think are the hints extracted from the diagnosis.

The quantitative results of our experiments are shown in table 1 and were run on an `Intel® Core™ i7-8565U` CPU with 8 processors running at 1.80GHz frequency with 32 Gb of RAM memory over `Ubuntu 20.04 (Linux 5.4 kernel release)`. Total time taken by the diagnosis algorithm is measured in seconds and labeled as *Diag. time(s)*, *Diag. steps* shows the number of steps the diagnosis algorithm had to take to yield a minimal plant, this can be thought as de maximum depth of recursion reached while exploring the semi-lattice of alternating Sub-LTS structures. The values in column $|\varphi_e + \varphi_s|$ are the sum of environmental and system liveness formulae, $|\Delta_E|$ the number of transitions in the original plant, $|\Delta_{E'}|$ the number of transitions in the minimization, , $|\Delta_{E'}|/|\Delta_E|$ is the relation of the minimization volume against the original plant, , $|\Delta_C|$ the number of transitions in the controller for the realizable

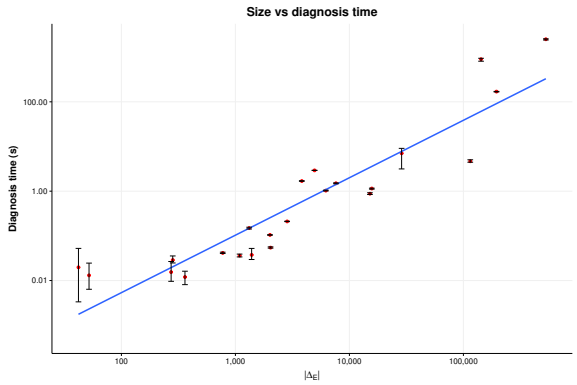


**Figure 11: Minimization size vs. diagnosis time.**

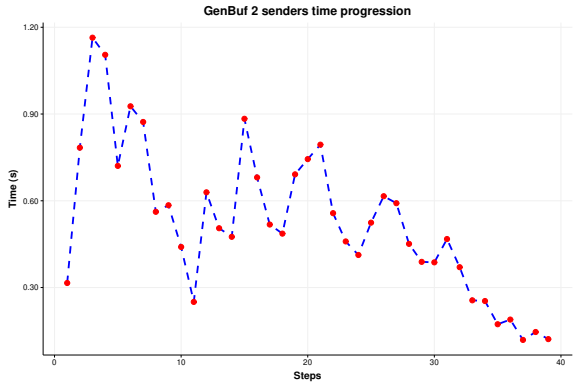


**Figure 12: Generic buffer (2 senders) time progression.**

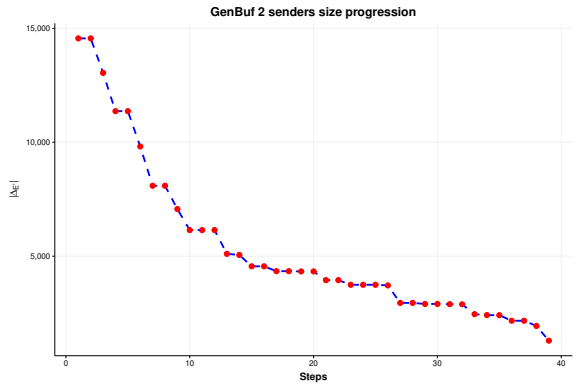


**Figure 13: Generic buffer (2 senders) size progression.**

specification, , $|\Delta_{E'}|/|\Delta_C|$ is the relation of the minimization volume against the controller.

The results we observe after applying the technique on this initial set of case studies show the proposed approach is feasible even for a significant number of signal based specifications. As expected with any algorithm working with enumerative models, the minimization technique still under performs against symbolic approaches on both size and time. In figure 11 we expose the relation between diagnosis time and plant size (measured in number of initial transitions). We run each instance seven times and reported the mean

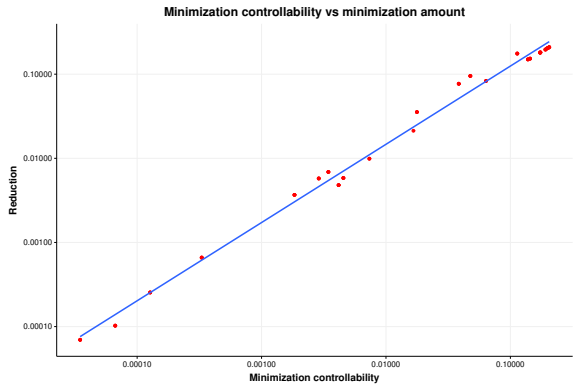


**Figure 14: Minimization controllability vs. minimization percentage.**

as both the time value for the table and the central dot, the error bars show the distance to the maximum and minimum time in each case. We argue that the performance of the algorithm is linearly consistent (the lineal fit having a p-value of $< 3.387e^{-11}$) across the range of sizes present in the test set (going from tens to hundred thousands of transitions). We show the behavior of the technique while diagnosing the first non realizable version of the generic buffer for two senders, figure 12 shows the time taken at each step of the minimization and fig. 13 shows the incremental reduction of the plant as the minimization progresses. We also found that the amount of minimization achievable seems to be related to how controllable the result expressed in figure 14 as the number of controllable transitions in the minimization over the number of total transitions. The relation seems to fit a lineal model (p-value $< 2.2e^{-11}$). 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. Even when the reduction is significant in the bigger examples, as the 0.4% reduction for 528650 transitions in the biggest generic buffer case, the resulting 2539 transitions are still enough to make manual inspection difficult. We assume that in this cases the technique can be used as the initial step of the diagnosis, allowing for other techniques to be applied afterwards, such as existential quantification or collapsing $\sigma$-traps.

## 7 DISCUSSION AND RELATED WORK

Various approaches to providing feadback from unrealizable specifications exist. In [26] authors provide a technique that reduces the specification while preserving non-realizability. The specification is provided as a set of LTL formulas and the feedback is a minimal subset of the formulas that continues to be unrealizable. Other approaches that follow this form of minimization include [40].

The approach presented in this paper differs feedback for specifications described using automata with an asynchronous execution semantics while in [26] the environment and system execute synchronously (i.e., in lockstep). It is possible to model the same problem in one setting or the other, or translate specifications from one to another, however problems in different domains can be more naturally described into one or the other setting.

| | Diagnosis | | Plant ($E$) | | Minimization ($v_{\mathcal{U}}$) | | Controller ($v_C$) | |
|---|---|---|---|---|---|---|---|---|
| Name | Diag. time(s) | Diag. steps | $\lvert\varphi_e + \varphi_s\rvert$ | $\lvert\Delta_E\rvert$ | $\lvert\Delta_{E'}\rvert$ | $\lvert\Delta_{E'}\rvert/\lvert\Delta_E\rvert$ | $\lvert\Delta_C\rvert$ | $\lvert\Delta_{E'}\rvert/\lvert\Delta_C\rvert$ |
| Lift Controller 2 (missing assumption) | 0.013 | 3 | 5 | 52 | 4 | 0.0769 | 116 | 0.4483 |
| Lift Controller 4 (missing assumption) | 0.036 | 8 | 7 | 1088 | 4 | 0.0037 | 3238 | 0.3360 |
| Lift Controller 6 (missing assumption) | 1.147 | 12 | 9 | 15720 | 4 | 0.0003 | 61400 | 0.2560 |
| Lift Controller 8 (missing assumption) | 169.294 | 17 | 11 | 195040 | 20 | 0.0001 | 937004 | 0.2082 |
| Collector 1 (missing assumption) | 0.020 | 4 | 2 | 42 | 4 | 0.0952 | 32 | 0.1250 |
| Collector 2 (missing assumption) | 0.029 | 17 | 2 | 282 | 10 | 0.0355 | 207 | 0.0483 |
| Collector 3 (missing assumption) | 0.054 | 20 | 3 | 2034 | 14 | 0.0069 | 1424 | 0.0098 |
| Collector 4 (missing assumption) | 0.878 | 38 | 4 | 15138 | 10 | 0.0007 | 10326 | 0.0010 |
| Collector 5 (missing assumption) | 4.676 | 43 | 5 | 114882 | 8 | 0.0001 | 77306 | 0.0001 |
| Exploration Robot 12 (missing assumption) | 0.149 | 6 | 3 | 1320 | 202 | 0.1530 | 162 | 1.2469 |
| Exploration Robot 15 (missing assumption) | 0.105 | 2 | 3 | 2010 | 365 | 0.1816 | 320 | 1.1406 |
| Exploration Robot 18 (missing assumption) | 0.210 | 2 | 3 | 2844 | 512 | 0.1800 | 456 | 1.1228 |
| Exploration Robot 21 (missing assumption) | 1.685 | 7 | 3 | 3822 | 775 | 0.2028 | 698 | 1.1103 |
| Exploration Robot 24 (missing assumption) | 2.922 | 7 | 3 | 4944 | 970 | 0.1962 | 894 | 1.0850 |
| Exploration Robot 27 (missing assumption) | 1.030 | 2 | 3 | 6210 | 1301 | 0.2095 | 1220 | 1.0664 |
| Exploration Robot 30 (missing assumption) | 1.503 | 2 | 3 | 7620 | 1568 | 0.2058 | 1476 | 1.0623 |
| Exploration Robot 6 (missing assumption) | 0.012 | 2 | 3 | 361 | 30 | 0.0831 | 174 | 0.1724 |
| Exploration Robot 9 (missing assumption) | 0.041 | 6 | 3 | 774 | 116 | 0.1499 | 86 | 1.3488 |
| Genbuf 1 (missing assumption) | 0.015 | 6 | 4 | 273 | 48 | 0.1758 | 307 | 0.8893 |
| Genbuf 2 (missing assumption) | 7.050 | 42 | 5 | 28706 | 610 | 0.0137 | 78008 | 0.3680 |
| Genbuf 3 (missing assumption) | 2513.970 | 126 | 6 | 528650 | 2539 | 0.0048 | 1918013 | 0.2756 |
| Genbuf 1 (removed safety) | 0.037 | 10 | 5 | 1388 | 8 | 0.0058 | 307 | 0.0261 |
| Genbuf 2 (removed safety) | 897.961 | 194 | 8 | 142850 | 834 | 0.0058 | 78008 | 0.0107 |

**Table 1: Quantitative results for minimized plants**

However, a more fundamental difference between this work and that of [26] is that here we minimize behavior rather than specification. Reducing an LTL specification by removing transition means that there are more behaviors that the environment can exhibit. Indeed, an engineer inspecting a winning environment strategy for a reduced specification as in [26] may be reasoning about behavior that is not allowed in the original specification.

An alternative approach to providing unrealizability is to allow inspecting counter strategies. In [8] 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 [27] counter-strategy and it is restricted to scenario-based specifications that essentially encode LTL formulas. In [29] the authors present justice violation transition systems for LTL specifications as a way to abstract and simplify the winning strategy for the environment. States in the JVTSs are labeled with invariants, since they collapse states of the counter strategy in order to expose relevant environmental decisions. Inspection of counter strategies that build on minimization of LTL specifications [8, 29] the fact that the behavior of the specification being analyzed has increased. Thus, the inspection of the counter strategy may lead an engineer to reasoning about behavior that is not allowed in the original specification.

Note that our approach also differs from depicting as an CLTS the winning strategy for the environment. The CLTS modeling the counter strategy is not a sub-CLTS of $E$ as the strategy must remember which assumption was the last to be satisfied. In general, the worst case is that the strategy of the environment will have $n$ times more states than the minimal sub-CLTS that can be used to generate it, where $n$ is the number of liveness assumptions that the environment must satisfy. Furthermore, the sub-CLTS we produce can allow for more than one counter strategy and hence provide insight to more than one problem with the original specification.

The idea of [24] 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 [23] for weighted transition systems.

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 [28] for CTL* specifications, in [25] for the GR(1)[37] subset, further explored in [33] for the declarative version and in [14] 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 [9] and [10], among 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 [38]), 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 [13] 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 [17] 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], [32] and [7] 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 under specified variables. In [32], the authors propose mining assumptions out of an unrealizable GR(1) specification through the use of a counter strategy from [27]. Again, a template based approach is used. In [7] an assumption computing technique is presented based in Craig's interpolants is presented. It obtains refinements by negating plays of the counter-strategy.

## 8 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a technique that 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 CLTS automata. We believe that our work complements pre-existing approaches bases on LTL minimization. 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] 2020. The Reactive Synthesis Competition. (2020). http://www.syntcomp.org/

[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] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. 2010. RATSY - A New Requirements Analysis Tool with Synthesis. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings (Lecture Notes in Computer Science)*, Tayssir Touili, Byron Cook, and Paul B. Jackson (Eds.), Vol. 6174. Springer, 425–429. DOI: http://dx.doi.org/10.1007/978-3-642-14295-6_37

[5] 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

[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] 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

[8] 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

[9] 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.

[10] 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.

[11] 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).

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

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

[20] 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

[21] 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

[22] 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

[23] 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

[24] Kangjin Kim, Georgios E. Fainekos, and Sriram Sankaranarayanan. 2012. On the revision problem of specification automata. In *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 5171–5176. DOI: http://dx.doi.org/10.1109/ICRA.2012.6224826

[25] 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

[26] 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

[27] 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

[28] 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.

[29] 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

[30] Emmanuel Letier and William Heaven. 2013. Requirements Modelling by Synthesis of Deontic Input-output Automata. In *Proc. of the 2013 Int. Conf. on Software Engineering (ICSE '13)*. IEEE Press.

[31] 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

[32] 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

[33] 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

[34] 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/

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

[36] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. 2004. Planning and Monitoring Web Service Composition. In *Artificial Intelligence: Methodology, Systems, and Applications*, Christoph Bussler and Dieter Fensel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 106–115.

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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.

[42] 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

[43] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Trans. Software Eng.* 28, 2 (2002), 183–200. DOI: http://dx.doi.org/10.1109/32.988498

[44] Sebastián Zudaire, Martín Garrett, and Sebastián Uchitel. 2020. Iterator-Based Temporal Logic Task Planning. *CoRR* abs/2001.07678 (2020). arXiv:2001.07678 https://arxiv.org/abs/2001.07678


# A PROOFS

The appendix includes proofs and is for reviewer's convenience. The appendix would not be part of the published paper if accepted.

Hay que reveer

Proof for Lemma 4.6

**HAY QUE REVER ESTA PORQUE CAMBIO LA DEFINICI‰ON DE SUBLTS**

Proof. $\mathcal{I}_1$ and $\mathcal{I}_2$ share the same winning condition so they also share fluents definition. Since $E_2 \sqsubseteq E_1$ 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 $S_{g_2} \subseteq S_{g_1}$. We can see that following the definition of the game $G(\mathcal{I}_1)$ constructed from $\mathcal{I}_1$ if $\Delta_2 \subseteq \Delta_1$ and $s_{g_1} = (s_e, \alpha_1, \ldots, \alpha_k) \in S_{g_1}$ for all $(s_e, l, s'_e) \in \Delta_1$ the transition $(s_{g_1}, (s'_e, \alpha'_1, \ldots, \alpha'_k))$ will be added to $\Gamma_1^+$ if $l \in \mathcal{C}$ or to $\Gamma_1^-$ if $l \in \mathcal{U}$. 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 $E_2 \sqsubseteq E_1$ we know, starting with pure states, that for all $s \in S_2, \Delta_1(s) \cap \mathcal{U} = \emptyset$, $l \in \mathcal{C}$ if $(s, l, s') \in \Delta_1$ then $(s, l, s') \in \Delta_2$ and also for every state $s_{g_1}$ related to $s$ in the game: $s_{g_1} \in s \times \Pi_{i=0}^{k}\{true, false\}$ and for all $l \in \mathcal{C}$ such that $(s, l, s') \in \Delta_1$, $(s, l, s') \in \Delta_2$ and for $s'_{g_1} = (s', \alpha'_1 \ldots, \alpha'_k)$, $(s_{g_1}, s'_{g_1}) \in \Gamma_1^+$ and $(s_{g_1}, s'_{g_1}) \in \Gamma_2^+$ hold, thus implying $\Gamma_1^+(s_{g_1}) = \Gamma_2^+(s_{g_1})$. If $s$ is a mixed state in $E_1$, following the construction of $G(\mathcal{I}_1)$ we know that $\Gamma^+(s, \alpha'_1 \ldots \alpha'_k) = \emptyset$, from the definition of Sub-LTS $s$ will be a mixed state in $E_2$ and $\Gamma^{+\prime}(\alpha'_1 \ldots \alpha'_k) = \emptyset$ implies that $\Gamma^+(s_g) = \Gamma^{+\prime}(s_g)$. If $s$ is not a mixed state and non controllable for all $s \in S_2$ if there exists an $l \in \mathcal{U}$ such that $(s, l, s') \in \Delta_1$ then from $E_2 \sqsubseteq E_1$ it must exists an $l' \in \mathcal{U}$ such $(s, l', s'') \in \Delta_2$. 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 $(s_{g_1}, s'_{g_1}) \in \Gamma_1^-$ and for $l'$ in $\Delta_2$, let $s''_{g_1} = (s'', \alpha''_1 \ldots, \alpha''_k)$, we have $(s_{g_1}, s''_{g_1}) \in \Gamma_2^-$ implying $\Gamma_1^-(s_{g_1}) \neq \emptyset \rightarrow \Gamma_2^-(s_{g_1}) \neq \emptyset$. If $s$ is a mixed state there exists $l' \in \mathcal{U}$ s.t. $(s, l', s'') \in \Delta_1$ and then it holds that $(s_{g_1}, s'_{g_1}) \in \Gamma_1^-$. Following Sub-LTS definition there must exist $l'' \in \mathcal{U}$ s.t. $(s, l'', s'') \in \Delta_2$ implying the existence of $(s_{g_1}, s_{g''_1}) \in \Gamma_2^-$. □

Proof for Theorem4.7

**HAY QUE REVER ESTA PORQUE CAMBIO LA DEFINICI‰ON DE SUBLTS**

Proof. By way of contradiction suppose that $P$ is Alternating Sub-LTS of $M$ and $\mathcal{C}$ and that there exists a counter strategy in $G(\mathcal{I}_P)$ that loses $G(\mathcal{I}_M)$. Let $\pi$ 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 $G(\mathcal{I}_M)$. $\pi$ 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 $\pi = s_0 \dots s_a \dots$ 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 $s_a$ ($\pi = s_0 \dots s_b \dots s_a \dots$) where the play stepped out of $S_{G_P}$. If $s_b$ is non controllable 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 $G(\mathcal{I}_P)$. □

Proof for Lemma 5.1

**HAY QUE REVER ESTA PORQUE CAMBIO LA DEFINICI‰ON DE SUBLTS**

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 $\sigma$-trap that always falsifies the property $\varphi$. 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 $\pi$ 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 non controllable transition or removing all of them at a non controllable 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 between Alternating Sub-LTSs. □