Behaviour Minimization Feedback for Unrealizable Specifications

M. Cerrutti[+], V. Braberman[+], N. D'Ippolito[+], S. Uchitel[+†]
[+]Departamento de Computación, FCEN, Universidad de Buenos Aires, Argentina
[†]Departament of Computing, Imperial College, London

***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 them 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. This is the first technique that is designed to be used to provide feedback for GR(1) synthesis problems in which the environment is described as an automaton and the goals as liveness LTL formulas. 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.

I. 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 [1], [2]. 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 [2]. 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 [3].

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 [4], FOND planning [5] and controller synthesis [6], [7]. 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 unrealisable, the engineer is left with the onerous task of understanding why no such strategy exists and then with the task of changing the specification. Techniques for providing feedback on unrealizability have been proposed [8]–[10]. 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 [11], 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 [12] as in [13], statecharts [14] as in [15], and process algebra [16], [17] as in [18]) would require a non-trivial property extraction and/or a very large sets of formulae encoding every state transition.

In this paper, we propose a novel technique that provides feedback on unrealisable specifications. The technique does not require a declarative LTL representation of the specification and does not work through syntactic minimisation. Instead, the approach works on the state space of the specification, reducing allowable behaviour while preserving unrealisability. The result is a semantically minimised specification model in which the cause for unrealisability is the same as in the original specification.

Behaviour minimisation that preserves the cause of unrealisability can be thought of as follows: 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 unrealisable, 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 actioning over its actuators, it cannot achieve its goals even if the environment behaves according to its assumptions. Reducing the specification while preserving one of its causes of unrealizabilty is to produce a new specification

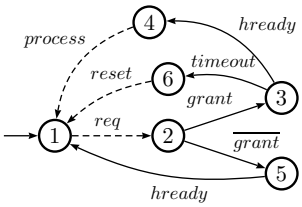

Fig. 1. Bus Access example.

which allows less behaviour yet in which the strategy for the environment that prevents the system from achieving its goals also works in the original specification.

In addition to being the first unrealisability feedback technique that can be applied to synthesis problems defined with automata, the technique may have 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 minimisation and non-realizability preservation. In Section V we present the minimisation algorithm and then discuss validation of our technique. We then proceed with a discussion on related work and conclusions.

II. Motivation

We motivate and introduce informally relevant concepts by discussing a small example inspired by [24]. 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 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 timeout, 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.

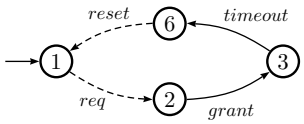

Fig. 2. Bus Access example (E_1).

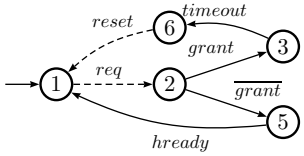

Fig. 3. Bus Access example (E_2).

The specification E_1 in Figure 2 is a subgraph of E, describing an environment in which strictly less behaviour 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 behaviour that is irrelevant with respect to a cause for unrealizability.

Imagine that after analyzing this first unrealisability 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 the next. In general, the worst case is that the strategy of the environment will have a times more states than the minimal subLTS that can be used to generate it, where A is the number liveness assumptions that the environment must satisfy.

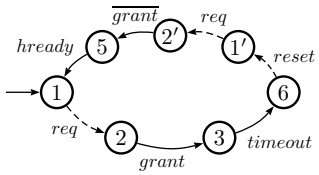

Fig. 4. Bus Access counter-strategy.

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

III. Preliminaries

In this section we present background on realisability, which is linked to the problem of controller synthesis [19], [20] which, in turn, requires a formal specification that, in this paper, is assumed to be, as in [25] provided as a labelled transition system and a subset of linear temporal logic formula called Generalised Reactivity 1 [20].

We assume the safety assumptions and goals to be described as a labelled transition system. This transition system may be provided using textual notation (for instance, process algebra [22] or the safety subset of linear temporal logic) and possibly in a compositional fashion, separating goals from assumptions.

Definition III.1. *(Labelled Transition Systems)* A *Labelled Transition System* (LTS) is $P = (S,\ \Sigma,\ \Delta,\ s_0)$, where S is a finite set of 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 $\Delta(s) = \{s' \mid (s, a, s') \in \Delta\}$. We say an LTS is deterministic if (s, ℓ, s') and (s, ℓ, s'') are in Δ implies $s' = s''$. An execution of P is a word $s_0, a_0, s_1, \ldots$ where $(s_i, a_i, s_{i+1}) \in \Delta$. A word π is trace of P if there is an execution ε of P such that $\varepsilon|_\Sigma = \pi$. We define TrP to define the set of traces of P.

A trace of E is $\pi = s_0, \ell_0, s_1, \ell_1, \cdots$, where s_0 is an initial state of E and, for every $i \geq 0$, we have $(s_i, \ell_i, s_{i+1}) \in \Delta$. We denote the set of infinite traces of E by $Tr(E)$.

We describe liveness goals and assumptions using fluent linear temporal GR(1) formulae. Linear temporal logics (LTL) are widely used to describe behaviour requirements [**?**]. Fluent LTL is a linear-time temporal logic for reasoning about fluents. 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 $I_{fl}, T_{fl} \subseteq Act$ and $I_{fl} \cap T_{fl} = \emptyset$. When we omit $Initially_{fl}$, we assume the fluent is initially *false*. We use $\dot{\ell}$ as short for the fluent defined as $fl = \langle \ell, Act \setminus \{\ell\}\rangle$.

Given the set of fluents Φ, an FLTL formula is defined inductively using the standard boolean connectives and temporal operators $\mathbf{X}$ (next), $\mathbf{U}$ (strong until) as follows: $\varphi ::= fl \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi$, where $fl \in \Phi$. We introduce $\wedge$, $\mathbf{F}$ (eventually), and $\mathbf{G}$ (always) as syntactic sugar.

Let Π be the set of infinite traces over Act. For $\pi \in \Pi$, we write π^i for the suffix of π starting at a_i. The suffix π^i satisfies a fluent fl, denoted $\pi^i \models fl$, if and only if one of the following conditions holds:

- $Initially_{fl} \wedge (\forall j \cdot 0 \leq j \leq i \Rightarrow a_j \notin T_{fl})$
- $\exists j \cdot (j \leq i \wedge a_j \in I_f) \wedge (\forall k \in \mathbb{N} \cdot j < k \leq i \Rightarrow a_k \notin T_{fl})$

The sublogic referred to as GR(1) allows liveness formulae of the form:

$$\varphi = \bigwedge_{i=1}^{n} \Box\Diamond\phi_i \implies \bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$$

where $\{\phi_1 \ldots \phi_n\}$ represent the set of assumptions that should always eventually be satisfied in order to always eventually satisfy the set of goals $\{\gamma_1 \ldots \gamma_m\}$. GR(1) is an expressive subset that has gained increased interest recently due to the development of tractable synthesis algorithms for it [20] and has been used in the context of EXAMPELS OF DOMAINS AND REFERENCES TO THEM.

To define controller synthesis a notion of parallel composition of LTS is needed. We use a standard definition inspired from [22] which is defined as an LTS that models the asynchronous execution of composed models, interleaving non-shared actions but forcing synchronisation on shared actions.

Definition III.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 Interface Automata [23]: A controller does not block the actions that it does not control. Intuitively, it says that M is a legal environment for E if in every state (m, e) of $M \| E$ where m and e are states of M and E respectively, if an action a not controlled by M is enabled in e then it is also enabled in (m, e).

Definition III.3. *(Legal Environment)* Given $M = (S_M,\ \Sigma_M, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma_P, \Delta_P, s_{P_0})$ LTSs, where $L_M = \Sigma_{M_c} \cup \Sigma_{M_u}$, $\Sigma_{M_c} \cap \Sigma_{M_u} = \emptyset$, $\Sigma_P = \Sigma_{P_c} \cup \Sigma_{P_u}$ and $\Sigma_{P_c} \cap \Sigma_{P_u} = \emptyset$. We say that M is a legal environment for P if the interface automaton $M' = \langle S_M, \{s_{M_0}\}, \Sigma_{M_u}, \Sigma_{M_c}, \emptyset, \Delta_M\rangle$ is a *legal environment* for the interface automaton $P' = \langle S_P, \{s_{P_0}\},\ \Sigma_{P_u}, \Sigma_{P_c}, \emptyset, \Delta_P\rangle$.

We can now define the synthesis control problem formally for formulas of the form $\bigwedge_{i=1}^{n} \Box\Diamond\phi_i \implies \bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$.

Definition III.4. *(LTS Control)* Given a specification for a problem domain in the form of an environemnt LTS E, a set of controllable actions $\mathcal{C}$, and a LTL formula $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond\phi_i \implies \bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$ where $\bigwedge_{i=1}^{n} \Box\Diamond\phi_i$ and $\bigwedge_{j=1}^{m} \Box\Diamond\gamma_j$ are LTL formulas specifying assumptions and goals respectively, the solution for the LTS control problem $\mathcal{I} = \langle E, \mathcal{C}, \varphi\rangle$ is to find an LTS M with controlled actions $\mathcal{C}$ and uncontrolled $\mathcal{U}$ 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 $\pi \models \varphi$.

Although the synthesis problem for general FLTL goals is 2EXPTIME complete [19]. Nevertheless, restrictions on the form of the goal and assumptions specification have been studied and found to be solvable in polynomial time. The formulation above, which is restricted to GR(1), has a polinomial solution [24]. An adaptation of GR(1) in the context of LTS has been presented in [25]

IV. Behaviour Minimisation Feedback

We first formalise the problem we aim to solve and then discuss a solution for it.

A. Problem Statement

Assume an unrealizable specification of the form $< E, \mathcal{C}, \varphi >$ where E is an LTS that describes safety assumptions and goals, $\mathcal{C}$ describes the set of events that the system can control, and where φ is a GR(1) property that encodes liveness assumptions and goals. 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 is meant by a reduced specification and preservation of unrealisability and then explain how such a specification can be produced automatically.

If $\mathcal{I} =< E, \mathcal{C}, \varphi >$ is a non realizable control problem then there is no LTS M that is a legal environment with respect to E such that $E||X \models \varphi$. Which means that (from Property **??**) there is a winning strategy S_E for the environment such that there is no M legal with respect to E that can achieve φ (i.e., $M \| \overline{S_E} \models \varphi$. We aim to build a minimal unrealisable specification $\mathcal{I}' =< E', \mathcal{C}, \varphi >$ such that any winning strategy $S_{E'}$ for $\mathcal{I}'$ is also a winning strategy for $\mathcal{I}$

We define minimality over the partial order defined by a notion inspired from sub-graph **??**

Definition IV.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 $\Delta_P \subseteq \Delta_M$.

We formally define an unrealisability preserving control problem as follows:

Definition IV.2. *(Unrealisability Preservation)* GIVEN A CONTROL PROBLEM I, WE SAY THAT I' preserves unrealisability of I IF ALL ENVIRONMENT WINNING STRATEGIES OF I' ARE ALSO ENVIRONMENT WINNING STRATEGIES OF I

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

Definition IV.3. *(Problem Statement)* Given a unrealisable control problem $\mathcal{I} =< E, \mathcal{C}, \varphi >$, find an unrealizable control problem $\mathcal{I}' =< E', \mathcal{C}, \varphi >$ such that $E \subseteq E'$, $\mathcal{I}'$ preserves unrealisability of $\mathcal{I}$, and that there is no $\mathcal{I}'' =< E'', \mathcal{C}, \varphi >$ such that $E' \subseteq E''$ and $\mathcal{I}''$ preserves unrealisability of $\mathcal{I}$,

B. Behaviour Minimisation

We first show that to preserve unrealizability it is sufficient to satisfy inclusion of E_κ in E and require a notion of legality for E_κ w.r.t. E and $\mathcal{C}$. The inclusion requirement is self explanatory since we want to stick with the behavior as originally specified, legality 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 *non realizability* legal definition (IV.4).

Definition IV.4. *(Non Realizability Legal Sub-LTS)* Given $M = (S_M, \Sigma, \Delta_M, s_{M_0})$ and $P = (S_P, \Sigma, \Delta_P, s_{P_0})$ LTSs, where $\Sigma = \mathcal{C} \cup \mathcal{U}$, $\mathcal{C} \cap \mathcal{U} = \emptyset$ and $P \subseteq M$. We say that P is a non realizability legal sub-LTS of M if $\forall s_P \in S_P$ the following holds: $(\Delta_M(s_P) \cap \mathcal{U} = \emptyset \rightarrow \Delta_P(s_P) \cap \mathcal{C} = \Delta_M(s_P) \cap \mathcal{C}) \wedge (|\Delta_M(s_P)| > 0 \rightarrow |\Delta_P(s_P)| > 0)$.

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

Theorem IV.1. (Legality and Non Realizability preserves Counterstrategy) *Let P be non realizability legal w.r.t. M and $\mathcal{C}$, if there exists $f_1^{\neg\varphi}$ a winning strategy for the environment in $\mathcal{I}_P = \langle P, \mathcal{C}, \varphi \rangle$ then $f_1^{\neg\varphi}$ is winning for the environment in $\mathcal{I}_M = \langle M, \mathcal{C}, \varphi \rangle$.*

Proof. By way of contradiction suppose that P is legal w.r.t. M and $\mathcal{C}$ and that there exists $f_1^{\neg\varphi}$ a winning strategy for the environment in $\mathcal{I}_P$ but not winning in $\mathcal{I}_M$. Let π be a play $s_0 l_0 s_1 l_1 \ldots$ conforming to $f_1^{\neg\varphi}$ that wins in P but loses in M. π leading to a finite win in P at $s_\perp$ (not a finite state in M) will be prevented by the legality requirement where $\forall s : s_P \in S_P : |\Delta_M(s_p)| > 0 \rightarrow |\Delta_P(s_p)| > 0$. If the environment has an infinite win in P not feasible in M and since $P \subseteq M$ it follows that the system was able to take the play out of the domain of $f_1^{\neg\varphi}$. In order to accomplish this a state s_a must be reached in $\pi = s_0 l_0 \ldots s_a l_a \ldots$ where the environment can no longer play according to $f_1^{\neg\varphi}$. 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 legal environment $\Delta_P(s_a) \cap \mathcal{C} = \Delta_M(s_a) \cap \mathcal{C}$, 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 $s_a \in S_M \setminus S_P$ there should exist s_b the last state before s_a ($\pi = s_0 l_0 \ldots s_b l_b \ldots s_a l_a \ldots$) where the play stepped out of S_P. If s_b is monitored it would keep playing according to $f_1^{\neg\varphi}$, so we can assume that s_b is controllable, but if this is the case, again, since all the controllable options are preserved in P and the winning strategy holds, by definition, against every system choice, the

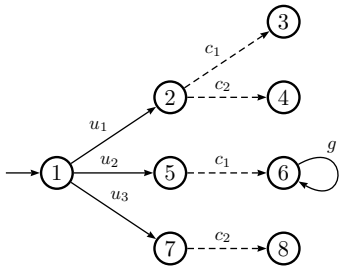

Fig. 5. Several conflicts example (E).

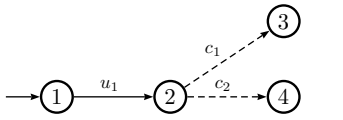

Fig. 6. Several conflicts example minimization(E_1).

environment should be able to keep playing accordingly, not stepping out of P. □

In order to find a minimal representation of the non realizability problem in terms of $\mathcal{I}$, we expect $min(\mathcal{I})$ to incrementally reduce a plant while checking if a local minimum has been reached. We define the following relation over LTSs:

Definition IV.5. *(Maximal Non Realizability Legal 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 non realizability legal plant from M if P is an non realizability legal plant for M and the following holds: $\nexists Q$ non realizability legal plant for M, $P \neq Q$ s.t. P is non realizability legal plant for Q.

The closest non realizability legal plant relation defines a semi-lattice where the original plant E lies at the top and the minimal (non-empty) non realizability legal non realizability legal plants appear as the bottom leaves. We will use this notion to explore potential minimizations starting from a given plant and moving towards its closest non realizability legal 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 $\mathcal{C} = \{c_1, c_2, g\}$, the original plant E will stay at the top of the semi lattice 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 plants non realizability legal by removing just one monitored transition. The automaton non realizability legal by removing $\{u_1, u_3\}$ is not shown since it is non realizability legal but does not preserve non realizability. We can now introduce the notion of a minimal non realizability legal plant.

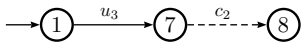

Fig. 7. Several conflicts example minimization(E_2).

Definition IV.6. *(Minimal Counter-Strategy Preserving Sub-LTS)* Given a control problem $\mathcal{I}_\kappa =< E_\kappa, \mathcal{C}, \varphi >$ where $E_\kappa = (S_{E_\kappa}, \Sigma, \Delta_{E_\kappa}, s_0)$ we say that E_κ is a minimal non realizability legal plant if I_κ is not realizable then the following holds:

$$\forall E'_\kappa \text{s.t. } E'_\kappa \text{ is a closest non realizability legal plant from } E_\kappa$$
$$\Longrightarrow$$
$$\mathcal{I}'_\kappa =< E'_\kappa, \mathcal{C}, \varphi > \text{ is realizable}$$

This definition (IV.6) encapsulates what we expect from $min(\mathcal{I})$, i.e., if $min(\mathcal{I}) = \mathcal{I}_\kappa$, $\mathcal{I}_\kappa =< E_\kappa, \mathcal{C}, \varphi >$ then E_κ should be a minimal non realizability legal plant from E.

NO FALTA UNA NOCION DE COMPLETITUD. ES DECIR, SI UN PROBLEMA ES NO REALIZABLE, EXISTE UN SUBLTS QUE PRESERVA NO REALIZABILIDAD?

V. A Behavior Minimization Procedure

In order to find an implementation for $min(\mathcal{I})$ we need to construct an algorithm that will yield at least one minimal E_κ Alternating Sub-LTS for E and $\mathcal{C}$. To automatize the procedure that looks for the set of minimal Alternating Sub-LTS for E we propose a procedure that inspects the elements of the semi lattice of Alternating Sub-LTSs and then checks them for (non) realizability.

The following implementation eagerly looks for a single conflict in the search space conformed by the aforementioned semi-lattice. The code can be succinctly 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 back track will be fired every time a realizable representation is reached. A minimal Alternating Sub-LTS will be found when no further reduction is available.

```
1  dfs_min(E):
2        E_new = E_old = E
3        T_u = E.get_monitored_transitions() (A)
4        to_remove = [] (B)
5        while(|T_u| > 0):
6              do:
7                    if (|T_u| == 0):
8                          return E_old
9                    t = T_u.remove(0) (C)
10             while(¬E_old.contains(t) or d_out(E_old,t) == 1)
11             to_remove.add(t)
12             E_new = E.get_legal_reduction(to_remove) (D)
13
14             if(E_new == ∅):
15                   realizable = ⊤ (E)
16             else:
17                   realizable = E_new.check_realizability() (F)
18             if (¬ realizable):
19                   T_u = T_u ∩ E_new.get_monitored_transitions()
20                   E_old = E_new
21             else:
22                   to_remove.remove(t) (G)
23       return E_old (H)
```

Fig. 8. Eager single-conflict exploration

In order to ensure that this algorithm properly finds a minimal Alternating Sub-LTS we present the following property:

Lemma V.1. *(*Alternating Sub-LTSs preserve realizability*) Let E be the plant for the non realizable control problem $\mathcal{I} =< E, \mathcal{C}, \varphi >$, E_1, E_2 be Alternating Sub-LTSs from E and E_2 Alternating Sub-LTS for E_1 then:*

$$E_1 \textit{ is realizable} \rightarrow E_2 \textit{ is realizable}$$

Proof. If the system has a winning strategy f_0^{φ} in E_1 but no winning strategy in E_2 would imply that the environment is able to either take the play into a deadlock state or a σ-trap that falsifies the property φ. Suppose that the system is able to play according to f_0^{φ} up to state s_i in E_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-LTSs, 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-LTS definition inclusion is satisfied. Is proven by contradiction that realizability is preserved under plant induction. □

This lemma was introduced to show that if we keep removing elements from a set $\mathcal{T}_u$ of candidate monitored transitions from E, reducing it to its closest legal representation until we can not, while preserving non-realizability, remove anything else we reach an Alternating Sub-LTS that is also minimal, since no Alternating Sub-LTS from it exist that preserves non realizability, implying that all the closest Alternating Sub-LTSs in its neighborhood are realizable proves that the current plant is minimal. Having observed this we can define an 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 later 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 by the Alternating Sub-LTS reduction. This reduction, at (D), looks for the closest Alternating Sub-LTS obtained after removing transitions in `to_remove` from E. If the reduction rendered the plant vacuous a backtrack step is triggered by fixing the realizability variable as true (E), otherwise the oracle is asked for realizability of the plant under evaluation (E_{new}) (F). If the check indicates that the plant is non realizable the candidate set and the last non realizable legal plant E_{old} are updated (since further minimization is possible), otherwise a backtrack is triggered by removing t the transition under test from the effective set `to_remove` (G). 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_{old} is returned as a minimal Alternating Sub-LTS for E and $\mathcal{C}$ (H).
The set $\mathcal{T}_u$ is monotonically reduced, one can argue that in the step (G) the transition t was removed from the candidate set but was retained in E_{old}. What happens if it should have been removed later on by the algorithm? In such case removing t from E_{old} 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_{old} and getting the legal representation $E_{1,\ldots,n}$ 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 legal representation we would get a new plant E'_{new} that can be Alternating Sub-LTS for E_{new}, and from lemma V.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 sacrificing minimality.

VI. Validation

The evaluation of our technique was guided by two main concerns. We wanted to know if our approach was actually minimizing the initial plant, and if the minimization was offering valuable insight on the non realizability cause.
When faced with the task of validating our work beyond the scope of synthetic examples we stumbled upon two obstacles. The first was that unrealizable specifications are seldom to come by in the existing literature and the second, that most published cases were presented under the declarative approach (as a set of LTL formulas), rendering them incompatible with our formal framework. We wanted to use our own non realizable specifications only for particular cases and not as the whole validation space. We discarded the idea of using non realizable specifications written by students, as in [27] since we do not think that they are necessarily relevant (for a discussion on this topic refer to [28]). Considering the previously stated we decided to explore the existing work written in a declarative fashion as part of our validation, translating it into our formalism and then trying out the technique over these translated examples.
All case studies were run using an extension of the MTSA tool [29]. MTSA natively supports specificaiton of LTS [30] and properties using a textual, process algebraic notation and FLTL [21]. The tool supports synthesis of controllers for SGR(1) control problems. Prior to this work the only feedback given to the user when specifying an unrealizable problem was a legend reading that *there was no controller for the given specification*. After running the minimization the tool presents the conflict as a new LTS in both its enumerative and graphical representation.
We present the analysis of applying our minimization to the Mine Pump Controller [31], Modified Lift Controller [10], t-strong-fairness example [32] and the CTvPlatform Controller [33] case studies as a way to validate our minimization with published work. We took two case studies from signal oriented specifications to show that our technique is also

useful in complementing existing minimization tools, such as the formulae based minimization from [34].

The other two case studies described here come from behavior based specifications and show the incremental analysis of the problem in the first example, and the existence of relatively simple problems when highly modular specifications are involved.

We tried our technique over 30 unrealizable specifications obtaining the quantitative results partially shown in table I. The experiments 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 operative 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, $pct(S_e)$ 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, $pct(\Delta_e)$ is the reduction percentage of the later two. $Time/S_e$ is the relative measure of time taken per state. *steps* reports how many iterations the algorithm needed before finding the first conflict. $pct(explored)$ reports the percentage of how many edges the algorithm needed to explore before finding the first conflict. The Jukebox example was taken from [35] and the Biscotti example was adapted from [32].

A. Mine Pump Controller (MPC)

We wrote a version of the Mine Pump Controller (MPC) originally presented in [36]. The original problem specified 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 wrote a version of the problem in `Ratsy`([37]) and used our automated translation tool to get the equivalent control problem specification. We added a liveness system requirement asking for water level to be kept within reasonable limits infinitely often ($\square\diamond\neg water.Hi$). In [31] the authors extract two potential safety goals out the original example, these are: $\square(methane.hi \implies \bigcirc(\neg running))$ and $\square(water.hi \implies \bigcirc(running))$. Instead of this, we transformed the MPC example into a GR(1) problem, since it is reasonable that an acceptable level of water should be achieved infinitely ofter for the mine to operate. We applied our minimization on the translated specification to get the plant depicted in Figure 9. We can see here that the minimization properly slices the plant to expose what in [31] is characterized as the boundary condition ($\diamond hm \wedge m$) and in our case is preventing the system from achieving its goal.

This slice should lead the engineer towards adding the missing assumption $\square\diamond(\neg methane.hi \wedge \neg water.hi)$. If the original problem and the implementation proposed in the appendix from [36] are expected to be deployed, this assumption should

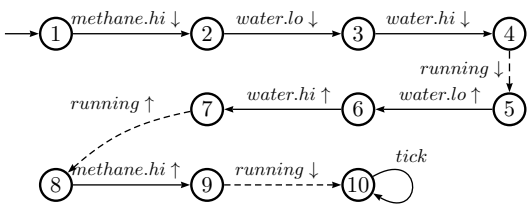

Fig. 9. Mine Pump Controller (MPC) minimization.

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 no 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 oportunity to satisfy its goal.

This example shows the relevance of behavior base minimization for unrealizable specifications initially described as a set of LTL formulae over boolean variables. This is an explicit representation of the unrealizability cause that gives immediate feedback about the interaction of the system and the environment that leads to the violation of the liveness guarantee.

B. Modified Lift Controller

Another example that has been evaluated is that of the Modified Lift Controller from [10] first presented in its realizable version in [20]. 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 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 specification 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

Case	Time	$\|S_e\|$	$\|S_{e\kappa}\|$	$pct(S_e)$	$\|\Delta_e\|$	$\|\Delta_{e\kappa}\|$	$pct(\Delta_e)$	$Time/S_e$	steps	$pct(explored)$
Biscotti	15.80	8	4	50.00%	12	4	33.33%	1.98	2	16.67%
flood no g	11.75	4	3	75.00%	7	4	57.14%	2.94	4	57.14%
more simple	9.15	4	4	100.00%	7	6	85.71%	2.29	4	57.14%
flood	13	5	3	60.00%	8	4	50.00%	2.60	4	50.00%
mine sweeper	12.45	5	3	60.00%	8	4	50.00%	2.49	5	62.50%
several conflicts vic	14.47	7	3	42.86%	9	3	33.33%	2.07	3	33.33%
t strong fairness	10.65	7	4	57.14%	9	4	44.44%	1.52	3	33.33%
t strong fairness b	15	7	5	71.43%	9	6	66.67%	2.14	5	55.56%
memory counter	13	5	3	60.00%	9	4	44.44%	2.60	2	22.22%
motivating example	15	7	6	85.71%	10	6	60.00%	2.14	6	60.00%
mine pump	8.7	14	5	35.71%	20	5	25.00%	0.62	6	30.00%
lift controller	121	95	43	45.26%	130	50	38.46%	1.27	35	26.92%
counter tea	194.2	87	37	42.53%	182	49	26.92%	2.23	52	28.57%
Tv controller	11031	218	8	3.67%	1416	21	1.48%	50.60	43	3.04%
fec	90355.8	9472	17	0.18%	14590	19	0.13%	9.54	346	2.37%
tictactoe3	66054	4658	176	3.78%	18561	354	1.91%	14.18	414	2.23%
tictactoe4	70896	4696	236	5.03%	18647	472	2.53%	15.10	529	2.84%
tictactoe5	69738	4696	236	5.03%	18647	472	2.53%	14.85	529	2.84%
tictactoe6	75279	4696	236	5.03%	18647	472	2.53%	16.03	529	2.84%
tictactoe2	65449	4590	184	4.01%	19241	420	2.18%	14.26	418	2.17%
gen buf	6688142	10964	2235	20.38%	14894	2963	19.89%	610.01	1093	7.34%

TABLE I
QUANTITATIVE RESULTS FOR MINIMIZED PLANTS

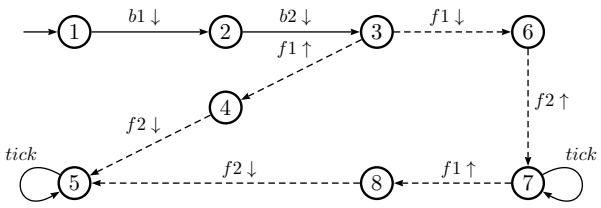

Fig. 10. Modified Lift Controller minimization.

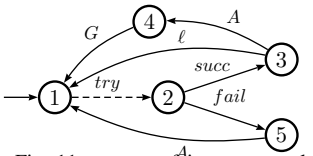

Fig. 11. t-strong fairness example.

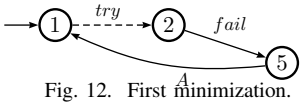

Fig. 12. First minimization.

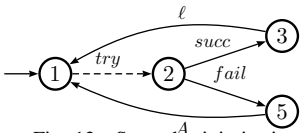

Fig. 13. Second minimization.

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 [10] 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(. . .)*. Such a counter strategy is properly captured by the minimal induced plant since the strategy for the environment can be expressed in this case as $f_1^{\neg\varphi}(s_1) = b1\downarrow$, $f_1^{\neg\varphi}(s_2) = b2\downarrow$.

C. t-strong-fairness

In [32] a concise example was presented to show that the notion of t-strong-fairness was not enough to achieve synthesis of controller over fallible domains. The plant is depicted in figure 11, where the only controllable action is try. The system should achieve to produce event G infinitely often provided that the environment agrees to produce A infinitely often. The idea in [32] is to treat a failure as something not systematic, different from a truly antagonistic event. The example shows how this could not be captured with weaker notions of fairness, we take the plant in two different incarnations. In the first we keep the liveness properties intact i.e., the system should accomplish $\square\diamond G$ if the environment agrees to comply with $\square\diamond A$. After applying our technique we get the minimized plant shown in figure 12. The minimization clarifies the cause of non realizability since it is clear that the environment can keep the play within the language described as $(try, fail, A)^*$.

Now, we can add a new environmental liveness assumption, forcing the environment to produce infinitely many $succ$ events. After applying our technique we get the result in figure 13. This minimization is better explained in conjunction with the strategy shown in figure 14. We can see that state 4 has been removed and state 2 preserves both the $succ$ and $fail$ options since the environment will take each one alternately to comply with the $\square\diamond A$ assumption while avoiding state 4, since this will allow infinitely many Gs. This example is very synthetic but depicts a scenario where behavior based diagnosis proves itself a useful tool not only for repairing a specification but to reason incrementally about requirements'

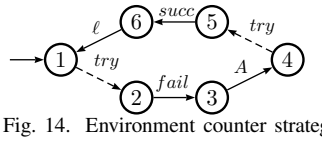

Fig. 14. Environment counter strategy.

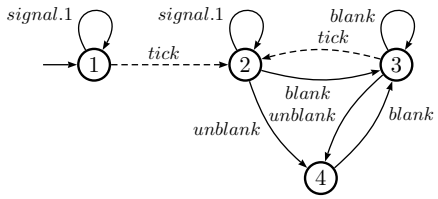

Fig. 15. CTvPlatform TV Controller minimization.

limitations.

D. CTvPlatform Controller

In [33] the authors introduce a modular specification for a TV System using their Koala model. At the time of this writing another member of our group was working on a specification inspired in this particular system and introducing some abstractions, such as the user module, from [38].
As in the original scenario the specification was decoupled in several modules to achieve cohesion. Relevant to this case study is the TV set that can display random noise while tuning, set the output to the specified channel afterwards or eventually reset the channel selection and go blank. The other modules describe the behavior of the user (selecting a particular channel), the tuning sub-system (linking user selection with frequency filtering) and the screen driver (that outputs the corresponding image, be it a channel, random noise or plain nothing when going blank). The goal was to synthesize a controller for a TV that should satisfy the following liveness requirement: $\square\diamond(select.i \implies display.i)$. Compositionality trades cohesion for behavioral abstraction. This means that even when the behavior of each component is fairly simple their concurrent interaction is difficult to grasp. Applying our minimization technique to several ongoing specifications within our group we found that even though the plant exposed a complex potential behavior (in terms of its state space) unrealizability could be explained without going too far away from the starting point.
The CTvPlatform TV Controller diagnosis, as depicted in figure 15 showed that the tuner introduced a non controllable self loop where the environment could flood the execution with incoming tuning signals ($signal.i$) and that even if a controllable action was allowed to escape the looping state the screen was allowed to produce the $blank$ event indefinitely.

There is a restriction that we want to make explicit in the translation from the declarative specifications into the operative ones. Since the translation is constructing an explicit representation of the declarative formulas by extracting the space of feasible valuations for the variables in $\mathcal{V} = \mathcal{X} \cap \mathcal{Y}$ and serializing the difference between values of elements belonging to the transition relation induced by ρ, we face a state explosion problem no necessarily present in the symbolic setting. This restriction forced ourselves to leave some interesting problems out of the validation. We expect to solve this situation in our future work.

VII. 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, which is usually the case, 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 [39] for CTL* specifications, in [40] for the GR(1) [24] subset, further explored in [27] for the declarative version and in [41] for the generative one. The problem of completeness (where the idea is to find states of formulas irrelevant to the realization of the goals) has been developed in [42] and [43], amongst others. There are to different although related problems when in 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 reasoned 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 [19]), 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 [44] an hypothetical testing scenario is introduced where an air plane is progressively stripped down to its minimum working core while preserving the fault that caused a previous crash. Although somehow laugh-provoking the concept behind this is that reducing, or slicing, the subject under study while preserving the undesired behavior is desirable, since it allows the engineer to zoom into her problem while removing every irrelevant feature from it. In our case the airplane will be the original `LTS` and the passenger seats, coffee machine and dvd player being removed to reach a minimum fault-preserving core will be the environment transitions that are irrelevant to the falsification of φ.
In what we consider to be one the works more closely related to our technique [8] the authors present a debugging framework for non realizable specifications that provides several diagnosis mechanism, 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 [9]). 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 [8]. 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.

[9] 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 [8]. The authors take a syntactical approach when simplifying a formula by over or under approximation of its components, clearly different that our semantical approach. The idea of [45] is to look for the closest satisfiable specification w.r.t. 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 [46] 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 [31] 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 [47] 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. This two approaches are different to ours since they do not, nor intend to, treat the non realizability problem.

[10], [48] and [49] 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. [10] 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 [48], the authors propose mining assumptions out of an unrealizable GR(1) specification through the use of a counter strategy from [34]. Again, a template based approach is used. In [49] an assumption computing technique is presented based in Craig's interpolants is presented. It obtains refinements by negating plays of the counter-strategy.

In [50] the authors offer a flexible framework for quantifying how well an implementation comes to satisfy initially incompatible requirements. This is achieved by defining a solution distance metric parametrized by an error model.

[51] introduces a report-based debugging technique that computes statistics, positional information for the counter strategy, detecting superfluous assumptions and analyzing error-resilience against violations of the environment assumptions.

There has been some interesting applications of non realizability diagnosis over different domains, in [52] the authors use the notion of sanity checking of requirements for avionics systems. This sanity checking is composed by consistency checking, redundancy checking and completeness of the requirements. Is worth noting that this is a model-free approach. [53] applies the debugging of formal requirements for `MITL` and `STL` specifications. In [54] imperative programs are processed to get diagnostic data, analyze non realizability causes and then implement template-based corrections. In [50] an interactive play-out for scenario-based specifications is presented. If the specifications is unrealizable an interaction game is presented to the user in order to expose the cause of non realizability. It is built following [34] counter-strategy. In [55] 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 interaction game is provided to explain the cause of non realizability.

Several works have used mobile robotics examples as motivation but non realizability diagnosis has been applied to this domain is the case of [56], and [57].

In [58] an approach to compute differences between LTSs is introduced in the context of evolving behaviors. This is somewhat related with our work in the sense that it copes with the potential divergence between the current model and the initial user intent.

VIII. 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 the engineer. The problem was introduced in the context of operational specifications where the guarantees and assumptions are provided as LTL formulas and 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 and it should serve as a kick start to the development of other diagnosis techniques for this kind of specifications. It should also promote the publication of non realizable operational specifications from industrial domains. Future work can specialize our approach by defining domain specific exploration of the search space. If the problem falls into a particular subset, e.g. GR(1). One should be able to use this to improve the search for a minimal non realizability preserving representative. Beyond this, we thinks that irrelevant automata can also be marked out of the diagnosis to improve understanding and in order to expand applicability the ground formalism can be slightly adapted to cope better with signal-based specifications.

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

REFERENCES

[1] M. Jackson, "The world and the machine," in *Proceedings of the 17th international conference on Software engineering*, ser. ICSE '95. New York, NY, USA: ACM, 1995, pp. 283–292. [Online]. Available: http://doi.acm.org/10.1145/225014.225041

[2] E. Letier and A. van Lamsweerde, "Agent-based tactics for goal-oriented requirements elaboration," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 83–93. [Online]. Available: http://doi.acm.org/10.1145/581339.581353

[3] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, pp. 978–1005, October 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=357525.357521

[4] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

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

[6] S. Maoz, J. O. Ringert, and B. Rumpe, "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*, ser. LNI, W. Hasselbring and N. C. Ehmke, Eds., vol. 227. GI, 2014, pp. 63–64. [Online]. Available: http://eprints.uni-kiel.de/23752/

[7] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2011.08.007

[8] R. Könighofer, G. Hofferek, and R. Bloem, "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, 2009, pp. 152–159. [Online]. Available: http://dx.doi.org/10.1109/FMCAD.2009.5351127

[9] V. Schuppan, "Towards a notion of unsatisfiable and unrealizable cores for LTL," *Sci. Comput. Program.*, vol. 77, no. 7-8, pp. 908–939, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2010.11.004

[10] R. Alur, S. Moarref, and U. Topcu, "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, 2013, pp. 26–33. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6679387

[11] E. Torlak, F. S.-H. Chang, and D. Jackson, "Finding minimal unsatisfiable cores of declarative specifications," in *Proceedings of the 15th International Symposium on Formal Methods*, ser. FM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 326–341.

[12] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, pp. 371–384, July 1976. [Online]. Available: http://doi.acm.org/10.1145/360248.360251

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

[14] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987. [Online]. Available: http://dx.doi.org/10.1016/0167-6423(87)90035-9

[15] S. Uchitel, J. Kramer, and J. Magee, "Synthesis of behavioral models from scenarios," *IEEE Trans. Software Eng.*, vol. 29, no. 2, pp. 99–115, 2003. [Online]. Available: http://dx.doi.org/10.1109/TSE.2003.1178048

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

[17] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 26, no. 1, pp. 100–106, Jan. 1983. [Online]. Available: http://doi.acm.org/10.1145/357980.358021

[18] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "Synthesizing nonanomalous event-based controllers for liveness goals," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 9:1–9:36, 2013. [Online]. Available: http://doi.acm.org/10.1145/2430536.2430543

[19] A. Pnueli and R. Rosner, "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, 1989, pp. 179–190. [Online]. Available: http://doi.acm.org/10.1145/75277.75293

[20] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2011.08.007

[21] D. Giannakopoulou and J. Magee, "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*, J. Paakki and P. Inverardi, Eds. ACM, 2003, pp. 257–266. [Online]. Available: http://doi.acm.org/10.1145/940071.940106

[22] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978. [Online]. Available: http://doi.acm.org/10.1145/359576.359585

[23] L. de Alfaro and T. A. Henzinger, "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. M. Tjoa and V. Gruhn, Eds. ACM, 2001, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/503209.503226

[24] N. Piterman, A. Pnueli, and Y. Sa'ar, "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*, ser. Lecture Notes in Computer Science, E. A. Emerson and K. S. Namjoshi, Eds., vol. 3855. Springer, 2006, pp. 364–380. [Online]. Available: http://dx.doi.org/10.1007/11609773_24

[25] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "Synthesis of live behaviour models," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, G. Roman and K. J. Sullivan, Eds. ACM, 2010, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/1882291.1882305

[26] A. Church, "Logic, arithmetic and automata," in *Proceedings of the international congress of mathematicians*, 1962, pp. 23–35.

[27] S. Maoz and J. O. Ringert, "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*, T. Zimmermann, J. Cleland-Huang, and Z. Su, Eds. ACM, 2016, pp. 362–372. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950300

[28] D. G. Feitelson, "Using students as experimental subjects in software engineering research–a review and discussion of the evidence," *arXiv preprint arXiv:1512.08409*, 2015.

[29] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel, "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, 2008, pp. 475–476. [Online]. Available: http://dx.doi.org/10.1109/ASE.2008.78

[30] R. M. Keller, "Formal verification of parallel programs," *Commun. ACM*, vol. 19, no. 7, pp. 371–384, 1976. [Online]. Available: http://doi.acm.org/10.1145/360248.360251

[31] R. Degiovanni, N. Ricci, D. Alrajeh, P. F. Castro, and N. Aguirre, "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*, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 507–518. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970349

[32] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "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*, R. N. Taylor, H. C. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 211–220. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985823

[33] R. C. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The koala component model for consumer electronics software," *IEEE Computer*, vol. 33, no. 3, pp. 78–85, 2000. [Online]. Available: http://dx.doi.org/10.1109/2.825699

[34] R. Könighofer, G. Hofferek, and R. Bloem, "Debugging unrealizable specifications with model-based diagnosis," in *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Barner, I. G. Harris, D. Kroening, and O. Raz, Eds., vol. 6504. Springer, 2010, pp. 29–45. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19583-_98

[35] L. Ryzhyk and A. Walker, "Developing a practical reactive synthesis tool: Experience and lessons learned," in *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016.*, ser. EPTCS, R. Piskac and R. Dimitrova, Eds., vol. 229, 2016, pp. 84–99. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.229.8

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

[37] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, and R. Seeber, "RATSY - A new requirements analysis tool with synthesis," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. B. Jackson, Eds., vol. 6174. Springer, 2010, pp. 425–429. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_37

[38] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "System architecture: the context for scenario-based model synthesis," in *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004, Newport Beach, CA, USA, October 31 - November 6, 2004*, R. N. Taylor and M. B. Dwyer, Eds. ACM, 2004, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/1029894.1029903

[39] O. Kupferman and M. Y. Vardi, "Vacuity detection in temporal model checking," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 4, no. 2, pp. 224–233, 2003.

[40] U. Klein and A. Pnueli, "Revisiting synthesis of GR(1) specifications," in *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Barner, I. G. Harris, D. Kroening, and O. Raz, Eds., vol. 6504. Springer, 2010, pp. 161–181. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19583-9_16

[41] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "Synthesizing nonanomalous event-based controllers for liveness goals," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 9:1–9:36, 2013. [Online]. Available: http://doi.acm.org/10.1145/2430536.2430543

[42] H. Chockler, O. Kupferman, R. P. Kurshan, and M. Y. Vardi, "A practical approach to coverage in model checking," in *International Conference on Computer Aided Verification*. Springer, 2001, pp. 66–78.

[43] H. Chockler, O. Kupferman, and M. Y. Vardi, "Coverage metrics for temporal logic model checking," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2001, pp. 528–542.

[44] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Trans. Software Eng.*, vol. 28, no. 2, pp. 183–200, 2002. [Online]. Available: http://dx.doi.org/10.1109/32.988498

[45] K. Kim, G. E. Fainekos, and S. Sankaranarayanan, "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, 2012, pp. 5171–5176. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2012.6224826

[46] K. Kim and G. E. Fainekos, "Minimal specification revision for weighted transition systems," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. IEEE, 2013, pp. 4068–4074. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2013.6631151

[47] S. Hagihara, N. Egawa, M. Shimakawa, and N. Yonezaki, "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*, I. Crnkovic, M. Chechik, and P. Grünbacher, Eds. ACM, 2014, pp. 629–634. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642968

[48] W. Li, L. Dworkin, and S. A. Seshia, "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*, S. Singh, B. Jobstmann, M. Kishinevsky, and J. Brandt, Eds. IEEE, 2011, pp. 43–50. [Online]. Available: http://dx.doi.org/10.1109/MEMCOD.2011.5970509

[49] D. G. Cavezza and D. Alrajeh, "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*, ser. Lecture Notes in Computer Science, A. Legay and T. Margaria, Eds., vol. 10205, 2017, pp. 281–297. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-54577-5_16

[50] P. Cerný, S. Gopi, T. A. Henzinger, A. Radhakrishna, and N. Totla, "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*, A. Jerraya, L. P. Carloni, F. Maraninchi, and J. Regehr, Eds. ACM, 2012, pp. 53–62. [Online]. Available: http://doi.acm.org/10.1145/2380356.2380371

[51] R. Ehlers and V. Raman, "Low-effort specification debugging and analysis," in *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014.*, ser. EPTCS, K. Chatterjee, R. Ehlers, and S. Jha, Eds., vol. 157, 2014, pp. 117–133. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.157.12

[52] J. Barnat, P. Bauch, N. Benes, L. Brim, J. Beran, and T. Kratochvila, "Analysing sanity of requirements for avionics systems," *Formal Asp. Comput.*, vol. 28, no. 1, pp. 45–63, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00165-015-0348-9

[53] A. Dokhanchi, B. Hoxha, and G. E. Fainekos, "Formal requirement elicitation and debugging for testing and verification of cyber-physical systems," *CoRR*, vol. abs/1607.02549, 2016. [Online]. Available: http://arxiv.org/abs/1607.02549

[54] R. Könighofer and R. Bloem, "Automated error localization and correction for imperative programs," in *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, P. Bjesse and A. Slobodová, Eds. FMCAD Inc., 2011, pp. 91–100. [Online]. Available: http://dl.acm.org/citation.cfm?id=2157671

[55] S. Maoz and Y. Sa'ar, "Two-way traceability and conflict debugging for aspectltl programs," *Trans. Aspect-Oriented Software Development*, vol. 10, pp. 39–72, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36964-3_2

[56] V. Raman and H. Kress-Gazit, "Automated feedback for unachievable high-level robot behaviors," in *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 2012, pp. 5156–5162. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2012.6224807

[57] ——, "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*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 663–668. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22110-1_54

[58] Z. Xing, J. Sun, Y. Liu, and J. S. Dong, "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*, ser. Lecture Notes in Computer Science, S. Qin and Z. Qiu,

Eds., vol. 6991. Springer, 2011, pp. 537–552. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24559-6_36

[59] S. Barner, I. G. Harris, D. Kroening, and O. Raz, Eds., *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 6504. Springer, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19583-9

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