Unavoidable Boundary Conditions: A Control Perspective on Goal Conflicts

Francisco Cirelli
Universidad de Buenos Aires, Argentina
franciscocirelli@gmail.com

Dalal Alrajeh
Imperial College, London, UK
dalal.alrajeh@imperial.ac.uk

Sebastian Uchitel
CONICET/U. de Buenos Aires, Argentina
Imperial College, London, UK
sebastian.uchitel@gmail.com

***Abstract*—Boundary conditions express situations under which requirements specifications conflict. They are used within a broader conflict management process to produce less idealized specifications. Several approaches have been proposed to identify boundary conditions automatically. Some introduce a prioritization criteria to reduce the number of boundary conditions presented to an engineer. However, identifying the few, relevant boundary conditions remains an open challenge. In this paper, we argue that one of the problems of the state of the art is with the definition of boundary condition itself—it is too weak. We propose a stronger definition which we refer to as *Unavoidable Boundary Conditions* (UBCs), which utilizes the notion of realizability in reactive synthesis. We show experimentally that UBCs non-trivially reduce the number of conditions produced by existing boundary condition identification techniques. We also relate UBCs to existing concepts in reactive synthesis used to provide feedback for unrealizable specifications (including counter-strategies and unrealizable cores). We then show that UBCs provide a targeted form of feedback for repairing unrealizable specifications.**

***Index Terms*—Boundary Conditions, Reactive Synthesis, Realizability, Requirement Engineering**

I. Introduction

Requirements engineering (RE) is concerned with the elicitation of system-to-be goals and their refinement into specifications for the agents (humans, software systems, etc.) such that if all agents implement their specifications correctly, the system-to-be goals will also be achieved [41].

Requirements engineers must deal with conflicting requirements that result from multiple stakeholders [20], [36]. In addition, conflicting requirements can arise because of specification errors or due to unforeseen consequences of the emergent behavior prescribed by various composed requirements.

Conflicts should, eventually, be resolved. Identifying conflicts provides stakeholders and engineers with feedback that drives requirements negotiation and evolution towards error free and less idealized specifications [41].

Logical inconsistency is one manifestation of conflicting requirements. However, inconsistency does not capture more subtle forms of conflicts. One concept that has been studied extensively is that of boundary conditions [40]. These are conditions under which the goals of the system cannot be achieved. A requirements specification may be consistent yet have boundary conditions. Should the boundary condition occur, then logical inconsistency arises.

Methods for generating boundary condition from formal specifications have been proposed including manual pattern-based methods [40], evolutionary algorithms [15] and tableaux-based satisfiability procedures [16].

One of the limitations of the automated techniques is that they generate many boundary conditions, more than what may be reasonable for engineers to consider. For this reason, boundary condition generation approaches discuss various forms of reducing or prioritizing generated boundary conditions (e.g., [25], [43]).

In this paper, we argue that the problem is not in the boundary condition generation and prioritization algorithms themselves, but in the definition of boundary condition itself. We argue that the definition of boundary condition is too weak and provide a stronger definition, *Unavoidable Boundary Conditions* (UBCs), by making use of the notion of *realizability* [3] taken from Reactive Synthesis [6]. For a condition to be a UBC to a set of goals, we not only require it to be a boundary condition but also that the agent that is responsible for achieving the goal cannot prevent the condition from occurring. That is, the agent cannot *control* its environment to prevent the boundary condition from occurring.

Presenting engineers with boundary conditions in realizable specifications presents little value. In these cases, the boundary condition is simply another constraint that must be taken into account when implementing the agent. If the specification is realizable then the boundary condition is avoidable by the software system-to-be. On the other hand, by focusing on unavoidable boundary conditions in unrealizable specifications, we reduce the number of conditions that engineers must consider to resolve conflicts in their specifications.

In this paper, we formalize the notion of UBC and show experimentally that UBCs reduce non-trivially the number of conditions that existing boundary condition generation techniques produce.

By enriching boundary conditions with a control perspective, we bring UBCs closer to the world of Reactive Synthesis, where conflict and boundary conditions are not discussed. Indeed, in Reactive Synthesis focus is on understanding causes (e.g., [12], [13]) and repairs (e.g., [9]) for unrealizable specifications rather than conditions leading to unsatisfiability.

In this paper, we show that UBCs complement existing concepts and approaches to providing unrealizability feedback and repair. In particular, we show that the existence of UBCs

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00031

and unrealizability are equivalent notions and that counter-strategies to a UBC (i.e., a witness of how the agent cannot avoid the boundary condition) are counter-strategies to the specification itself (i.e., a witness of how the specification cannot be realized by the software-to-be). We show that UBCs can provide alternative, finer grained, explanations than unrealizable cores (minimal subsets of goals that are unrealizable). Finally, we show how UBCs relate to well-separation [21] and how they can be used in realizability repair.

In summary: (i) *we propose unavoidable boundary conditions (UBCs) that brings together concepts from conflict analysis in Requirements Engineering and realizability from Reactive Synthesis,* (ii) *we argue the relevance of UBCs for conflict detection and show experimentally that they can reduce significantly the number of conditions that boundary condition generation methods present to engineers,* (iii) *we relate UBCs with specification unrealizability, counter-strategies and unrealizable cores, and finally* (iv) *we show how UBCs relate to specification realizability repair.*

The rest of the paper is as follows: We start by providing, in Section II the background on boundary conditions, realizability, specifications and control problems. These are necessary concepts to provide a motivating example in Section III, and a formal definition of unavoidable boundary conditions in Section IV. In Section IV-A we study the extent to which UBCs can reduce the boundary conditions generated by existing methods, and in Section V we discuss how UBCs relate to unrealizability, counter-strategies and well-separation. Finally, we present a discussion on related work and conclusions.

II. Background

In this sections, we introduce boundary conditions, linear-time temporal logic, realizability and unify terminology.

A. Goals, domain assumptions and boundary conditions

We use the original formulation of boundary conditions given by van Lamsweerde et al. [40]. A set of goals $\{G_1, \ldots, G_n\}$ is said to be *divergent* with respect to domain assumptions Dom if there exists a *boundary condition* ϕ such that the following conditions hold:

$$\{Dom, \phi, \bigwedge_{1 \leq i \leq n} G_i\} \models \bot \qquad (\textit{inconsistency})$$

$$\{Dom, \phi, \bigwedge_{j \neq i} G_j\} \not\models \bot, \text{ for } 1 \leq i \leq n \qquad (\textit{minimality})$$

$$\phi \neq \neg(G_1 \wedge \ldots \wedge G_n) \qquad (\textit{non-triviality})$$

For short, we will say that ϕ is a boundary conditions for $\{Dom, G\}$ where $G = G_1 \wedge \ldots \wedge G_n$. Intuitively, a boundary condition captures a particular situation that makes the goals inconsistent. The first condition establishes that goals $G_1, \ldots, G_n$ and Dom cannot be satisfied simultaneously in any situation where ϕ holds. The second condition states that removing any of the goals removes the logical inconsistency between the goals and domain assumptions. The third condition prohibits a boundary condition to be the negation of goals. Notice also that the minimality condition forbids trivial boundary conditions (ϕ cannot be equivalent to $\bot$ nor can it be the negation of one of the goals), and requires it to be consistent with the domain assumptions.

B. Linear-Time Temporal Logic

Linear-Time Temporal Logic (LTL) is a formalism extensively employed to express properties of systems [26]. It is widely used in RE literature to formalize requirements. LTL assumes a linear topology of time, i.e., each time instant is followed by a unique future time instant, and its formulas are evaluated over infinite sequence of states, sometimes referred to as traces, that represent system executions. Given a set $\mathcal{V}$ of propositional variables, LTL formulas are inductively defined using the standard logical connectives and temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until), as follows:

1) every $p \in \mathcal{V}$ is an LTL formula, and
2) if f_1 and f_2 are LTL formulas, then so are $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$, $\bigcirc f_1$ and $f_1 \mathcal{U} f_2$.

We consider the usual definition for the operators $\Diamond$(eventually), $\Box$ (always), and $\mathcal{W}$ (weak until) in terms of $\bigcirc$, $\mathcal{U}$ as follows: $\Diamond f_1 = \top \mathcal{U} f_1$, $\Box f_1 = \neg\Diamond\neg f_1$, $f_1 \mathcal{W} f_2 = (f_1 \mathcal{U} f_2) \vee \Box f_1$ and logical connectives $\rightarrow$ and $\wedge$ in terms of $\neg$ and $\vee$. Given a set of LTL formulas $\{\phi_i\}$, we will refer to the formula $\bigwedge\{\phi_i\}$ as their conjunctive formula.

LTL formulas are evaluated over infinite sequences of valuations of the variables in $\mathcal{V}$. Let $\mathcal{I} = \{I : \mathcal{V} \rightarrow \{\textit{true}, \textit{false}\}\}$ be the set of all possible valuations of variables in $\mathcal{V}$, and let $\mathcal{I}^\omega$ denote the set of infinite sequences of elements from $\mathcal{I}$. Given a trace $w \in \mathcal{I}^\omega$ and a position $0 \leq i$, formulas with no temporal operators are evaluated in the first positions, i.e., initial state. $\bigcirc f$ is true in w iff f is true in w at position 1, and $f_1 \mathcal{U} f_2$ is true in w iff there exists a position i such that f_2 holds in w at i, and for all $0 \leq j < i$, f_1 holds in w at j. We note that a formula f holds in w as $w \models f$.

C. Realizability

We use the formulation of realizability from [6]. Intuitively, given an LTL formula ϕ, realizability can be thought of as a turn-based game in which there are two players, the environment, who plays first, and the controller, who plays second. The set of propositional variables in ϕ are split (we assume full observability for simplicity) into those controlled by the controller $\mathcal{Y}$ and those by the environment $\mathcal{X}$. A play results in a sequence σ of valuations of propositional variables, where part of the valuation is defined by each player.

The objective of the controller is to set its variables to make ϕ hold (i.e., $\sigma \models \phi$), whereas the environment attempts to achieve the opposite. ϕ is *realizable* when there is a winning strategy for the controller, i.e, a way to win the game regardless of what the environment does. Similarly, ϕ will be *unrealizable* when the environment has a winning strategy.

More formally, let ϕ be an LTL formula and $\mathcal{Y}$ and $\mathcal{X}$, named the controllable and uncontrollable variables respectively, be finite, disjoint subsets of propositional variables such that the propositions of ϕ (noted $Prop(\phi)$) are a subset of $\mathcal{Y} \cup \mathcal{X}$.

A *strategy for the controller* is a function $\sigma_\mathcal{C} : S \times \mathcal{P}(\mathcal{X}) \rightarrow S \times \mathcal{P}(\mathcal{Y})$, where S is a memory element with a distinguished

element S_0, the *initial memory state*. A *run* r of a strategy σ is a sequence $(S_n, U_n)_{n\in\mathbb{N}} \subseteq (S \times \mathcal{P}(\mathcal{X}))^{\mathbb{N}}$ such that $S_1 = S_0$, i.e, it "starts" in the initial state and $\sigma(S_n, U_n) = (S_{n+1}, C_n)$ for all $n \in \mathbb{N}$. A strategy $\sigma_{\mathcal{C}}$ is called a *winning strategy for the controller* for ϕ (denoted as $\sigma_{\mathcal{C}} \models \phi$) if the *induced* LTL trace $w = \mathcal{I}_1\mathcal{I}_2...$ with $\mathcal{I}_1 = U_1 \cup C_1$, $\mathcal{I}_2 = U_2 \cup C_2$, etc satisfies ϕ, i.e., $w \models \phi$ for every run of the strategy. Analogously, strategies and winning strategies, $\sigma_{\mathcal{E}} : S \times \mathcal{P}(\mathcal{Y}) \to S \times \mathcal{P}(\mathcal{X})$, for the environment are defined, with the added condition that $\sigma_{\mathcal{C}}(S_i, -)_{\mathcal{P}(\mathcal{Y})}$ is constant for every state $S_i \in S$, as this accounts for the fact that the environment "plays first" and cannot react to what the controller has done within the same turn, only being able to change the future state depending on the controller's actions. Throughout this paper we will refer to a winning strategy for the environment as a *counter-strategy*. The counter-strategy can be played back by an engineer to better understand why the specification is unrealizable (e.g., [29], [37]).

D. Specifications and Unrealizable Cores

We unify some of the terminology given above as follows. Given a set of goals $G_1, \ldots, G_n$, and assumptions $A_1, \ldots, A_m$ expressed as LTL formulae, we refer to G and Dom as their conjunction. In other words, $G = G_1 \wedge \ldots \wedge G_n$, and $Dom = A_1 \wedge \ldots \wedge A_m$. We define a *control problem* or *specification* as a tuple $(Dom, G, \mathcal{X}, \mathcal{Y})$, where $\mathcal{X} \cup \mathcal{Y}$ is the set of propositions appearing in Dom and G, split into uncontrolled and controlled boolean variables. We say a specification $(Dom, G, \mathcal{X}, \mathcal{Y})$ is realizable (resp. unrealizable) if $Dom \to G$ is realizable (resp. unrealizable).

We say $C \subseteq \{G_1, G_2, ..., G_n\}$ is an *unrealizable core* of $(Dom, G, \mathcal{X}, \mathcal{Y})$ if $(Dom, C, \mathcal{X}, \mathcal{Y})$ is unrealizable and $(Dom, C', \mathcal{X}, \mathcal{Y})$ is realizable for every $C' \subset C$.

III. Motivating Example

In this section, we exemplify a shortcoming of the boundary condition definition (see Section II) by setting the scene for a conflict analysis of a well known case study, the Mine Pump [23]. We then hint at the solution, unavoidable boundary conditions, and explain their role in the context of an unrealizability analysis of the Mine Pump case study.

A. Conflict Analysis

Consider a system designed to control a pump in a mine. The goal is to use the pump to drain water from the mine and avoid flooding. The system controls a pump that can be on or off (modelled by p) and two sensors; one that senses high water (h) and another that senses methane (m). Whenever the water level is high, the pump should be turned on to lower it, and should methane be present, the pump should be turned off to prevent an explosion. This is specified in LTL as follows:

Domain Assumptions (Dom):

1) **Name:** *PumpWorks*
   Description: After two time steps of the pump being on, the water level is lowered.
   Formula: $\square((p \wedge \bigcirc(p)) \to \bigcirc(\bigcirc\neg h))$

Goals (G):

1) **Name:** *NoFlooding*
   Description: If the water level is high, the pump is turned on
   Formula: $\square(h \to \bigcirc(p))$
2) **Name:** *NoExplosion*
   Description: If there is methane present, the pump is turned off
   Formula: $\square(m \to \bigcirc(\neg p))$

Uncontrolled Variables ($\mathcal{Y}$): $\{h, m\}$

Controlled Variables ($\mathcal{X}$): $\{p\}$

The Mine Pump specification has a boundary condition: $\Diamond(h \wedge m)$ (i.e., some time in the future, h and m hold). If the boundary condition holds, then goals *NoFlooding* and *NoExplosion* cannot be satisfied as when h and m eventually hold, the goals require contradictory settings for p in the next time point. Goal *NoFlooding* requires p to be true, while *NoExplosion* requires p to be false.

Note that the specification is not inconsistent. There are traces that satisfy *PumpWorks* $\wedge$ *NoFlooding* $\wedge$ *NoExplosion*. As long as $h \wedge m$ never holds, the conjunction of goals and domain assumptions holds. Also note that $\Diamond(h \wedge m)$ is minimal and non-trivial as per the definition of boundary conditions.

Indeed, the boundary condition is informative of a problem with the specification despite the fact that specification is consistent. Moreover, it is informative because the mine pump controller does not control neither h nor m. So there are no guarantees that the boundary condition will not occur. Reaching a state where $h \wedge m$ holds is entirely up to the environment and should it happen, is is impossible for the controller to satisfy the specification.

In conclusion, the specification should be revised in the light of the identified boundary condition.

Suppose an air extractor (ext) is added to the mine pump system. The extractor is controlled by the system, and prevents methane from being present. We capture this through the following domain assumption:

Additional Domain Assumptions:

2) **Name:** *ExtractorWorks*
   Description: When the extractor is on, no methane is present.
   Formula: $\square(ext \to \neg m)$

Additional Controlled Variables: $\{ext\}$

Unlike in the original Mine Pump problem, in the Extended Mine Pump specification, the system can avoid reaching a state in which $h \wedge m$ holds by simply keeping the extractor on at all times.

Despite being avoidable by the mine pump controller, the formula $\Diamond(h \wedge m)$ remains a boundary condition for the Extended Mine Pump specification. However, presenting the boundary condition to an engineer to evolve the Extended Mine Pump specification may be a waste of time. The engineer may argue that there is no problem with the specification, a system can be built to satisfy it.

The problem is that the original definition of boundary

condition ignores what is controlled by the the mine pump controller. Thus boundary conditions may be attempting to signal problems in specifications when there are none. In the next section we define *Unavoidable Boundary Conditions* (UBCs) that extend the original definition of boundary condition with a notion of controlability taken from Reactive Synthesis.

B. Unrealizability Analysis

The Mine Pump specification is unrealizable. In other words, the mine pump controller has no strategy for setting the controllable variable p so that the goals G are achieved. In fact, the opposite is true. There is a counter-strategy (a strategy for the environment) for setting uncontrolled variables h and m that satisfies the domain assumptions Dom and does not violate the goals G.

One standard way of supporting an engineer in their attempt to understand and repair an unrealizable specification is to present them with a reduced specification with less goals but that is still unrealizable, the unrealizable core.

Unfortunately, for Mine Pump specification, there is no reduction. Its unrealizable core is the conjunction of *NoFlooding* and *NoExplosion*, each one individually is not a problem. We argue that presenting an engineer with the unavoidable boundary condition $\Diamond(h \wedge m)$ may be more informative than $\Box(m \rightarrow \bigcirc(\neg p)) \wedge \Box(h \rightarrow \bigcirc(p))$.

Do all unrealizable specifications have unavoidable boundary conditions that could be used for feedback instead of unrealizable cores? In Section V we show that this is the case.

Another way of supporting the exploration of an unrealizable specification to gain insights into causes and potential repairs is to have the engineer play through the counter-strategy that the environment has. Note that playing through a counter-strategy is not like playing through a (linear) execution of the system, rather it is a tree like structure in which the engineer explores how the counter-strategy reacts to different choices that a mine pump controller may make.

As we have an boundary condition that is unavoidable by the mine pump controller ($\Diamond(h \wedge m)$), there is a counter-strategy that the environment can use to force the unavoidable boundary condition. Such counter-strategy is also a counter-strategy to the entire specification (we show this in Section V). Thus, having an engineer play through the counter-strategy for the unavoidable boundary condition will not only give them insights as to the cause of unavoidability of that condition but also, at the same time, give them insight into the cause for unrealizability of the entire specification.

A common approach to furthering understanding of an unrealizable specification is to assume that a certain known bad environment condition cannot hold and to explore if the new specification, with stronger assumptions, is realizable. If it continues to be unrealizable, expoloration to identify new problems can continue. This strategy is also useful for repairing (i.e., making specifications realizable).

When introducing new assumptions it is important they be achievable by the environment, otherwise the system-to-be may implement the specification by forcing the assumptions to be violated. This problem is referred to in various ways in the literature, including well-separation [21], P-All and P-Reach [27], and assumption compatibility [17].

The negation of the unavoidable boundary condition $\Diamond(h \wedge m)$ can be added to original mine pump example as an assumption in *Dom* because it is achievable by the environment whenever it wants. It can set h and m to true as it controls them. Furthermore, by extending *Dom* with $\neg\Diamond(h \wedge m)$ we obtain a realizable specification.

Are all UBCs well-separated? No, as shown in Section VI. Can some UBCs be included as negated assumptions and yield realizable specifications? Yes, as shown in Section VI.

We now turn our attention back to the Extended Mine Pump specification. This is a realizable specification. In other words, the mine pump controller has a strategy for setting the controllable variable p and ext so that the goals G are achieved so long Dom holds.

The boundary condition $\Diamond(h \wedge m)$ is not of interest as we already know the mine pump controller can fulfill the specification. In fact, it will do so by avoiding the boundary condition. Indeed, as the specification is realizable it is guaranteed to not have UBCs (see Section V). Thus, it may not be worth presenting any boundary conditions at all to an engineer once the specification is realizable.

IV. Unavoidable Boundary Conditions for Conflict Analysis

We introduce unavoidable boundary conditions (UBCs) as boundary conditions that are unavoidable by the system. That is, under the assumptions of the domain, even without having to achieve its goals, it cannot achieve the negation of the boundary condition.

Definition 1 (Unavoidable Boundary Conditions): A formula ϕ is an Unavoidable Boundary Condition for the specification $(Dom, G, \mathcal{X}, \mathcal{Y})$ if:

$\{Dom, \phi, \bigwedge_{1 \leq i \leq n} G_i\} \models \bot$	(*inconsistency*)
$\{Dom, \phi, \bigwedge_{j \neq i} G_j\} \not\models \bot$, for $1 \leq i \leq n$	(*minimality*)
$BC \neq \neg(G_1 \wedge \ldots \wedge G_n)$	(*non-triviality*)
$Dom \rightarrow \neg\phi$ is unrealizable	*(unavoidability)*

Given the above, any UBC is also a BC by definition. An example to show that the reciprocal does not hold is given in the previous section which showed $\Diamond(h \wedge m)$ in the modified Mine Pump example is a boundary condition but not a UBC, because condition 4 does not hold. In other words, $Dom \rightarrow \neg\phi$ is realizable. The strategy for the system to achieve $\neg\Diamond(h \wedge m)$ is: $\sigma_{\mathcal{C}} : \{S_0, S_1\} \times \mathcal{P}(\mathcal{X}) \rightarrow \{S_0, S_1\} \times \mathcal{Y}$, and for any $x \subseteq \mathcal{X}$ we have $\sigma_{\mathcal{C}}(S_0, x) = (S_1, \{p, ext\})$, $\sigma_{\mathcal{C}}(S_1, x) = (S_1, \{p, ext\})$, i.e, the system strategy that consists of always turning on the extractor and the pump, is a winning strategy. For ease of reference, we will call BCs that do not meet the unavoidability condition, *avoidable*.

Note the it is straightforward to show that if ϕ is a UBC for $(Dom, G, \mathcal{X}, \mathcal{Y})$ then ϕ is a boundary condition for Dom, G.

However, the reciprocal does not hold.

A. UBCs in Boundary Condition Benchmarks

The fact that UBCs are a subset of the boundary conditions of a specification allows using existing boundary condition generation algorithms to find UBCs, only needing to filter out the boundary conditions that do not satisfy the unavoidability requirement. In the remainder of this section, we empirically investigate the following research question:

RQ1 - To what extent do existing methods for boundary condition generation and prioritization produce unavoidable boundary conditions.

Benchmarks. We first identify from the literature approaches that generate or prioritize boundary conditions:

1) ACORE: In [9] an automated goal-conflict resolution technique is proposed. The authors provide in a repository [10] a set of 25 LTL specifications, each with boundary conditions from which repairs are generated. The specifications include a Rail Road Crossing System [5] (rrcs), a Mine Pump Controller [23] (minepump), an elevator controller [39] (elevator), the TCP network protocol [19] (tcp), a telephone specification [19] (telephone) and four arbiter synchronization protocol specifications [9] (arbiter, simple_arbiter_v1, simple_arbiter_v2, s_arbiter_icse18).

Note that we identified a mismatch in the repository. The formulae given as boundary conditions for the achievepattern case study were actually boundary conditions for the specification under ATM folder which in turn was contained the achievepattern specification. We used the specification under the ATM folder as the achievepattern specification and did not consider the ATM case study.

2) CBC: In [25] an order relation between boundary conditions (contrasty) is proposed for filtering "redundant" boundary conditions, thus alleviating the validation effort of engineers. The case studies referred to in the paper are similar to those in [9]. However the original specifications used in the paper are not available. Also, the implementation of the algorithm for computing contrasty is not available. Hence, we re-implemented the contrasty filtering algorithm and applied it to the boundary conditions provided in [9] to generate contrasty boundary conditions (CBCs).

3) GENCON: In [15] a genetic algorithm is proposed for generating boundary conditions. They present 16 case studies with boundary conditions. Authors provide the results of 10 executions of their algorithm for each case study, including boundary conditions generated and the "minimal" subset of these i.e, those that logically imply all other boundary conditions. Of the 16 case studies, 11 are also in [9]. New case studies include an ATM specification (atm), and specifications taken from the SYNTCOMP synthesis competition repository [1]: a load balancer (loadbalancer), an advanced microcontroller bus architecture (AMBA) specification and a lift controller (liftcontroller). In our experimentation we did not use the LAS case study as we were unable to determine input and output variables from the specification provided.

4) MCOUNT: In [14], a prioritization method based on the probability of a boundary condition happening is presented—boundary conditions that have a non-zero probability of happening as time moves to infinity is prioritized. We will refer to these prioritized boundary conditions *likely* BCs, while others as *unlikely* BCs. The paper discusses 5 case studies with their corresponding boundary conditions. The case studies are a subset of those mentioned previously and were all considered in our experimentation, aside from LAS, which we do not cover for the reasons explained above.

5) SyntaxBC and SemanticBC: In [43], a new method for generating boundary conditions is proposed. The authors claim that the resulting conditions are of higher quality (i.e., relevance to engineers) than competing techniques ([15] and [25]). They use 15 case studies taken from [15]. The authors provided the boundary conditions they generated for each case study. As with [15], the LAS case study is excluded.

For each paper, we consider whether their BCs are UBCs by checking the unavoidability condition in Definition‘1. For this, we use the tool Strix [30] which supports LTL unrealizability checks. We also deploy the Spot tool [18] to conduct the satisfiability and implication checks needed for our implementation of the CBC algorithm. The boundary conditions and the UBCs for all results reported below can be found at [2]

Results. In Table I, we show the number of boundary conditions detected (#BCs) for each specification in [9], the number of BCs that are UBCs (#UBCs) and the reduction (presented as a percentage) in the number of boundary conditions that would be presented to an engineer for validation %Red. The percentage varies greatly across the case studies. Some, such as *achievepattern* have no avoidable BCs, thus restricting validation of the boundary conditions in [9] to UBCs makes no difference for these case studies. Others, like s_arbiter_icse18 and round-robin, had 100% of avoidable boundary conditions, which questions the quality of the feedback that [9] generates for that case study. In the case of round-robin, other techniques do generate UBCs (see Table V), which would provide better feedback to engineers. In the case of s_arbiter_icse18, as we will discuss below, there are no UBCs because the specification is realizable and it may be pointless to validate all the avoidable BCs found.

The aggregate result of Table I shows that [9] produces 65% of boundary conditions that are avoidable on average, thus focusing on UBCs may reduce the validation effort significantly.

Table I also shows the number of BCs resulting from filtering out boundary conditions that according to contrasty (#CBCs), the number of those that are UBCs (#UCBCs) and finally the reduction (presented as a percentage) in the number of CBCs that would be presented to an engineer for validation as a result.

Results show, on one hand, that there are 9 case studies for which all CBCs were UBCs (i.e., 0% reduction): achievepattern, arbiter, detector, lily01, lily02, lily15, minepump, priori-

TABLE I
UNAVOIDABLE BOUNDARY CONDITIONS IN [9] AND UNAVOIDABLE CONTRASTY BOUNDARY CONDITIONS IN [25]

Specifications	#BCs	#UBCs	%Red.	#CBCs	#UCBCs	%Red.
RG2	9	6	33%	1	0	100%
achievepattern	40	40	0%	1	1	0%
arbiter	20	19	5%	1	1	0%
detector	15	7	53%	1	1	0%
elevator	3	2	33%	3	2	33%
lily01	5	5	0%	1	1	0%
lily02	11	10	9%	1	1	0%
lily11	4	3	25%	4	3	25%
lily15	19	16	15%	1	1	0%
lily16	36	4	88%	1	0	100%
ltl2dba27	10	3	70%	10	3	70%
ltl2dba_R_2	5	2	60%	5	2	60%
ltl2dba_theta_2	2	1	50%	2	1	50%
minepump	15	11	26%	1	1	0%
prioritizedArbiter	9	7	22%	2	2	0%
retractionpattern1	2	2	0%	1	1	0%
retractionpattern2	10	10	0%	1	1	0%
round-robin	11	0	100%	1	0	100%
rrcs	14	14	0%	1	1	0%
s_arbiter_icse18	17	0	100%	1	0	100%
simple_arbiter_v1	27	23	14%	1	1	0%
simple_arbiter_v2	20	15	25%	1	1	0%
tcp	10	3	70%	3	1	66%
telephone	3	3	0%	1	1	0%
Total	**317**	**206**	**35%**	**46**	**27**	**41%**

tizedArbiter, simple_arbiter_v1, and telephone. This provides some indication that the contrasty condition can reduce the number of boundary conditions to be validated while keeping the focus on conditions that cannot be avoided and thus must be fixed. However, two case studies (RG2 and lily16) show very clearly that the contrasty condition can lose relevant feedback; both specifications had several UBCs but these have been lost when the contrasty filter was applied. Moreover, the unavoidable $\Diamond(h \wedge m)$ boundary condition discussed in Section III is lost when filtering out for contrasty in the minepump problem. As before, aggregate results demonstrate that there is a significant amount of contrasty boundary conditions that are avoidable, and that focus on unavoidability would reduce the number of conditions to be validated by 41% on average.

In Table II, we report on boundary conditions generated using [15]. Note that these are not a subset of the conditions used for Table I. Recall [15] use randomization to generate boundary conditions, thus, for each case study the authors run the generation algorithm 10 times providing 10 sets of boundary conditions. Then a minimal set containing only the those not implied by others are obtained. We refer to the former set as "non-minimized", and the second as minimized. In Table II, the first three columns report on the non-minimized set where: the first column is the average number of boundary conditions generated for each case study over the runs that produced at least one boundary condition; the second is the average number of UBCs that these runs generated; while the third column is the average proportion of boundary conditions that are not UBCs. The next three columns summarize results for the runs after the minimization procedure in [15] is used.

While the percentages vary greatly from one case study to another, there are a few case studies where practically no boundary conditions are UBCs (e.g., retractionpattern1, retractionpattern2, and telephone) despite the fact that these specifications do have UBCs (see Table I). This highlights that there is room for improvement in the heuristics used in [15] to find UBCs rather than avoidable boundary conditions.

Note that there is a connection between unavoidability and being minimal, in the sense that, if $\phi \rightarrow \psi$ and ϕ is unavoidable so is ψ and thus any non-minimal avoidable boundary condition must be implied by an avoidable boundary condition. Thus, minimizing the set of boundary conditions will not lead to losing all avoidable boundary conditions (as in Table I). However, minimization changes the reduction percentages as this depends heavily on the number of boundary conditions implied by a minimal boundary condition.

Similarly to previous results discussed, Table II shows that the proportion of non-UBCs generated by [15] is not negligible, thus supporting the reduction in the number of conditions that need to be validated.

In Table III, we report on the number of UBCs over the number of boundary conditions prioritized according to the criteria developed in [14] based on model counting. We split the table into two parts: likely and unlikely boundary conditions as done in [14]. Despite the sample size being considerably reduced when compared to previously discussed papers, the results in Table III show that model counting is not strong enough to filter out avoidable formulas. Moreover, in the case of minepump, the the unavoidable one is the unlikely boundary condition, whereas those reported as likely to happen are in fact avoidable. We believe that model counting may be a complementary approach to UBCs, but should be geared towards counting "unavoidable models" or counter-strategies rather than models based on satisfiability.

Finally, Tables IV and V consider boundary conditions generated by the SemanticBC and SyntaxBC tools reported in [43] respectively. Table IV shows that the number of confirmed boundary conditions that the semantic approach generates is quite small for most cases—a significant number of case studies lack even one boundary condition. In other words, the SemanticBC criteria is very stringent. However, in most cases, the criteria drops all the unavoidable boundary conditions, thus de-prioritizing conditions that the system-to-be cannot avoid and that lead to conflicts.

On the other hand, in Table V, we report on the UBCs generated by SyntaxBC both before and after applying a contrasty filter on them (in [43], both sets of boundary conditions are discussed). SyntaxBC generated considerably more conditions than SemanticBC, and also more unavoidable conditions. Nevertheless, there number of avoidable boundary conditions that can be dropped in validation is in the order of 80%.

In summary, results show that current boundary condition generation and prioritization methods yield a considerable proportion of boundary conditions that can be avoided by the system-to-be, and consequently do not need to be validated with engineers.

TABLE II
UNAVOIDABLE BOUNDARY CONDITIONS IN [15] (AVERAGED OVER UP TO 10 RUNS)

Specifications	Non-Minimized			Minimized		
	#BCs	#UBCs	%Red.	#BCs	#UBCs	%Red.
achievepattern	21.6	21.6	0%	3.4	3.4	0%
atm	10	0.33	98.2%	2.33	0.11	89%
amba	2.833	0	100%	2.33	0	100%
elevator	7.7	7.3	4%	2.4	2.2	7%
liftcontroller	3.4	1.4	80%	2.8	1.2	80%
loadbalancer	3	0	100%	1.66	0	100%
minepump	18	8.6	39%	3.8	2.6	22%
prioritizedArbiter	13.5	9.0	37%	2.25	2	9%
retractionpattern1	27.8	27.8	0%	3.3	3.3	0%
retractionpattern2	22.75	22.75	0%	2.25	2.25	0%
rrcs	16.125	15.625	4%	3.5	3.5	0%
s_arbiter_icse18	15.25	0	100%	2.375	0	100%
tcp	8.33	0.55	95.1%	2.11	0.33	90.8%
telephone	24	24	0%	3	3	0%
round-robin	37.11	0	100%	4	0	100%
Total	**3704**	**2168**	**41.4%**	**642**	**378**	**41.1%**

TABLE III
UNAVOIDABLE BOUNDARY CONDITIONS IN [14]

Specifications	Likely BCs			Unlikely BCs		
	#BCs	#UBCs	%Red.	#BCs	#UBCs	%Red.
atm	1	0	100%	2	0	100%
minepump	1	0	100%	1	1	0%
tcp	2	1	50%	0	0	N/A
telephone	1	1	0%	0	0	N/A
Total	**5**	**2**	**60%**	**3**	**1**	**66%**

TABLE IV
UNAVOIDABLE SEMANTICBC BOUNDARY CONDITIONS IN [43]

Specifications	#BCs	#UBCs	%Red.
achievepattern	1	1	0%
atm	0	0	N/A
elevator	0	0	N/A
liftcontroller	0	0	N/A
loadbalancer	1	0	100%
minepump	3	0	100%
s_arbiter_icse18	0	0	N/A
prioritizedArbiter	45	0	100%
retractionpattern1	0	0	N/A
retractionpattern2	0	0	N/A
round-robin	1	0	100%
rrcs	0	0	N/A
tcp	0	0	N/A
telephone	0	0	N/A
Total	**51**	**1**	**98%**

TABLE V
UNAVOIDABLE SYNTAXBC BOUNDARY CONDITIONS IN [43]

Specifications	Not Filtered			Filtered by Contrasty		
	#BCs	#UBCs	%Red.	#BCs	#UBCs	%Red.
achievepattern	4	4	0%	1	1	0%
atm	4	0	100%	1	0	100%
elevator	4	4	0%	2	2	0%
liftcontroller	0	0	N/A	0	0	N/A
loadbalancer	0	0	N/A	0	0	N/A
minepump	4	4	0%	1	1	0%
s_arbiter_icse18	7	0	100%	2	0	100%
prioritizedArbiter	12	12	0%	3	3	0%
retractionpattern1	4	4	0%	2	2	0%
retractionpattern2	4	4	0%	1	1	0%
round-robin	6	4	33%	3	2	33%
rrcs	3	3	0%	2	2	0%
tcp	4	4	0%	2	2	0%
telephone	4	4	0%	2	2	0%
Total	**60**	**47**	**21%**	**22**	**18**	**18%**

V. UNAVOIDABLE BOUNDARY CONDITIONS FOR UNREALIZABILITY ANALYSIS

In the previous section, we considered UBCs in the context of conflict analysis. In this section, we discuss UBCs in the context of providing feedback on unrealizable specifications. We analyzed all the specifications in [9]. These are all unrealizable except for one: s_arbiter_icse18.

A. Counter-strategies and realizability

Counter-strategies provide concrete evidence of how the environment can prevent the system-to-be from satisfying a specification. If a specification has a UBC ϕ, then there is a strategy that shows how the environment can force ϕ. It also means there is a counter-strategy for the goal $\neg\phi$. Thus, by exploring the counter-strategy for ϕ, an engineer is also exploring a counter-strategy for the entire specification. The following theorem shows that this is the case:

Theorem 1: (UBC counter-strategies are specification counter-strategies) If ϕ is a UBC to specification $\mathcal{S} = (Dom, G, \mathcal{X}, \mathcal{Y})$ then every counter-strategy σ to $Dom \rightarrow \neg\phi$ is a also a counter-strategy to $Dom \rightarrow G$, i.e, σ is a counter-strategy for $\mathcal{S}$.

Proof: Let σ be a counter-strategy to $Dom \rightarrow \neg\phi$. Thus, $\sigma \models Dom \wedge \phi$. Since ϕ is a UBC, $Dom \wedge G \wedge \phi \models \bot$, or equivalently, $(Dom \wedge \phi) \rightarrow \neg G$ is a tautology. Therefore, $\sigma \models Dom \wedge \neg G$ and thus, σ is a counter-strategy to $Dom \rightarrow G$. ■

A corollary of the above theorem is that the existence of a UBC means that the specification is unrealizable.

Corollary 1: If a specification $(Dom, G, \mathcal{X}, \mathcal{Y})$ has a UBC then it is unrealizable

The results in the previous section are consistent with this result; no UBCs were found for the realizable specification (s_arbiter_icse18). Of course, for some non-realizable specification, no UBCs were found. However, this may be attributed to the boundary condition generation and prioritization methods considered in our study.

It is worth investigating the other direction of the implication, i.e., that an unrealizable specification always has a UBC. This is not the case due to the limited expressiveness of LTL which cannot encode arbitrary counter-strategies of unrealizable specification as these have memory, which require additional propositional variables to express. However, we show a proof for a slightly weaker result:

Theorem 2: (Unrealizable specifications have UBCs) Let $\sigma_{\mathcal{E}} : S \times \mathcal{P}(\mathcal{Y}) \rightarrow S \times \mathcal{P}(\mathcal{X})$ be a counter-strategy to a specification $(Dom, G, \mathcal{X}, \mathcal{Y})$ and $n = |S|$. If $\mathcal{V}_{prop} = \{s_1, s_2, ..., s_n\}$ a set of propositional variables not in $\mathcal{X} \cup \mathcal{Y}$. Then there exists a UBC ϕ for the specification $(Dom, G, \mathcal{X} \cup \mathcal{V}_{prop}, \mathcal{Y})$

Proof: We construct a formula that satisfies the property, thus proving the theorem. The variables in $\mathcal{V}_{prop}$ are going to represent the n different memory states $\sigma_{\mathcal{E}}$ can be in. Without

loss of generality, S_1 will be the initial state. Given $(a, b) \in A \times B$ we will use $(a, b)_A$ and $(a, b)_B$ to refer to a and b respectively.

The UBC ϕ will be the formula $\mathit{OnlyState}_1 \wedge \Box((\bigvee_{1\leq i\leq n} \mathit{OnlyState}_i) \wedge (\bigwedge_{1\leq i\leq n}(s_i \rightarrow \bigvee_{P\in\mathcal{P}(\mathcal{X})} \mathit{VarState}_{(S_i,P)} \wedge \mathit{Transition}_{(S_i,P)})))$. Formula ϕ represents the counter-strategy. The $\mathit{OnlyState}_1$ term represents that the counter-strategy starts in S_1, and $(s_i \rightarrow \bigvee_{P\in\mathcal{P}(\mathcal{X})} \mathit{VarState}_{(S_i,P)} \wedge \mathit{Transition}_{(S_i,P)}))$ represents that if the counter-strategy is in state S_i and the controller plays P, then the state of the variables is $\mathit{VarState}_{(S_i,P)}$ and the next state is s_j.

If w is an LTL trace (which without loss of generality only has variables in $\mathcal{X} \cup \mathcal{Y}$) such that $w \models \phi$, then clearly, w corresponds to the induced trace of a run of σ and thus $w \models Dom \wedge \neg G$. Therefore, $\phi \wedge Dom \wedge G \models \bot$. In addition, since ϕ utilizes propositional variables not in G, it is non-trivial. Furthermore, ϕ is unavoidable when looking at control problem $(Dom, G, \mathcal{X} \cup \mathcal{V}_{prop}, \mathcal{Y})$ since $\sigma'_{\mathcal{E}} : S \times \mathcal{P}(\mathcal{Y}) \rightarrow S \times (\mathcal{P}(\mathcal{X} \cup \mathcal{V}_{prop}))$, $\sigma'_{\mathcal{E}}(S_i, P) = (\sigma_{\mathcal{E}}(S_i, P)_S, \sigma_{\mathcal{E}}(S_i, P)_{\mathcal{P}(\mathcal{Y})} \cup \{s_i\})$ is trivially a counter-strategy. ■

Remark 1: (Adding uncontrollable variables does not change realizability) Let $(Dom, G, \mathcal{X}, \mathcal{Y})$ be a control problem, $\mathcal{V}'$ a set of propositional variables such that $\mathcal{V}' \cap \mathcal{X} = \mathcal{V}' \cap \mathcal{Y} = \emptyset$. Then the control problem $(Dom, G, \mathcal{X} \cup \mathcal{V}', \mathcal{Y})$ is realizable if and only if $(Dom, G, \mathcal{X}, \mathcal{Y})$ is realizable.

Finally, to support that reporting avoidable boundary conditions to requirement engineers does not help characterize problems in an unrealizable specification, note that every counter-strategy to a non-realizable problem has a run in which the avoidable boundary condition does not hold. This indicates that despite a controller succeeding in avoiding the boundary condition, it still fails to achieve the specification.

Remark 2: (Counter-strategies have runs where avoidable boundary conditions do not occur) Let $(Dom, G, \mathcal{X}, \mathcal{Y})$ be a non-realizable control problem and σ a counter-strategy. If ϕ is an avoidable boundary condition, then there is a run π of σ such that $\pi \models \neg\phi$.

Proof: Let $\sigma_{\mathcal{E}} : S_{\mathcal{E}} \times \mathcal{P}(\mathcal{Y}) \rightarrow S_{\mathcal{E}} \times \mathcal{P}(\mathcal{X})$ be a counter-strategy to the control problem and $\sigma_{\mathcal{C}} : S_{\mathcal{C}} \times \mathcal{P}(\mathcal{X}) \rightarrow S_{\mathcal{C}} \times \mathcal{P}(\mathcal{Y})$ the controller's winning strategy for $Dom \rightarrow \neg\phi$. Let $S^0_{\mathcal{E}}$ and $S^0_{\mathcal{C}}$ be the initial states of the environment's and the controller's strategies, respectively. Let $E_0 = \sigma_{\mathcal{E}}(S^0_{\mathcal{E}}, \emptyset)_{\mathcal{P}(\mathcal{X})}$ be the environment's initial play (recall it is constant with respect to the second parameter as the environment cannot react to the controller's play before their turn). Moreover, we define $(S^1_{\mathcal{C}}, C_0) = \sigma_{\mathcal{C}}(S^0_{\mathcal{C}}, E_0)$ and $(S^1_{\mathcal{E}}, E_0) = \sigma_{\mathcal{E}}(S^0_{\mathcal{E}}, C_0)$.

We now recursively define $E_{n-1} = \sigma_{\mathcal{E}}(S^{n-1}_{\mathcal{E}}, \emptyset)_{\mathcal{P}(\mathcal{X})}$, $(S^n_{\mathcal{C}}, C_{n-1}) = \sigma_{\mathcal{C}}(S^{n-1}_{\mathcal{C}}, E_{n-1})$ and $(S^n_{\mathcal{E}}, E_{n-1}) = \sigma_{\mathcal{E}}(S^{n-1}_{\mathcal{E}}, C_{n-1})$. What we are doing here is basically simulating the realizability game, with each player employing their strategy to achieve their desired goal. Thus, the resulting run should achieve both the negation of the control problem and the negation of the boundary condition, thus proving our result.

By definition, $(S^n_{\mathcal{E}}, C_n)_{n\in\mathbb{N}_0}$ and $(S^n_{\mathcal{C}}, E_n)_{n\in\mathbb{N}_0}$ are runs of $\sigma_{\mathcal{E}}$ and $\sigma_{\mathcal{C}}$ respectively. Note that the induced LTL traces of both runs is $\mathcal{I}_0\mathcal{I}_1...$, where $\mathcal{I}_0 = (C_0 \cup E_0)$ and $\mathcal{I}_1 = (C_1 \cup E_1)$ and therefore, $\mathcal{I}_0\mathcal{I}_1... \models \neg(Dom \rightarrow G)$ and $\mathcal{I}_0\mathcal{I}_1... \models \neg\phi$ ■

B. Unrealizable Cores

One advantage of providing feedback in terms or unrealizable cores is that it is provided in the same specification language as the original specification. UBCs also have this advantage. However, they have the potential to provide more focused feedback than unrealizable cores. Recall the example in Section III, in which the unrealizable core for the mine pump specification is the specification itself, while the UBC $\Diamond(h \wedge m)$ arguably provides more focused information.

As expected, there is a close relation between UBCs and unrealizable cores. Indeed, if $C \subseteq \{G_1, \ldots G_n\}$ is an unrealizable core, it satisfies the inconsistency property of boundary conditions (i.e., $\neg(\bigwedge_{G_i\in C} G_i) \wedge Dom \models \bot$), and also the unavoidability property of UBCs (i.e., $Dom \rightarrow \neg \bigwedge_{G_i\in C} G_i$) is unrealizable).

Strictly speaking, however, $\bigwedge_{G_i\in C} G_i$ is not a boundary condition to $\{G_1, \ldots G_n\}$ as the minimality condition does not hold, nor is it a boundary condition to C as the non-triviality condition does not hold.

Nevertheless, the relationship with unrealizable cores is not limited to that. Due to Theorem 1, given a UBC ϕ, all counter-strategies to $Dom \rightarrow \neg\phi$ are counter-strategies to $Dom \rightarrow G$, meaning all counter-strategies for UBCs violate at least one unrealizable core. Moreover, from Theorem 2 it is clear that, with an extension of the alphabet of the specification, every unrealizable core has a UBC whose associated counter-strategies violate it.

Finally, as we illustrate in Section III, UBCs can provide a simpler explanation to what the conflict cause may be than minimal cores.

VI. Unavoidable Boundary Conditions for Unrealizability Repair

There are several approaches to repairing unrealizable specifications (e.g., [4], [11], [22], [28]). Assumption refinement is one in which new assumptions are inferred from counter-strategies and then added to eliminate such counter-strategies which make the specification unrealizable (e.g., [11]).

Given that UBCs display unavoidable events that lead to the controller "losing", then a potential way of repairing specifications could be to add the negation of a UBC to refine the current assumption. That way, the controller is given the guarantee that the UBC will not be reached. Nevertheless, if the controller can force the UBC to happen, that repair could result in a non-interesting specification, as the controller could achieve a win by forcing the assumptions to be violated, instead of reaching the goals.

This issue ties into the concept of *well-separation*. A control problem $(Dom, G, \mathcal{X}, \mathcal{Y})$ is well-separated if for any reachable state, there is no strategy for the controller that forces the environment to violate the assumptions. Thus, it is interesting to ask if there is a relation between a boundary

condition being unavoidable and it being well-separated when introduced as a repair to an unrealizable specification.

We refer to a UBC ϕ as being well-separated for $(Dom, G, \mathcal{X}, \mathcal{Y})$ if $(Dom \wedge \neg\phi, G, \mathcal{X}, \mathcal{Y})$ is well separated. There is no direct relation between a boundary condition being unavoidable and it being well separated. In fact, no implication between the two holds.

An example of avoidable boundary condition that is well separated is $\Diamond(h \wedge m)$ in the modified mine pump example. As previously mentioned, it is avoidable because the controller can simply force $\Box\neg m$ by keeping ext true. It is well separated because the environment can, at any point, achieve $Dom \wedge \neg\Diamond(h \wedge m)$ by simply setting h and m to false. The point is that the controller can force $\neg m$ but cannot force m. Counter-examples for the other implications are as follows: Suppose p is a controllable variable and q an uncontrollable one. Formula $\Diamond q$ could be a UBC and well separated. $\Diamond p$ can be an avoidable boundary condition and non-well separated, while $\Diamond(p \vee q)$ is unavoidable and non-well separated.

Having established that UBCs and well-separation are related but not logically implied concepts, we return to the following research question:

RQ2 - To what extent can UBCs induce, once negated, assumption refinements to repair unrealizable specifications without trivializing them (allowing the controller to achieve its goals by forcing the UBC).

To answer this question we look at the *assumption compatibility* notion studied in [17] which requires the assumptions to be realizable by the environment. Thus, if we are to negate a UBC ϕ and add it to the assumptions, we must check that $\neg(Dom \wedge \neg\phi) \equiv Dom \rightarrow \phi$ is not realizable by the system, which can be checked with the Strix tool.

What is important about assumption compatibility then is that if refining Dom with the negation of a UBC results in compatible assumptions, then it can be used to move towards a non-trivial repair. In Table VI, we show that most of the UBCs found in the benchmark [9], when added negated to Dom, result in compatible assumptions.

It is interesting to explore if these compatible assumptions result in a realizable specification. We simulated an engineer using the feedback provided as UBCs in the following way: For each case study, we selected a UBC that could be used to obtain an assumption refinement, and then checked its realizability. If the specification was not realizable we repeated the procedure for another UBC, and so on. The last column of Table VI shows that for most of the case studies (76%, 16 out of 21), a realizable specification was reached. Note that we do not consider 3 of the 24 case studies: s_arbiter_2018 because it is realizable, round robin because it had no UBCs nor lily16 because it had no compatible UBCs. The four for which there were compatible UBCs but a realizable specification was not achieved (RG2, lily15, priritizedArbiter, and telephone) may be due to limitations of [9]; it was not capable of finding UBCs that would have repaired the specification.

TABLE VI
REPAIR USING COMPATIBLE UNAVOIDABLE BOUNDARY CONDITIONS IN [9]

Specifications	#UBC	#Comp. UBCs	$\frac{\#\text{Comp. UBCs}}{\#\text{UBCs}}$ %	Repaired
RG2	6	4	66%	No
achievepattern	40	40	100%	Yes
arbiter	19	19	100%	Yes
detector	7	7	100%	Yes
elevator	2	2	100%	Yes
lily01	5	1	20%	Yes
lily02	10	5	50%	Yes
lily11	3	2	66%	Yes
lily15	16	2	12.5%	No
ltl2dba27	3	3	100%	Yes
ltl2dba_R_2	2	2	100%	Yes
ltl2dba_theta_2	1	1	100%	No
minepump	11	11	100%	Yes
prioritizedArbiter	7	7	100%	No
retractionpattern1	2	2	100%	Yes
retractionpattern2	10	10	100%	Yes
rrcs	14	14	100%	Yes
simple_arbiter_v1	23	23	100%	Yes
simple_arbiter_v2	15	14	93%	Yes
tcp	3	3	100%	Yes
telephone	3	3	100%	No
Total	**202**	**175**	**87%**	**16/21**

In conclusion, the fact that 76% of the specifications were repairable provides evidence that a repair can be achieved by exclusively inspecting unavoidable boundary conditions.

VII. DISCUSSION AND RELATED WORK

An alternative definition of UBC that solely relies on notions of realizability and is devoid of inconsistency requirement may be desirable. To this end, an argument could be made for changing the logical inconsistency condition in the boundary condition definition (Definition 1) with the condition that all counter-strategies of $\neg\phi$ should be counter-strategies of the specification. This would achieve a UBC definition that only deals with realizability instead of mixing satisfiability and realizability. However, such definition leads to a strictly weaker definition than the one we propose, and would yield conditions that fail to give meaningful insight into problems with the specification. As an example, consider the condition $\psi = \neg(p \wedge q)$ for the specification $(\top, p, \{p\}, \{q\})$. ψ satisfies the alternative proposal for UBCs: it is unavoidable and the only counter-strategy to $\neg\psi = p \wedge q$ is by setting p to $false$, which is a counter-strategy for the specification too. However, ψ is not a good indicator of the issues with the specification, q is irrelevant. In fact, ψ does not even satisfy the inconsistency requirement of boundary conditions (i.e., $\top \wedge \neg(p \wedge q) \wedge p$ is satisfiable). A benefit of keeping the inconsistency requirement is that checking that all counter-strategies of $\neg\psi$ are counter-strategies of the specification is computationally more expensive than checking for satisfiability.

Seminal literature in requirements engineering stresses the importance of considering controlability aspects of the agents that are to enforce specifications (e.g., [33], [44]). However, these notions related to control have been neglected both in the original formulation of boundary conditions [40] and subsequent work (e.g., [9], [16], [41]).

As noted in [41], "Resolving inconsistencies [including boundary conditions] sooner or later in the process is a necessary condition for successful development of the software implementing those requirements". Eliminating avoidable boundary conditions is not a necessary condition for implementing (i.e., realising) the requirements, while eliminating UBCs is. Providing avoidable BCs as feedback may trigger valuable discussions as any other property inferrable (and not obvious) of the specification. However, including avoidable BCs, at least to start with, may be overwhelming and may mislead engineers to unnecessarily limit the system behaviour or strengthen environment assumptions to pursue realisability.

We have mentioned existing work on boundary condition generation and prioritization in previous sections (e.g., [9], [15], [16], [25], [43]). Regarding conflict resolution, a number of manual-based approaches exist. The work in [40], discusses several pattern-based strategies to *resolving conflicts* in goals. One such approach is avoiding boundary conditions by adding the condition $\neg BC$ to the domain assumptions. However, as discussed, introducing non-compatible assumptions yields trivially realizable specifications in which the system can trivially achieve the specification. Instead, a preferable resolution approach would be to elicit a weakened version of G, G' (i.e., $G \rightarrow G'$) and require it to be achieved when the UBC ϕ occurs (i.e., $Dom \rightarrow (G \vee (\phi \wedge G'))$. Other pattern-based resolution strategies are discussed in [40]. In [31] the use of argumentation frameworks for stakeholders to model and resolve conflicts is proposed.

In this paper we studied the relationship between UBCs and counter-strategies. Justice Violations Transition Systems (JVTS) [24] are an alternative to counter-strategies for debugging unrealizable specifications. These structures are more concise and their computation is more time-efficient than counter-strategies. Given that JVTS have the same expressive power as counter-strategies, similar results to those reported in Section V are likely to hold when relating JVTS and UBCs.

Using UBCs as a means for achieving realizable specifications is related to approaches for *assumption refinement* in reactive synthesis (e.g., [11], [12]). In [11] for instance, an approach based on Craig's interpolation is devised to extract LTL formulas that are intended to capture the 'cause' for a run in a counter-strategy for violating a set of goals. The relationship between such assumptions and boundary conditions is yet to be fully explored. However, it is expected that the formulae extracted via interpolation are not necessarily boundary conditions, since the minimality condition of UBCs at the least is not guaranteed.

The notion of realizability is relevant to controlability in discrete event control [34]. While in Reactive Synthesis, system-to-be and environment interact through shared variables and in strict (lockstep) turns, in discrete event control interaction is through events, akin to message passing, and execution is asynchronous (no assumptions on turns is made). We adopt notions of Reactive Synthesis because its semantics (lockstep and shared variables) aligns better with requirements engineering literature on conflict analysis (e.g., [9], [41]). However, the notion of unavoidable boundary conditions can be adapted to discrete event control that use event-based temporal logic specifications (e.g., [7], [32]). Realizability is also related to solvable-planning problems in non-deterministic AI planning (e.g., [35], [42]). Although there are differences in how control problems are represented and expressiveness (mainly related to liveness properties) there are results that show certain equivalences between planning and reactive synthesis (e.g., [8], [38]. It should be possible to transfer the notions discussed here to specifications described as planning problems.

VIII. Conclusion

In this paper, we argued that a limitation of the existing definition of boundary conditions is that it does not distinguish between those conditions that can be avoidable by a system-to-be and those that are not. That led us to propose a refined definition that borrows from the reactive synthesis community the notion of realizability, called unavoidable boundary conditions (UBCs, Definition 1). We showed empirically (Section IV-A) that current boundary condition generation and prioritization methods yield a considerable proportion of avoidable boundary conditions which could be, at the very least, given less priority at validation time than UBCs.

We also showed that UBCs can provide feedback for gaining insight on unrealizable specifications complementary to specification counter-strategy playback and manual inspection of unrealizability cores. We demonstrated that the existence of UBCs and unrealizability are equivalent notions (Corollary 1 and Theorem 2). This means that focusing on understanding and repairing UBCs can lead to repairing unrealizable specifications. We showed (Theorem 1) that counter-strategies to a UBC are indeed counter-strategies to the agent goals. This means that using counter-strategies to understand UBCs may help understanding the unrealizability of the entire specification too. In addition, Remark 2 provides an argument as to why presenting avoidable boundary conditions to engineers may not be helpful as all counter-strategies of an unrealizable specification include examples (i.e., runs) that show that unrealizability does not depend on avoiding the boundary condition. We also exemplify (Section III) UBCs can provide alternative, finer grained, explanations to unrealizability than unrealizable cores.

Finally, we discussed the potential of using UBCs to repair unrealizable specifications (Section VI). We showed that the assumptions refined using UBCs may or may not be well-separated, but when compatible, they can be lead to repaired specifications in many cases.

Future work includes revisiting boundary condition generation and prioritization approaches to include novel methods and heuristics for computing UBCs instead of avoidable boundary conditions. Also, a more thorough comparison between existing assumption refinement approaches and the use of UBCs for repair is required.

Acknowledgments: This work was partially supported by Pict 2019-1442, 2019-1793, and 2021-4862; Ubacyt 20020190100233BA and 20020220300134BA; IA-1-2022-1-173516 IDRC-ANII; and UK EPSRC grant EP/Y037421/1.

REFERENCES

[1] https://bitbucket.org/swenjacobs/syntcomp.

[2] https://zenodo.org/doi/10.5281/zenodo.13317017.

[3] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming*, pages 1–17, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[4] Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-strategy guided refinement of GR(1) temporal logic specifications. *CoRR*, abs/1308.4113, 2013.

[5] Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. Symbolic causality checking using bounded model checking. In Bernd Fischer and Jaco Geldenhuys, editors, *Model Checking Software - 22nd International Symposium, SPIN 2015, Stellenbosch, South Africa, August 24-26, 2015, Proceedings*, volume 9232 of *Lecture Notes in Computer Science*, pages 203–221. Springer, 2015.

[6] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012. In Commemoration of Amir Pnueli.

[7] Victor Braberman, Nicolas D'Ippolito, Nir Piterman, Daniel Sykes, and Sebastian Uchitel. Controller synthesis: From modelling to enactment. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1347–1350, 2013.

[8] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Towards a unified view of ai planning and reactive synthesis. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):58–67, May 2021.

[9] Luiz Carvalho, Renzo Degiovanni, Matías Brizzio, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, and Mike Papadakis. Acore: Automated goal-conflict resolution. In *Fundamental Approaches to Software Engineering: 26th International Conference, FASE 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings*, page 3–25, Berlin, Heidelberg, 2023. Springer-Verlag.

[10] Luiz Carvalho, Renzo Degiovanni, Matías Brizzio, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, and Mike Papadakis. https://sites.google.com/view/acore-goal-conflict-resolution/. 2023.

[11] Davide G. Cavezza and Dalal Alrajeh. Interpolation-based GR(1) assumptions refinement. In Axel Legay and Tiziana Margaria, editors, *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*, volume 10205 of *Lecture Notes in Computer Science*, pages 281–297, 2017.

[12] Davide G. Cavezza, Dalal Alrajeh, and András György. Minimal assumptions refinement for realizable specifications. In *2020 IEEE/ACM 8th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 66–76, 2020.

[13] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In Francesco Logozzo, Doron A. Peled, and Lenore D. Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 52–67, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[14] Renzo Degiovanni, Pablo Castro, Marcelo Arroyo, Marcelo Ruiz, Nazareno Aguirre, and Marcelo Frias. Goal-conflict likelihood assessment based on model counting. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1125–1135, 2018.

[15] Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. A genetic algorithm for goal-conflict identification. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 520–531, 2018.

[16] Renzo Degiovanni, Nicolas Ricci, Dalal Alrajeh, Pablo Castro, and Nazareno Aguirre. Goal-conflict detection based on temporal satisfiability checking. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 507–518, 2016.

[17] Nicolás D'ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.*, 22(1), mar 2013.

[18] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022.

[19] Amy P. Felty and Kedar S. Namjoshi. Feature specification and automated conflict detection. *ACM Trans. Softw. Eng. Methodol.*, 12(1):3–27, 2003.

[20] Anthony Hunter and Bashar Nuseibeh. Analyzing inconsistent specifications. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, page 78, USA, 1997. IEEE Computer Society.

[21] Uri Klein and Amir Pnueli. Revisiting synthesis of gr(1) specifications. In Sharon Barner, Ian Harris, Daniel Kroening, and Orna Raz, editors, *Hardware and Software: Verification and Testing*, pages 161–181, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[22] Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):563–583, 2013.

[23] Jeff Kramer, Jeff Mageee, Moris Sloman, and A. Lister. Conic: an integrated approach to distributed computer control systems. *IEE Proceeedings E*, 130(1):1–10, 1983.

[24] Aviv Kuvent, Shahar Maoz, and Jan Oliver Ringert. A symbolic justice violations transition system for unrealizable GR(1) specifications. In *11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, number 1, pages 362–372, 2017.

[25] Weilin Luo, Hai Wan, Xiaotong Song, Binhao Yang, Hongzhen Zhong, and Yin Chen. How to identify boundary conditions with contrasty metric? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1473–1484, 2021.

[26] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[27] Shahar Maoz and Jan Oliver Ringert. On Well-Separation of GR(1) Specifications. 1 2016.

[28] Shahar Maoz, Jan Oliver Ringert, and Rafi Shalom. Symbolic repairs for gr(1) specifications. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1016–1026, 2019.

[29] Shahar Maoz and Yaniv Sa'ar. Counter play-out: Executing unrealizable scenario-based specifications. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 242–251, 2013.

[30] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 578–586, Cham, 2018. Springer International Publishing.

[31] Pradeep K. Murukannaiah, Anup K. Kalia, Pankaj R. Telangy, and Munindar P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 156–165, 2015.

[32] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel. Assured and correct dynamic update of controllers. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '16, page 96–107, New York, NY, USA, 2016. Association for Computing Machinery.

[33] DL Parnas and J Madey. Functional documentation for computer systems engineering, vol. 2. *McMaster University, Hamilton, Ontario, Technical Report CRL*, 237, 1991.

[34] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[35] Miquel Ramírez, Nitin Yadav, and Sebastian Sardiña. Behavior Composition as Fully Observable Non-Deterministic Planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, ICAPS, 2013.

[36] K Ryan and S Greenspan. Requirements engineering group report. In *Succeedings of IWSSD8-8th Intl. Workshop on Software Specification and Design, ACM Software Engineering Notes*, pages 22–25, 1996.

[37] Leonid Ryzhyk and Adam Walker. Developing a practical reactive synthesis tool: Experience and lessons learned. *Electronic Proceedings in Theoretical Computer Science*, 229:84–99, 11 2016.

[38] Sebastian Sardiña and Nicolás D'Ippolito. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence,*

IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pages 3200–3206. AAAI Press, 2015.

[39] Frank Strobl and Alexander Wissp eintner. Specification of an elevator control system. Technical report, Institut Fur Informatik, Technische Universitat TUM, 1999.

[40] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[41] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing, 1st edition, 2009.

[42] Christopher Weber and Daniel Bryce. Planning and Acting in Incomplete Domains. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011.

[43] Yechuan Xia, Jianwen Li, Shengping Xiao, Weikai Miao, and Geguang Pu. Identifying boundary conditions with the syntax and semantic information of goals. https://arxiv.org/abs/2203.12903, 2022.

[44] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, jan 1997.