Risk-Driven Revision of Requirements Models

Dalal Alrajeh* Axel van Lamsweerde† Jeff Kramer* Alessandra Russo* Sebastian Uchitel*‡

*Department of Computing, Imperial College London, UK
{da04, jk, ar3, su2}@doc.ic.ac.uk

†ICTEAM, Université catholique de Louvain, Belgium
avl@info.ucl.ac.be

†Departamento de Computación, Universidad de Buenos Aires and CONICET, Argentina

ABSTRACT

Requirements incompleteness is often the result of unanticipated adverse conditions which prevent the software and its environment from behaving as expected. These conditions represent risks that can cause severe software failures. The identification and resolution of such risks is therefore a crucial step towards requirements completeness. Obstacle analysis is a goal-driven form of risk analysis that aims at detecting missing conditions that can obstruct goals from being satisfied in a given domain, and resolving them.

This paper proposes an approach for automatically revising goals that may be under-specified or (partially) wrong to resolve obstructions in a given domain. The approach deploys a learning-based revision methodology in which obstructed goals in a goal model are iteratively revised from traces exemplifying obstruction and non-obstruction occurrences. Our revision methodology computes domain-consistent, obstruction-free revisions that are automatically propagated to other goals in the model in order to preserve the correctness of goal models whilst guaranteeing minimal change to the original model. We present the formal foundations of our learning-based approach, and show that it preserves the properties of our formal framework. We validate it against the benchmarking case study of the London Ambulance Service.

Categories and Subject Descriptors

H.1.3.1 [**Software and its engineering**]: Software creation and management—*Designing software, Requirements analysis*

Keywords

Requirements completeness; obstacle analysis; goal-oriented requirements engineering; inductive learning; theory revision.

1. INTRODUCTION

Requirements completeness is a major concern in requirements engineering. Incompleteness commonly arises from a lack of anticipation of adverse conditions which prevent the software and its environment from behaving as expected: no specific requirements are engineered for such cases. These unanticipated conditions represent risk, and may be the cause of severe software failures [35]. Risks may cover a wide range of undesirable situations, such as safety hazards, security threats and data inaccuracies amongst others. The elicitation and analysis of these risks is at the heart of the requirements engineering process [36, 16, 8, 26].

Obstacle analysis is a model-based, goal-oriented form of analysis [35, 27], that focuses on a particular type of domain-specific risk: the non-satisfaction of desired goals due to events or conditions present within a domain and obstructing them. In this context, an obstacle to a goal is defined as a precondition for the non-satisfaction of that goal. Obstacle analysis comprises three steps [35]: (1) identification of obstacles to goals from previously elicited goals and domain properties; (2) quantitative assessment of the likelihood and criticality of those obstacles, in terms of severity of their consequences; and (3) resolution of likely and critical obstacles through countermeasures to be integrated in the goal refinement graph (hereafter referred to as the system's *goal model*), from which the software requirements are derived.

A number of approaches have been proposed to handle the identification (e.g., [36, 6]) and the assessment (e.g., [12]) steps of obstacle analysis. However, the systematic resolution of the identified and assessed obstacles towards a complete and robust goal model remains an open challenge.

Goal revision is a particular class of strategies for obstacle resolution — others include obstacle prevention and obstacle reduction. Goal revision typically involves modifying a goal model in order to eliminate obstructions. It covers the goal substitution and goal weakening strategies [36]. Given a goal model, a set of domain properties and an obstacle to the goal in the goal model, a goal revision strategy consists of finding *safe* revisions of the obstructed goal along with a set of propagated changes in the goal model, such that (a) the obstruction caused by the obstacle is eliminated, and (b) the resulting goal model remains complete and consistent.

In this paper, we propose a formal approach to obstacle resolution. Our approach automatically and *iteratively* revises goals, that may be under-specified or (partially) wrong, to eliminate obstructions in a given domain from traces exemplifying obstruction and non-obstruction occurrences. We define a goal revision problem as a learning-based revi-

ICSE '16, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2884781.2884838

sion task [37]. The approach generates obstruction-free goal revisions and automatically propagates them to other goals in the model to make the obstruction disappear while preserving model correctness and guaranteeing minimal change on the original model. If alternative revisions are found, the approach produces alternative goal models each of which is guaranteed to be satisfiable within the given domain. It makes use of a recently developed constraint-driven learning technique [9]. This technique allows us to specify syntactic and semantic constraints over the search space of possible goal revisions. We use this feature to ensure the correctness of the revised goal model. Our approach terminates once all identified obstructions have been removed within a given set of domain properties, or when no revision can be found for this set. The latter case calls for eliciting further domain properties from which a new cycle of obstacle analysis should be applied [6].

In this paper, we assume that a set of obstructed goals is given and that the identified obstacles to these goals have been prioritized in terms of criticality in the preceding assessment phase (e.g., through cost-benefit analysis) of obstacle analysis. We do not consider obstructions arising, for instance, from uncertainty about stakeholders, their priorities or implementation costs, as in [22]. Our contribution is summarized as follows:

- a formal underpinning of the notion of *safe goal revision* for obstacle resolution;
- an automated approach for *computing* safe revision of a goal from obstruction and non-obstruction traces;
- an automated technique for minimally *propagating changes throughout a goal model* while ensuring that the resulting model is correct;
- a new application of learning-based revision and constraint-driven learning to requirements engineering.

To the best of our knowledge, this paper is the first to automatically execute and propagate goal revisions in formal goal models.

The rest of the paper is organized as follows. Sec. II introduces background on goal modelling, obstacle analysis, and revision-based learning. Sec. III motivates our approach on a small example. Sec. IV presents the formal underpinning of our approach. Sec. V and VI, define the learning-based revision technique used for computing goal revisions and summarizes our evaluation on an ambulance dispatching system [17] respectively. Related work is discussed in Sec. VII.

2. BACKGROUND

We recall some basics on behavioural goal modelling, obstacle analysis and learning-based revision.

2.1 Goal-oriented system modelling

A *goal* is a prescriptive statement of intent to be satisfied by cooperating agents forming the considered system. Agents may include devices like sensors and actuators, people, pre-existing software and the software to be developed. Unlike goals, *domain properties* are descriptive statements about the problem space, e.g., physical laws. A *behavioural* goal captures a maximal set of intended system behaviour; unlike soft goals, it is satisfiable in a clear-cut sense [35].

A behavioural goal is of type *Achieve* or *Maintain*. To enable formal analysis, behavioural goals may be specified in Fluent Linear Temporal Logic (FLTL) [18]. A goal has the general form $\Box(C \rightsquigarrow \Theta T)$ where C and T are conjunctions of fluent expressions (defined hereafter), $\rightsquigarrow$ is one of the classical logical connectives $\rightarrow$ and $\leftrightarrow$ and Θ is a real-time temporal operator such as $\bigcirc$ (next), $\Diamond$ (sometimes in the future), $\Diamond_{\sim d}$ (sometimes in the future before deadline d), $\Box$ (always) and $\mathbf{U}$ (always in the future unless) [21]. The expression $C \Rightarrow \Theta T$ is a shorthand for $\Box(C \rightarrow \Theta T)$.

A fluent f is defined by a set I_f of initiating events, a set T_f of terminating events and an initial truth value either *true* or *false*. Given a set of event labels A, called *alphabet*, a fluent definition takes the form $f = \langle I_f, T_f, Init\rangle$, where $I_f \subseteq A$, $T_f \subseteq A$ and $I_f \cap T_f = \emptyset$, and $Init \in \{true, false\}$. A negative fluent literal is a fluent preceded with $\neg$. Otherwise it is called a positive literal. The notation $(\neg)f$ is a shorthand for either f or $\neg f$. A *fluent expression* is a fluent literal preceded by a temporal operator.

FLTL expressions are interpreted over sequences of state and events called *traces*. Events may occur simultaneously in a trace. We write $\sigma = s_0\{a, b\}_0 s_1\{c\}_1 s_2...$ to represent a trace in which the events a and b occur simultaneously from state s_0 followed by the event c from s_1. We sometimes write traces as $\sigma = 0 : \{a, b\}, 1 : \{c\}, 2 : ....$ Given a trace σ and a set of fluent definitions FD, a fluent is said to be *true* (resp. *false*) in a trace σ at position i, denoted $\sigma, i \models f$, if and only if either of the following conditions holds: (a) the fluent is initially *true* and no terminating event has occurred since, or (b) some initiating event has occurred before i with no terminating event occurring since then. The notation $\sigma \models f$ is a shorthand for $\sigma, 0 \models f$. The semantics of fluent expression are defined inductively. Given two FLTL expressions Φ and Ψ, $\{\Phi\} \models \Psi$ holds if all traces satisfying Φ also satisfy Ψ.

Given a trace σ and alphabet A, the projection of σ over A, denoted as $\sigma|_A$, is the trace obtained by eliminating from σ all elements that are not in A. We use the notation $\Sigma(\Phi)$ (resp. $\Sigma_A(\Phi)$) to denote all traces (resp. projections) satisfying Φ.

A *goal model* is a refinement graph in which each goal is represented as a node and associated with a unique label. An AND-refinement of a goal PG into a set of sub-goals $\{SG_1...., SG_n\}$ is *complete* when the sub-goals $\{SG_i\}$, possibly with domain properties in D, are together sufficient for satisfying PG; and it is consistent if the sub-goals are all together consistent with the domain properties. These properties are more precisely stated as follows, for all $1 < i < n$:

$\{SG_1, ..., SG_n, D\} \models PG$ (*complete refinement*)
$\{SG_1, ..., SG_n, D\} \not\models false$ (*consistent refinement*)

A goal model is said to be *correct* if every AND-refinement relation in the model is complete and consistent. In this paper, we define a goal model as a tuple $\langle \Gamma, \prec\rangle$ where Γ is the set of goals appearing in the model and $\prec$ is the set of AND-refinement relations between goals in Γ. We will sometimes write $\{SG_1...., SG_n\} \prec G$ to denote the notion that the set $\{SG_1...., SG_n\}$ is a refinement of G.

Obstacle analysis is a goal-oriented form of risk analysis. An *obstacle* to a goal is a domain-satisfiable precondition for the non-satisfaction of this goal [36, 35]:

$\{O, D\} \models \neg G$ (*obstruction*)
$\{O, D\} \not\models false$ (*domain consistency*)

We call any trace σ such that $\sigma \models D \wedge O$ an *obstruction trace* for goal G with respect to obstacle O in D. Similarly, we refer to a trace where $\sigma \models D \wedge \neg O$ as a *non-obstruction trace* for G with respect to O in D.

2.2 Inductive Learning and Theory Revision

Inductive Logic Programming (ILP) [29] is a logic-based inductive learning [28] technique that is concerned with learning generalizations H that, together with a given background knowledge B explain a given set E of examples, where H, B, E are represented as logic programs [25]. Typically, ILP algorithms generate minimal hypotheses, i.e., ones that are minimal both in terms of their size and number. Most recent state-of-the-art inductive learning systems [14, 15, 10], referred to as *non-monotonic ILP* systems, are capable of not only learning generalizations of given examples, but also revisions of existing knowledge. Assuming K to represent possibly incorrect knowledge (referred to as the revisable knowledge), a *learning-based revision* task is a computational task that consists of learning changes C to be applied to the revisable knowledge K, to generate a revised knowledge $K' = \rho(K, C)$ that entails the examples. The function ρ denotes the application of revisions C to the revisable knowledge K.

The search for revisions is performed within the scope of a search space s defined by a set of *mode declarations* (MD), a form of language bias that specifies the syntactic form of acceptable changes to be learned for a given revision task. We use $s(MD)$ to denote this space. In general, such search space is very large as it potentially includes all possible additions and deletions that could be applied to the revisable knowledge within the scope of a given language, some of which may be irrelevant for particular domains. *Inductive learning through constraint-driven bias* is a technique recently proposed to tackle this problem [9]. *Acceptable changes* are the set of possible additions and deletion to the revisable knowledge that conform to a given set of domain-specific constraints. This is formally defined as follow, where $\mathcal{P}$ denotes the operator of power set, $\mathcal{P}(s(MD))$ denote the set of possible changes and $\mathcal{P}(s(MD))]_{IC}$ the set of changes compatible with a given set of constraints IC.

Definition 1 (Constraint-driven revision). *A* learning-based revision *task through constraint-driven bias is a tuple $\langle B, K, E, MD, IC_{MD}\rangle$, where B denotes the background knowledge, K represents the revisable knowledge, E is the set of examples, MD is the language bias and IC_{MD} is the set of constraints over the search space $\mathcal{P}(s(MD))$ induced by MD. H is an* acceptable learned revision *for this task if and only if $H \in [\mathcal{P}(s(MD))]_{IC}$ and the revised knowledge $K' = \rho(K, H)$ satisfies the condition $B \cup K' \models \bigwedge_{e \in E} e$.*

Minimality in the constraint-driven revision case is determined by the number of of additions and deletions applied to K to satisfy the conditions above.

3. MOTIVATING EXAMPLE

Consider a simplified version of a car automatic handbrake control system [35] in which a car driver, the car's engine, and the handbrake must interact to achieve safe brake control. A high-level goal for this system requires "the handbrake to be released within three seconds of the driver's wanting to start the car" expressed in FLTL as

$$G_r = \mathit{DriverWantsToStart} \Leftrightarrow \Diamond_{\leq 3}\ \mathit{HandbrakeReleased}$$

Fig. 1.a shows a goal model for this root goal. In brief, the goal labelled G_r is refined into three sub-goals: a child goal labelled G_1 stating that "the driver presses the acceleration pedal whenever he/she wants to drive the car", a child goal labelled G_2 prescribing that "the motor revs increase after the pedal is pressed" and a child G_3 requiring that "once the motor revs increase (i.e., revving), the handbrake is to be released". Each of these are then refined into two leaf goals capturing the if and only-if part of $\Leftrightarrow$.

Suppose the domain properties D contain the following fluent definitions and a property that states that the motor revs increase whenever the air-conditioning in the car starts:

$$\begin{aligned}
&\mathit{AirConditioningStarts} \equiv \langle \mathit{startAC}, \mathit{stopAC}, \mathit{false}\rangle\\
&\mathit{MotorRevving} \equiv \langle \mathit{incRevMotor}, \mathit{decRevMotor}, \mathit{false}\rangle\\
&\mathit{DriverWantsToStart} \equiv \langle \mathit{insertKeys}, \mathit{removeKeys}, \mathit{false}\rangle\\
&\mathit{HandbrakeReleased} \equiv \langle \mathit{releaseHandbrake}\\
&\qquad\qquad \mathit{engageHandbrake}, \mathit{false}\rangle\\
&\mathit{ThrottlePedalPressed} \equiv \langle \mathit{pressPedal}, \mathit{releasePedal}, \mathit{false}\rangle
\end{aligned}$$

$$\mathit{AirConditioningStarts} \Rightarrow \bigcirc\ \mathit{MotorRevving}$$

The goal model in Fig. 1 fails to consider situations in which the motor revs increase due to other reasons: namely the air-conditioning starting, a possible behaviour within the domain that would obstruct the goal G_7 stating that "the acceleration pedal is pressed if the motor is revving", $\bigcirc\ \mathit{MotorRevving} \Rightarrow \mathit{ThrottlePedalPressed}$.

Using [6], the following obstacle condition is generated:

$$O_{G_7} = \Diamond(\mathit{AirConditioningStarts} \wedge \neg\, \mathit{ThrottlePedalPressed})$$

Although approaches such as [36] provide a list of strategies that may be deployed to resolve obstacles like this, they do not discuss when these strategies are applicable and in which cases nor how such strategies can be applied individually or in tandem. Furthermore, they do not provide support for ensuring that resolutions are propagated correctly within the entire goal model.

Consider for instance, the substitution strategy in which the obstructed goal G_7 may be replaced with a new one which no longer relies on the acceleration pedal being pressed. One choice would be to weaken the goal to become

$$\bigcirc\ \mathit{MotorRevving} \Rightarrow (\mathit{AirConditioningStarts} \vee\ \mathit{ThrottlePedalPressed})$$

Another possibility is to extend the domain description D with new fluents and properties that may allow alternative resolutions. For example, suppose D is extended to model a car with an Electronic Handbrake [19] that has an electronic handbrake button and a handbrake controller that adds tension to the brake cables: $\mathit{ManualBrakeButtonOff} \equiv \langle \mathit{switchButtonOff}, \mathit{switchButtonOff}, \mathit{false}\rangle$; $\mathit{HandBrakeCtrlOff} \equiv \langle \mathit{switchCtrlOff}, \mathit{switchCtrlOn}, \mathit{false}\rangle$.

In this extended domain, it is possible to consider a revised goal in which an electronic handbrake disables the handbrake controller.

$$\bigcirc\ \mathit{HandBrakeCtrlOff} \Rightarrow \mathit{ManualBrakeButtonOff}$$

Whichever choice is made, simply replacing the obstructed goal G_7 is clearly insufficient since the resulting refinements in the goal model may no longer be complete. Hence, the changes to the goal need to be correctly propagated to other goals in the model that were dependent on the previous goal

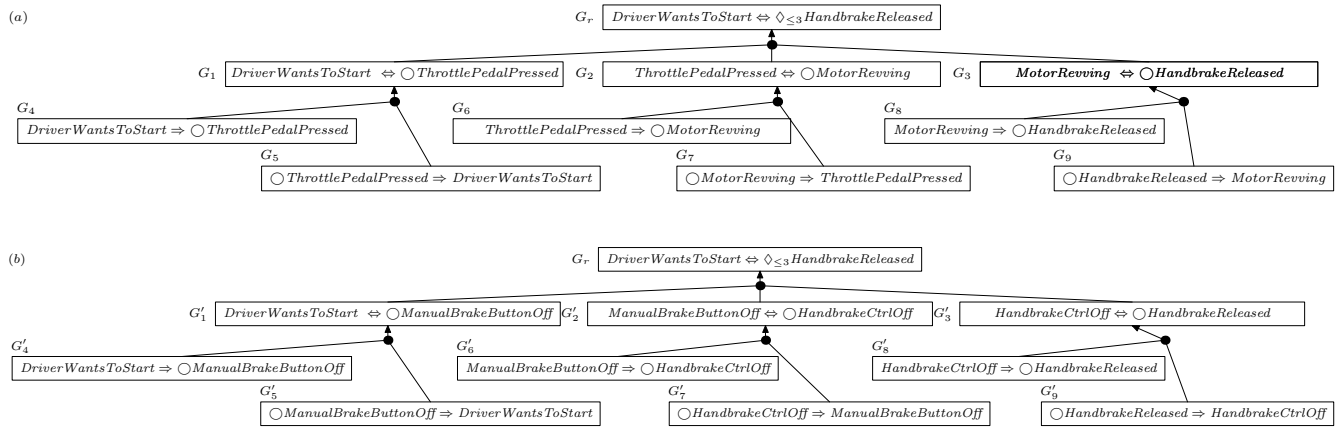

Figure 1: Goal model for Automatic Handbrake Control System: (a) original, (b) revised.

being satisfied. As a result of the propagation, the handbrake controller will release (or apply) the handbrake rather than the motor revving.

We present a learning-based revision technique that automatically produces the revised goal model in Fig. 1.b by substituting the obstructed goal G_7 and propagating the necessary changes to the rest of the model to guarantee it to be obstruction-free, consistent within the domain and preserve the correctness of the refinements in the goal model.

4. GOAL REVISION SETTING

As mentioned in Sec. 1, different revision strategies may be deployed to resolve an obstacle for a given goal, each of which involves a series of strategy-specific manipulations to the domain properties, and/or goal model. In spite of the specific strategy deployed, the goal model resulting from the resolution must satisfy a number of requirements to ensure that the obstacle is eliminated. In this section we discuss common requirements goal revision strategies should meet.

In what follows we use D' to represent the domain properties resulting from updates to the domain properties D, G' a revision to the goal G, M' the goal model resulting from integrating G' in the model M and propagating changes through the original goal model M. Finally ζ denotes a function that returns a user-defined distance metric that measures the difference between two goals (i.e., $\zeta(G, G')$), or sets of goals (i.e., $\zeta(\{G_i\},\{G'_i\})$).

The first requirement is concerned with domain-consistency. Since goals are expected to be achieved within specific domains, any revision strategy must ensure that it produces satisfiable revisions within that domain.

REQ1 [Revision Consistency] *If goal G is obstructed by obstacle O in domain D, then any goal revision G' should be consistent with D', i.e., $\{D', G'\} \not\models false$.*

The second requirement is concerned with obstacle elimination. Obstacle elimination may be achieved in two ways: either by ensuring the obstruction disappears (i.e., violating the obstruction condition in Subsection. 2.1) or making the obstacle inconsistent with the domain (i.e., violating the domain-consistency condition in Subsection. 2.1). In the first case, a minimum requirement for the resolution strategy is to ensure that the revised goal is no longer violated by the obstacle's occurrence.

REQ2 [Obstacle Elimination] *If goal G is obstructed by obstacle O in D, then any goal revision G' should no longer be obstructed by O in D', i.e. $\{D', O\} \not\models \neg G'$.*

Furthermore, the goal changes to be propagated throughout the model should not cause other changed goals to be obstructed by the obstacle O.

REQ3 [Non-obstruction Sustainability] *The propagation of changes to G through M should not reintroduce new obstructions, by obstacle O, to any changed goal G'_i of M' in D', i.e., $\{D', O\} \not\models \neg G'_i$.*

In addition, such change propagation should preserve the goal model's correctness.

REQ4 [Refinement Correctness] *The propagation of changes to G through M must ensure that every goal refinement in the model M' is a consistent and complete refinement.*

We call a goal revision *safe* if it satisfies REQ1–4. In addition, two other requirements appear desirable. Any behaviour that was permissible in the domain and in which the obstacle did not occur should be permissible after the revision in the updated domain. Assuming A_D denotes the alphabet of the domain D, prior to any update. This requirement is more precisely stated below.

REQ5 [Behaviour Preservation] *The revised goal model M' should preserve those behaviour from G in which the obstacle does not occur, i.e., $\Sigma(D \wedge \neg O \wedge G) \subseteq \Sigma_{A_D}(D' \wedge G')$.*

Finally, the revised goal model M' should be as close as possible to the original model M.

REQ6 [Minimal Change] *The propagation of the goal revision G' through M should be minimal, i.e., $\zeta(M, M')$ is minimal.*

5. OBSTACLE-DRIVEN GOAL REVISION

This section presents our approach for learning-based revision for computing safe goal revisions.

5.1 Approach Overview

The input is a goal model M, updated domain properties D, which include fluent definitions, and a set of detected obstacles $\{O_i\}$ to goals in M. We assume a set of obstacles have already been detected, for instance using the procedure presented in [6], and assessed against their likelihood and criticality using the method in [12]. Then for each ob-

stacle O to G of M in D, a finite set of obstruction traces Σ_O are obtained from a model satisfying D and O. We generate these by verifying $\{D, O\} \models \Box(C \rightsquigarrow \Theta T)$, in the spirit of [5]. To achieve the behaviour preservation requirement expressed in Sec. 4, the approach further considers a finite set of non-obstruction traces $\Sigma_{\neg O}$ that satisfy the domain and negation of the obstacle. These are generated by verifying a model of D and $\neg O$ against the obstructed goal's 'anti-target', i.e., $\{D, \neg O\} \models \Box(C \rightsquigarrow \neg\Theta T)$, as explained in [6]. We use the model checking approach described in [34] for automatically generating the traces. However, the approach is not bound to a specific verification method and assumes that domain-consistent traces are given.

Then, a learning-based revision task is constructed which aims at automatically finding revisions G' to the obstructed goal G such that the revised goal is satisfied in traces by all in $(\Sigma_{\neg O} \cup \Sigma_O)$ and is consistent with D. The revisions are automatically propagated to other goals in M. Where alternative revisions exist, alternative goal models are generated. The output is a set of alternative goal models $\{M'_j\}$ that are guaranteed to be satisfied in the traces generated. The revision procedure is repeated with respect to the revised goal until no traces in $\Sigma(D \wedge O)$ obstruct the revised goal or until no revision can be found within the given domain properties, indicating that the domain needs to be updated. In the latter case, new obstacles (if any) need to be identified and assessed within the updated domain in order to start a new revision cycle. The revision process is iteratively applied on every input obstacle in $\{O_i\}$. The approach terminates once a goal model is reached that meets the REQ1–6 of Sec. 4 for every O_i.

The order in which obstacles are eliminated are assumed to be determined by the engineer depending on their assessed criticality. In our illustrations, we assume that the structure of the goal model is maintained, i.e., the number of children and refinements for a parent goal does not change. We illustrate the proposed approach by considering a goal substitution strategy. Weakening is discussed in Sec. 6.

To enable the learning-based revision in the context of obstacle resolution, the task of finding a safe goal revisions must be defined in in terms of the ILP framework introduced in Sec. 2.2. The goal model, domain properties and traces are therefore automatically translated into a logic program, and together with appropriate constraints over the solution space, learning-based revision computes acceptable revisions as shown in the next subsection. Though the details are provided, the encoding and procedure for generating revisions are black-box computations from the engineer's perspective.

5.2 Encoding for Obstacle Elimination

We first introduce some terminologies used in our description. We assume that every goal G is associated with a label that uniquely identifies it, denoted . We refer to a goal SG in a refinement set of a parent goal PG as a *left child* if a fluent literal that appears in SG's current condition also appears in PG's current condition. Similarly, we call it a *right child* if a fluent literal that appears in SG's target condition also appears in PG's target condition. In addition, we say that the child goal SG_1 is *left* (resp. *right*) *of* another child SG_2 if a fluent literal that appears in SG_1's target condition (resp. current condition) also appears in the current (resp. target) condition of SG_2. We describe below the encoding of goals, the structure of the goal model, the traces and the setting for revision task. The encoding of domain properties is similar to that described in [6].[1]

The formalism used for encoding the input is based on the Event Calculus (EC) [20]. In addition to the standard EC predicates (e.g., `holds_at` and `happens`), we extend the language to include auxiliary predicates that capture the formulation of goals' current and target conditions together with their interpretation over traces.

For a given goal model $M = \langle \Gamma, \prec \rangle$, the encoding M is defined in terms of the encoding of the goal expressions in it (denoted $tr[\Gamma]$) and the refinement relations it contains (denoted $tr[\prec]$). The $tr[\Gamma]$ makes use of predicates like `c_holds` (meaning the current condition holds), `t_holds` (for the target condition holds), `t_holds_next` (meaning the target condition holds at the next time-point), and the predicate `t_holds_eventually` (meaning the target condition holds eventually), etc. The current condition C of a goal labelled l_G is automatically translated into the expression:

$$\texttt{c\_holds}(l_G, \texttt{I}, \texttt{S}) \leftarrow (\texttt{neg\_})pred_b[\phi_1, \texttt{I2}, \texttt{S}] \wedge ... \wedge (\texttt{neg\_})pred_b[\phi_n, \texttt{I2}, \texttt{S}]$$

where ϕ_j is a fluent expression in G's current condition, $pred_b[\phi_j, \texttt{I2}, \texttt{S}]$ is the EC literal corresponding to the fluent expression ϕ_j as given in Table 1 and the variables `I` and `S` are of type time-points and traces respectively. The encoding of target conditions results in rules of the form:

$$pred_h[\Theta T, l_G, \texttt{I}, \texttt{S}] \leftarrow pred_b[\Theta, \texttt{I}] \wedge (\texttt{neg\_})pred_b[\phi_1, \texttt{I2}, \texttt{S}] \wedge ... \wedge (\texttt{neg\_})pred_b[\phi_n, \texttt{I2}, \texttt{S}]$$

where $pred_b[\Theta, \texttt{I}]$ is the literal corresponding to Θ in Table 1, and $pred_h[\Theta T, l_G, \texttt{I}, \texttt{S}]$ is the literal corresponding to ΘT.

Table 1: Encoding of goals.

Fluent expression	EC encoding
$pred_b[(\neg)f, \texttt{I}, \texttt{S}]$	(neg_)holds_at(f,I,S)
$pred_b[\bigcirc(\neg)f, \texttt{I}, \texttt{S}]$	(neg_)holds_at_next(f,I,S)
$pred_b[\Diamond(\neg)f, \texttt{I}, \texttt{S}]$	(neg_)holds_at_eventually(f,I,S)
$pred_b[\Diamond_{\leq d}(\neg)f, \texttt{I}, \texttt{S}]$	(neg_)holds_at_eventually_by(f, d, I,S)
$pred_b[\emptyset, \texttt{I}]$	I=I2
$pred_b[\bigcirc, \texttt{I}]$	next(I,I2)
$pred_b[\Diamond, \texttt{I}]$	eventually(I,I2)
$pred_b[\Diamond_{\leq d}, \texttt{I}]$	eventually_by(I,d, I2),
$pred_h[T, l_G, \texttt{I2}, \texttt{S}]$	t_holds(l_G,I2,S)
$pred_h[\bigcirc T, l_G, \texttt{I2}, \texttt{S}]$	t_holds_next(l_G,I2,S)
$pred_h[\Diamond T, l_G, \texttt{I2}, \texttt{S}]$	t_holds_eventually(l_G,I2,S)
$pred_h[\Diamond_{\leq d} T, l_G, \texttt{I2}, \texttt{S}]$	t_holds_eventually_by(l_G,d, I2,S)

The following is an extract of the encoding obtained for the goals labelled G_r and G_7 of Fig. 1.

```
c_holds(gr, I, S) ←
    holds_at(driverWantsToStart, I, S).
t_holds_eventually_by(gr, 3, I, S) ←
    holds_at_eventually_by(handbrakeReleased, 3, I, S).
t_holds_eventually_by(gr, 3, I, S) ←
    holds_at_eventually_by(handbrakeReleased, 3, I, S).
c_holds(g7, I, S) ←
    holds_at_next(motorRevving, I, S).
t_holds(g7, I, S) ←
    holds_at(throttlePedalPressed, I, S).
```

[1]For details, see www.doc.ac.uk/∼da04/icse16/.

In addition to the above, $tr[\Gamma]$ contains rules that capture notions of vacuous and non-vacuous satisfactions of goals in traces, using predicates `holds_vacuously` and `holds_non-vacuously`. This is to preserve the semantics of the goal expressions over traces in the semantics of the logic programs (i.e., both when the goal condition holds and does not hold). Furthermore, since our logic programs have finite models, we extend the satisfaction notion of goals in finite traces to handle cases where the current condition of a goal holds, and the bound on traces is reached before the goal's target condition is satisfied. This is formalized using the predicate `holds_at_end_time`. Below is the encoding generated from the goals G_r and G_7.

$$\texttt{holds\_non\_vacuously}(\texttt{gr},\texttt{I},\texttt{S}) \leftarrow$$
$$\quad \texttt{c\_holds}(\texttt{gr},\texttt{I},\texttt{S}) \ \wedge \ \texttt{t\_holds\_eventually\_by}(\texttt{gr},3,\texttt{I},\texttt{S}).$$
$$\texttt{holds\_vacuously}(\texttt{gr},\texttt{I},\texttt{S}) \leftarrow$$
$$\quad \texttt{not c\_holds}(\texttt{gr},\texttt{I},\texttt{S}) \wedge$$
$$\quad \texttt{not t\_holds\_eventually\_by}(\texttt{gr},3,\texttt{I},\texttt{S}).$$
$$\texttt{holds\_at\_end\_time}(\texttt{gr},\texttt{I},\texttt{S}) \leftarrow$$
$$\quad \texttt{c\_holds}(\texttt{gr},\texttt{I},\texttt{S}) \wedge$$
$$\quad \texttt{eot}(\texttt{E}) \ \wedge \ \texttt{I} \leq \texttt{E} \ \wedge \ \texttt{E} < \texttt{I}+3,$$
$$\quad \texttt{not t\_holds\_eventually\_by}(\texttt{gr},\texttt{E},\texttt{I},\texttt{S}).$$
$$\texttt{holds\_non\_vacuously}(\texttt{g7},\texttt{I},\texttt{S}) \leftarrow$$
$$\quad \texttt{c\_holds}(\texttt{g7},\texttt{I},\texttt{S}) \ \wedge \ \texttt{t\_holds}(\texttt{gr},\texttt{I},\texttt{S}).$$
$$\texttt{holds\_vacuously}(\texttt{g7},\texttt{I},\texttt{S}) \leftarrow \texttt{not c\_holds}(\texttt{g7},\texttt{I},\texttt{S}).$$

The encoding of refinement relations $tr[\prec]$ makes use of the predicates `root`/1, `left_child`/2, `right_child`/2 and `left_of`/2 to formulate the refinement relation between goals in the goal model. For example, the following facts are generated for our running example.

`root(gr).` `left_child(gr, g1).`
`child_of(gr, g2).` `right_child(gr, g3).`

Our encoding also captures domain-independent axioms, denoted Ax, that define laws of persistency of fluents over time in traces [20], and of goal satisfaction, including:

$$\texttt{holds}(\texttt{G},\texttt{I},\texttt{S}) \leftarrow \texttt{holds\_non\_vacuously}(\texttt{G},\texttt{I},\texttt{S}).$$
$$\texttt{holds}(\texttt{G},\texttt{I},\texttt{S}) \leftarrow \texttt{holds\_vacuously}(\texttt{G},\texttt{I},\texttt{S}).$$
$$\texttt{holds}(\texttt{G},\texttt{I},\texttt{S}) \leftarrow \texttt{holds\_at\_end\_time}(\texttt{G},\texttt{I},\texttt{S}).$$

To enable the revision, the encoding formulates the notion of obstruction and obstacle elimination in Ax:

$$\texttt{obstacle\_holds}(\texttt{G},\texttt{I},\texttt{S}) \leftarrow \texttt{not holds}(\texttt{G},\texttt{I},\texttt{S}).$$
$$\texttt{obstructed} \leftarrow \texttt{obstacle\_holds}(\texttt{G},\texttt{I},\texttt{S})$$
$$\texttt{entailed} \leftarrow \texttt{not obstructed}.$$

The first rule says that an obstacle for a goal G holds at some time-point in a trace if the goal does not hold at that time-point. The second says that an obstruction occurs if there is a goal for which an obstacle holds. No obstruction gives entailment of all goal.

The encoding of traces $tr[\Sigma_O \cup \Sigma_{\neg O}]$ on the other hand generates a set of `happens` facts. For instance, from the obstruction trace $\sigma_{O_{G_7}} = 0$: {*startCar*}, 1:{*startAC*,*switchButtonOff*}, 2:{*incRevMotor*}, {*releaseHandbrake*}, and the non-obstruc-tion trace $\sigma_{\neg O_{G_7}} = 0$: {*startCar*}, 1:{*pressPedal*, *switchButtonOff*}, 2:{ *incRevMotor*}, {*releaseHandbrake*} we have the following facts:

$tr[\sigma_{O_{G_7}} \cup \sigma_{\neg O_{G_7}}] =$
{`happens(startCar, 0, s1)`, `happens(startAC, 1, s1)`,
`happens(switchButtonOff, 1, s1)`, `happens(incRevMotor, 2, s1)`,
`happens(switchButtonOff, 2, s1)`,
`happens(releaseHandbrake, 3, s1)`,
`happens(startCar, 0, s2)`, `happens(switchButtonOff, 1, s2)`,
`happens(pressPedal, 1, s2)`, `happens(incRevMotor, 2, s2)`,
`happens(switchButtonOff, 2, s2)`,
`happens(releaseHandbrake, 3, s2)`}.

where `s1` and `s2` are unique identifiers for encoded traces.

5.3 Computing Goal Revisions

To compute revisions to the obstructed goal, the set of examples E must be specified. For the learner, the example are facts that should follow from the background knowledge and revisable theory but as such do not. In the revision problem, we consider our examples to represent the fact that all encoded goals (including the obstructed one) should hold at all time-points and in all the traces, including the obstruction traces. Given the rules defining entailment in Sec. 5.2, E in its simplest form includes the single fact `entailed`.

The mode declaration MD is defined to cover all fluents expressions that may be constructed using the language of the domain and the predicates in Table 1. As these are typically domain-specific, the engineer may specify which fluents may be considered in the revision process, thus limiting the solution space to goals expressible using these.

To ensure the refinement correctness requirement (REQ4), we define a set of the constraints IC_{MD} over the solution space, requiring that any change to the obstructed goal forces, where necessary, the revision to be applied to other goals within the model. The constraints specify which changes to goals' current and target conditions are acceptable to ensure the completeness and consistency of refinements. For instance, it is required that any changes to the current condition of a left child are reflected correctly in the current condition of its parent. This is captured in the following constraints in IC_{MD}.

$$\leftarrow \texttt{in}(\texttt{R}_1,\texttt{current}(\texttt{PG}),\texttt{L}),\texttt{left\_child}(\texttt{PG},\texttt{SG}) \wedge$$
$$\quad \texttt{not in}(\texttt{R}_2,\texttt{current}(\texttt{SG}),\texttt{L}).$$
$$\leftarrow \texttt{in}(\texttt{R}_1,\texttt{current}(\texttt{SG}),\texttt{L}),\texttt{left\_child}(\texttt{PG},\texttt{SG}) \wedge$$
$$\quad \texttt{not in}(\texttt{R}_2,\texttt{current}(\texttt{PG}),\texttt{L}).$$

where the variable $\texttt{R}_i$ is a unique identifier (used by the learner) for a rule within the solution space, and `L` corresponds to an EC encoding of a fluent expression. The literal $\texttt{in}(\texttt{R}_1,\texttt{current}(\texttt{PG}),\texttt{L})$ for instance means the rule with identifier $\texttt{R}_1$ representing PG's current condition (i.e., `c_holds` in the head) has the EC encoding of the fluent expression `L` in its body. Although these constraints are pre-specified, the approach provides a language for users to specify other forms of constraints that are best suited for the domain. The revision task can now be defined as follows.

Definition 2 (Goal revision task). *A goal revision task is a learning-based revision task* $\langle B, K, E, MD, IC_{MD}\rangle$, *where* $K = tr[\Gamma]$, $B = tr[\prec] \cup tr[D] \cup tr[\Sigma_O \cup \Sigma_{\neg O}] \cup Ax$, $E = \{\texttt{entailed}\}$.

The encoding of the problem preserves the obstruction to goals, prior to the revision computation. For instance, given the trace $tr[\sigma_{O_{G_7}}]$, the current target of the goal G_7 holds at time-point 2 (`holds_at_next(motorRevving, 2, s1)`)

and consequently `c_ holds(g7, 2, s1)` also holds. However, `holds_at(throttlePe-dalPressed, 2, s1)` is not true at time-point 2 and thus G_7's target condition does not hold at time-point 2 (i.e., $\texttt{t\_holds}(\texttt{g7}, 2, \texttt{s1})$ is not true). It follows from this that `holds(g7, 2, s1)` is not true and therefore `obstructed` is also true. Hence `entailed` is not covered.

THEOREM 1 (CORRECTNESS OF ENCODING). *Let G be a goal, D the domain properties and σ_O be an obstruction trace to G in D. Let B and E be the background knowledge and example obtained from Def. 2. Then $\sigma_O \models \neg G$ iff $B \cup tr[G] \not\models$ `entailed`.*

The revision procedure operates by exploring the search space for goal revisions obtainable by deleting and adding fluent expressions to the current or target conditions of the obstructed goal, $\rho(tr[\Gamma], C)$. The choice of operations to apply and of goals to which these are applied is determined by the traces in B and the constraints IC_{MD}. For instance, to satisfy `entailed` in our running example, the literal $\texttt{holds\_at}(\texttt{throttlePedalPressed}, 2, \texttt{s1})$ is deleted from the body of the rule $\texttt{t\_holds}(\texttt{g7}, \texttt{I}, \texttt{S})$. Its substitution is identified by traversing the trace for literals that if added to the body $\texttt{t\_holds}(\texttt{g7}, \texttt{I}, \texttt{S})$ would result in the derivation of $\texttt{holds}(\texttt{g7}, 2, \texttt{s1})$. To satisfy the constraints IC_{MD}, further deletions and additions are applied to other goals.

The distance metric ζ is defined by the learning algorithm as a function that calculates the number of addition and deletion operations applied on the original model as a whole to obtain the revised model. Each application of an operation is given the score of 1. In our example above, the distance between $\zeta(M, M') = 12$ capturing the number of fluent expressions that are deleted (6), e.g., *ThrottlePedalPressed* from G_1, G_2, G_4, G_5, G_6 and G_7, and added (6). Where multiple revision candidates of similar distance exist, these are provided as alternative revisions. The learning produces the following set K' of revised rules.

$\texttt{t\_holds}(\texttt{g1}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

$\texttt{t\_holds}(\texttt{g4}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

$\texttt{c\_holds}(\texttt{g5}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

$\texttt{c\_holds}(\texttt{g2}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

$\texttt{c\_holds}(\texttt{g6}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

$\texttt{t\_holds}(\texttt{g7}, \texttt{I}, \texttt{S}) \leftarrow \texttt{holds\_at}(\texttt{manualBrakeButtonOff}, \texttt{I}, \texttt{S}).$

The output indicates that changes are required to 6 goals in Fig. 1.a in total. For instance, from the last rule above, the obstructed goal G'_7 is substituted by its revised version $\bigcirc \mathit{MotorRevving} \Rightarrow \mathit{ManualBrakeButtonOff}$. The goals shown in Fig. 1.b are computed following a second iteration of the approach to resolve the obstacle $\Diamond$ ($\mathit{manualBrakeButtonOff} \wedge \bigcirc \neg \mathit{MotorRevving}$) by substituting the dependency on the motor's revving with a hand-brake controller.

5.4 Theoretical Results

We report here on some of the results that follow from the approach. We do not provide full proofs owing to limited space.[2] The first of these is that the approach guarantees that any revision computed is consistent with the domain.

THEOREM 2 (DOMAIN CONSISTENCY). *Let G be a goal and D the domain properties. Let σ_O be an obstruction trace to G in D. Then for any computed goal revision G', we have $\{G', D\} \not\models \mathit{false}$.*

[2]See www.doc.ac.uk/∼da04/icse16/ for proofs.

The proof is is a consequence of Theorem 1 and the soundness of the learning [9]. Since obstruction traces form part of the background knowledge, the approach also ensures that every revision computed is satisfied in the obstruction trace, thus eliminating the obstacle from (at least) the encoded traces.

THEOREM 3 (OBSTRUCTION ELIMINATION). *Let G be a goal and D the domain properties. Let σ_O be an obstruction trace to G in D. Then for any computed goal revision G', we have $\sigma_O \models G'$.*

In addition, as the example requires that all considered goals to be satisfied in the obstruction traces, the approach meets the REQ3 with respect to those traces.

THEOREM 4 (NON-OBSTRUCTION SUSTAINABILITY). *Let $M = \langle \Gamma, \prec \rangle$ be a goal model and Σ_O a set of obstruction traces for $G \in \Gamma$ in $\Sigma(D \wedge O)$ and $\Sigma_{\neg O}$ a set of non-obstruction traces for G in $\Sigma(D \wedge \neg O)$. Let $M' = \langle \Gamma', \prec \rangle$ be a revised goal model computed form D, M, Σ_O and $\Sigma_{\neg O}$. Then $\forall G'_i \in \Gamma', \sigma \in \Sigma_O \cup \Sigma_{\neg O}$: $\sigma \models G'_i$.*

Another consequence of the proposed approach is that every refinement relation in the revised goal model is guaranteed to be consistent and complete. Consistency is preserved since the refinements are (at the least) satisfied in the traces provided as input. Completeness with respect to the traces provided is maintained since every trace satisfying the encoding of a goal's children satisfies the encoding of the goal. Full correctness may be guaranteed if the traces provided capture a complete characterization of the traces in D.

THEOREM 5 (REFINEMENT CORRECTNESS). *Let $M = \langle \Gamma, \prec \rangle$ be a goal model and Σ_O a set of obstruction traces for $G \in \Gamma$ in Dand $\Sigma_{\neg O}$ a set of non-obstruction traces for G in D. Then every refinement in a revised goal model $M' = \langle \Gamma', \prec \rangle$ computed form D, M, Σ_O and $\Sigma_{\neg O}$ is a consistent and complete refinement with respect to $\Sigma_O \cup \Sigma_{\neg O}$; i.e., $\forall \{SG_1, ..., SG_n\} \prec G$:*

- $\forall \sigma \in (\Sigma_O \cup \Sigma_{\neg O})$, *if* $\sigma \models \bigwedge_i SG_i$ *then* $\sigma \models G$,
- $\exists \sigma \in (\Sigma_O \cup \Sigma_{\neg O})$. $\sigma \models \bigwedge_i SG_i$

Furthermore, since non-obstruction traces, in which the original goal is satisfied, are encoded in the background knowledge, the learning guarantees that the revised goal is satisfied in these traces too, thus meeting REQ5.

THEOREM 6 (BEHAVIOUR PRESERVATION). *Let Σ_O a set of obstruction traces for G in D and $\Sigma_{\neg O}$ a set of non-obstruction traces for G in D. Then for any computed goal revision G' to G from D, Σ_O and $\Sigma_{\neg O}$ the following condition holds: $\forall \sigma \in \Sigma_{\neg O}$. If $\sigma \models G$ then $\sigma \models G'$.*

In our context, minimality of the revision is determined by the number of fluent expressions that need to be added or deleted to meet REQ1–5 with respect to the given traces. The approach guarantees that all the computed revisions are minimally different from the model prior to the revision for the given traces.

THEOREM 7 (MINIMAL CHANGE). *Let $M = \langle \Gamma, \prec \rangle$ be a goal model, Σ_O a set of obstruction traces for $G \in \Gamma$ in $\Sigma(D \wedge O)$. Let $M' = \langle \Gamma', \prec \rangle$ be a computed goal model revision. Then $\zeta(\Gamma, \Gamma')$ is minimal with respect to Σ_O.*

Our approach also guarantees progress towards fewer obstructions in the goal model that are caused by the detected obstacles. This is easy to show since each revised goal is satisfiable in at least one more trace, i.e., the obstruction trace. The proof is based on the assumption that the revision process is performed iteratively and that obstruction traces from previous steps are used in subsequent iterations.

THEOREM 8 (PROGRESS). *Let Σ_O be a set of obstruction traces for G in D. Then for any computed goal revision G' to G from D, Σ_O and G: $|\Sigma(D \wedge O \wedge G')| < |\Sigma(D \wedge O \wedge G)|$.*

6. VALIDATION

We apply our approach to a benchmark case study, the London Ambulance Service originally reported in [36]. We demonstrate its ability to apply both weakening and substitution strategies for goal revisions expressed in first-order FLTL. To facilitate comparison with other approaches, we consider goals and obstacles that have been discussed in [36]. A partial goal model is outlined below. The root goal *AmbulanceInterventionWhenIncidentReported* states that "an ambulance must intervene within 11 minutes of an incident being reported". Refinements of this goal are represented by the level of indentation. The notation $\leftarrow_{left}$ (reps. $\leftarrow_{right}$) indicates that the goal is a left (reps. right) child goal to the goal one refinement level above.

AmbulanceInterventionWhenIncidentReported
- $\leftarrow_{left}$ *AmbulanceAllocatedWhenIncidentReported*
- $\leftarrow_{right}$ *AmbulanceInterventionWhenAllocated*
  - $\leftarrow_{left}$ *AmbulanceMobilizedWhenAllocated*
    - $\leftarrow_{left}$ *AmbulanceMobilizedWhenOnRoad*
    - $\leftarrow_{right}$ *AmbulanceMobilizedWhenAtStation*
      - $\leftarrow_{left}$ *MobilizationOrderIssuedAtStation*
      - $\leftarrow_{right}$ *AmbulanceMobilizedFromMobOrder*
  - $\leftarrow_{right}$ *AmbulanceInterventionWhenMobilized*

The domain properties D contains the following fluent definitions.

Resolved(inc:Incident) $\equiv \langle$*resolve(inc), happens(inc),false*$\rangle$
Intervention(a:Ambulance)(inc:Incident)$\equiv$
$\langle$*intervene(a,inc),not_intervene(a,inc),false*$\rangle$
Allocated(a:Ambulance)(inc:Incident)$\equiv$
$\langle$*allocate(a,inc),deallocate(a,inc), false*$\rangle$
Mobilized(a:Ambulance)(inc:Incident)$\equiv$
$\langle$*mobilize(a,inc),demobilize(a,inc),false*$\rangle$
Available(a:Ambulance) $\equiv$
$\langle$*assign(a), free(a), true*$\rangle$
AtStation(a:Ambulance) $\equiv$
$\langle$*arriveAtStation(a), leaveStation(a), true*$\rangle$
MobOrderIssued(a:Ambulance)(inc:Incident)$\equiv$
$\langle$*issueMobOrder(a,inc), retractMobOrder(a,inc), true*$\rangle$
BrokenDown(a:Ambulance)$\equiv$
$\langle$*breakDown(a), repair(a), false*$\rangle$

6.1 Goal Revision by Weakening

Consider the goal *AmbulanceInterventionWhenMobilized*:

$\forall a : Ambulance, inc : Incident$
$Mobilized(a, inc) \Rightarrow \Diamond_{\leq 3}\ Intervention(a, inc)$

for which the obstacle below is identified in [36].

$\Diamond(\exists a : Ambulance, inc : Incident$
$Mobilized(a, inc) \wedge (\neg Intervention(a, inc)\ \mathcal{U}\ BrokenDown(a))$

The obstacle captures the situation in which an ambulance does not intervene in an incident after mobilization because it broke down. A domain-consistent obstruction trace exemplifying this is as follows.

$0 : \{report(inc_1)\}$,
$1 : \{allocate(a_1, inc_1), assign(a_1)\}$,
$2 : \{mobilize(a_1, inc_1), breakDown(a_1)\}$,
$3 : \{allocate(a_2, inc_1), assign(a_2), demobilize(a_1, inc_1)\}$,
$4 : \{mobilize(a_2, inc_1)\}$,
$5 : \{intervene(a_2, inc_1)\}$

This illustrates a scenario where an ambulance is allocated to a reported incident. When mobilized, it breaks down. Another ambulance is allocated to the same incident and intervenes. The obstacle is satisfied at time-point 3.

A domain-consistent non-obstruction trace may be one in which the original ambulance does not break down and successfully mobilizes and intervenes.

$0 : \{report(inc_1)\}$,
$1 : \{allocate(a_1, inc_1), assign(a_1)\}$,
$2 : \{mobilize(a_1, inc_1)\}$,
$3 : \{intervene(a_1, inc_1)\}$,
$4 : \{resolve(inc_1)\}$

Our approach generates alternative revisions to the goal *AmbulanceInterventionWhenMobilized* in which a fluent expression is added to *AmbulanceInterventionWhenMobilized*'s current condition.

$\forall a : Ambulance, inc : Incident.$
$Mobilized(a, inc) \wedge \neg BrokenDown(a)$
$\Rightarrow \Diamond_{\leq 3}\ Intervention(a, inc)$
$\forall a : Ambulance, inc : Incident.$
$Mobilized(a, inc) \wedge \Box\neg BrokenDown(a)$
$\Rightarrow \Diamond_{\leq 3} Intervention(a, inc)$
$\forall a : Ambulance, inc : Incident.$
$Mobilized(a, inc) \wedge (\neg BrokenDown(a)\ \mathcal{U} Intervention(a, inc))$
$\Rightarrow \Diamond_{\leq 3}\ Intervention(a, inc)$

The first revision requires intervention whenever the ambulance is mobilized and not broken down. The second is restricted to cases when the ambulance is mobilized and the ambulance never breaks downs. The third is conditional on the ambulance being mobilized, eventually intervening and not breaking down prior to its intervention. The disjunction of the last two correspond to the weakened version obtained in [36], whilst the first is not obtainable using existing approaches. We argue that the first goal is preferable since the second is not realizable and the third is conditional on the occurrence of intervention which overlooks cases in which intervention never occurs. With each solution, the learning automatically propagates the changes to the rest of the goal model where necessary, resulting in three alternative goal models. For instance the revised goal model containing the first goal revision also includes revisions to the goal *AmbulanceMobilizedWhenAllocated*,

$\forall a : Ambulance, inc : Incident.$
$Allocated(a, inc) \wedge free(a)\ \wedge time\_dist(inc.loc, a.loc) \leq 11$
$\Rightarrow \exists a' : Ambulance$
$Mobilized(a', inc) \wedge (\neg BrokenDown(a'))$

and to four other goals in the model, with a $\zeta(M, M') = 6$.

6.2 Goal Revision by Substitution

Consider the goal *AmbulanceMobilizedFromMobOrder*.

$\forall a : Ambulance, inc : Incident$
$MobOrderIssued(a, inc)\ \Rightarrow \Diamond_{\leq 2} Mobilized(a, inc)$

Suppose D' includes

$\forall a : Ambulance, inc : Incident.\ Allocated(a, inc)$
$\Rightarrow \neg\exists inc' : Incident.\ inc' \neq inc\ \wedge Allocated(a, inc')$
$\forall a : Ambulance, inc : Incident.$
$Mobilized(a, inc)\ \Rightarrow AmbMobilized(inc)$

$\forall a : Ambulance, inc : Incident.\ Allocated(a, inc)$
$\Rightarrow AmbAllocated(inc)$
$\forall a : Ambulance, inc : Incident.$
$Intervention(a, inc) \Rightarrow AmbIntervention(inc)$

where *AmbAllocated* (reps. *AmbIntervention*) is a fluent initiated when an ambulance is allocated to (resp. intervened in) incident *inc* and terminated when no ambulance is allocated to (resp. intervening in) incident *inc*. An obstacle to the goal *AmbulanceMobilizedFromMobOrder* is

$O = \Diamond(\exists a : Ambulance, inc : Incident$
$MobOrderIssued(a, inc) \wedge \Box_{\leq 2} \neg Mobilized(a, inc))$

An obstruction trace in $\Sigma(O \wedge D')$ is as follows.

$0 : \{report(inc_1)\},$
$1 : \{allocate(a_1, inc_1), assign(a_1), issueMobOrder(a_1, inc_1)\},$
$2 : \{deallocate(a_1, inc_1), allocate(a_2, inc_1), assign(a_2)\},$
$3 : \{mobilize(a_2, inc_1)\},$
$4 : \{intervene(a_2, inc_1)\}$

In this trace, an ambulance is deallocated soon after it was allocated to an incident, thus not only obstructing the goal *AmbulanceMobilizedFromMobOrder* but also its ancestors *MobilizationOrderIssuedAtStation* and *AmbulanceMobilizedWhenAllocated and AmbulanceInterventionWhenAllocated*. The non-obstruction trace (not shown here owing to space limit) exemplifies a scenario in which the ambulance for which the mobilization order was issued mobilizes towards the incident. The learning system identifies the problem in the goal requiring the ambulance for which the mobilization order was issued to mobilize. The impact of this obstruction affects the satisfaction of others goals in the model. A resolution is automatically generated in which the *Mobilized*(a, *inc*) literal in *AmbulanceMobilizedFromMobOrder* is replaced with $AmbMobilized(inc)$, which is true if there is an ambulance that mobilizes to the incident.

$\forall a : Ambulance, inc : Incident$
$(MobOrderIssued(a, inc) \Rightarrow AmbMobilized(inc))$

The change is automatically propagated to other goals in the model, in this case requiring for instance the goal *AmbulanceInterventionWhenAllocated*

$\forall a : Ambulance, inc : Incident$
$(Allocated(a, inc) \Rightarrow Intervention(a, inc))$

to be substituted by

$\forall a : Ambulance, inc : Incident$
$(Allocated(a, inc) \Rightarrow AmbIntervention(inc))$

The propagation resulted in changes to seven other goals in the model with a $\zeta(M, M') = 18$.

7. DISCUSSION AND RELATED WORK

Risk analysis is central to the requirements engineering process (e.g., [36, 8, 16]). In this context, *risk* is the effect of uncertainty on the achievement of system goals for which software is to be developed. The term obstacle was introduced in [30] to support informal scenario-based reasoning about goal obstruction. They were then used to drive requirements elaboration [7] de-idealizing the environment in which the software is to run.

The problem of obstacle identification has been addressed in different degrees of formality by many (e.g.,[33, 27, 16, 36, 6] among others). Obstacle identification is also inspired by and strongly related to techniques from the domain of system safety [24] such as with hazard analysis [23], fault trees [24] and threat trees [32]. Risk assessment is grounded in probabilistic or quantitative analysis [11] for which much automated reasoning support exists. In requirements engineering, risk assessment has been developed extensively (e.g. [16, 12]). Obstacle likelihood and criticality may be determined quantitatively over obstacle refinement trees and goal refinement trees, respectively [12, 31].

Obstacle resolution is a complex task not only because of the multiple strategies [36] that can be used but also because resolution strategies cannot be applied locally to the obstructed goal; resolution requires revising the goal model, modifying higher-level goals and possibly changing the structure of the goal graph. Operators encoding risk control strategies such as obstacle elimination (e.g., avoid the obstacle or substitute the obstructed goal) and obstacle tolerance (e.g., mitigate consequences) restore the goal [36, 35] but do not provide solutions for how to integrate the resolution patterns into a goal graph. Recently, [13] discussed the properties that the introduction of countermeasures should satisfy, such as progress towards completeness. However, a major problem remains: how to introduce and propagate countermeasures in a goal model while guaranteeing these properties are satisfied.

Our approach presents a new dimension to the framework described in [5], the integration of model checking with constraint-driven learning. Various instantiations of this framework have been developed (e.g., [2, 3, 4, 6, 1] but all have been concerned with local repairs; they do not address the problem of propagating repairs through a specification.

8. CONCLUSION

This paper introduces an automated learning-based revision approach for resolving obstacles in goal models. The method takes as input a goal model M, domain properties D and obstacles $\{O_i\}$ to goals $\{G_j\}$ in M. It computes traces exemplifying obstruction occurrences and non-occurrences in D using an off-the-shelf model checker. From these traces, it automatically generates revisions to the obstructed goals and propagate the changes through the goal model. The output is a set of alternative revised goal models that guarantee the *elimination of the obstacle*, the *correctness of the revised goal model* and the *preservation of non-obstruction behaviour* with *minimal change* to the original model. Our approach provides a systematic solution to the problem of which resolution strategy to deploy and how, by supporting the automatic exploration of all possible acceptable resolutions at once. It is generic in the sense that it may be used with different trace generation methods. Although the paper does the revision and propagation within the context of obstacle resolution, the results indicate that the approach can also be used to compute propagation of local changes globally over a goal model. It could therefore be applied to support other goal-oriented requirements engineering tasks.

Following on from this, we will extend this work to handle other strategies such as obstacle tolerance and reduction, and other forms of revisions that may affect the structure of the goal model. We also plan to integrate techniques for handling preferences amongst alternative revisions.

9. ACKNOWLEDGMENTS

We thank Duangtida Athakravi for her valuable support with the learning-based revision system. This work is partially funded by Alrajeh's Imperial College Junior Research Fellowship award.

10. REFERENCES

[1] D. Alrajeh and R. Craven. Automated error-detection and repair for compositional software specifications. In *Proceedings of the 12th International Conference on Software Engineering and Formal Methods*, pages 111–127, 2014.

[2] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning operational requirements from goal models. In *Proceedings of 31st International Conference on Software Engineering*, pages 265–275, 2009.

[3] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Deriving non-zeno behaviour models from goal models using ILP. *Journal on Formal Aspects of Computing*, 22:217–241, 2010.

[4] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning from vacuously satisfiable scenario-based specifications. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, pages 377–393, 2012.

[5] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Automated support for diagnosis and repair. *Communications of the ACM*, 58(2):65–72, 2015.

[6] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo, and S. Uchitel. Generating obstacle conditions for requirements completeness. In *Proceedings of the 34th International Conference on Software Engineering*, pages 705–715, 2012.

[7] A. Antón and C. Potts. The use of goals to surface requirements for evolving systems. In *Proceedings of the International Conference on Software Engineering*, pages 157–166, 1998.

[8] Y. Asnar, P. Giorgini, and J. Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, 16(2):101–116, 2011.

[9] D. Athakravi, D. Alrajeh, B. Broda, and A. Russo. Inductive learning using constraint-driven bias. In *Proceedings of the 24th International Conference on Inductive Logic Programming*, 2014.

[10] D. Athakravi, D. Corapi, K. Broda, and A. Russo. Learning through hypothesis refinement using answer set programming. In *Proceedings of the 23rd International Conference on Inductive Logic Programming*, pages 31–46, 2013.

[11] T. Bedford and R. Cooke. *Probabilistic risk analysis: foundations and methods*. Cambridge University Press, 2001.

[12] A. Cailliau and A. van Lamsweerde. Assessing requirements-related risks through probabilistic goals and obstacles. *Requirements Engineering*, 18(2):129–146, 2013.

[13] A. Cailliau and A. van Lamsweerde. Integrating exception handling in goal models. In *Proceedings of the 22nd International Requirements Engineering Conference*, pages 43–52, 2014.

[14] D. Corapi, A. Russo, and E. Lupu. Inductive logic programming as abductive search. In *Technical Communications of the 26th International Conference on Logic Programming*, 2010.

[15] D. Corapi, A. Russo, and E. Lupu. Inductive logic programming in answer set programming. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, pages 91–97, 2011.

[16] M. Feather and S. Cornford. Quantitative risk-based requirements reasoning. *Requirements Engineering*, 8:248–265, 2003.

[17] A. Finkelstein and J. Dowell. A comedy of errors: The london ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, pages 2–4, 1996.

[18] D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 257–266, 2003.

[19] A. Ingram. What is an electronic handbrake. https://www.carwow.co.uk/blog/Electronic-parking-brake-explained, 2014. [Online; accessed 15-Feb-2016].

[20] R. Kowalski and M. Sergot. A logic-based calculus of events. *New generation computing*, 4(1):67–95, 1986.

[21] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*, volume 651 of *(LNCS)*. Springer, 1992.

[22] E. Letier, D. Stefan, and E. T. Barr. Uncertainty, risk, and information value in software requirements and architecture. In *Proceedings of the 36th International Conference on Software Engineering*, pages 883–894, 2014.

[23] N. Leveson. An approach to designing safe embedded software. In *Proceedings of the 2nd International Conference on Embedded Software*, pages 15–29, 2002.

[24] N. G. Leveson. *Safeware: System Safety and Computers*. ACM, New York, NY, USA, 1995.

[25] J. Lloyd. *Foundations of logic programming*. Springer, 1984.

[26] M. S. Lund, B. Solhaug, and K. Stlen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[27] R. Lutz, A. Patterson-Hine, S. Nelson, C. R. Frost, D. Tal, and R. Harris. Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle. *Requirements Engineering*, 12:41–54, 2006.

[28] S. Muggleton and F. Marginean. Logic-based artificial intelligence. chapter Logic-based Machine Learning, pages 315–330. Kluwer Academic Publishers, 2000.

[29] S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19–20:629–679, 1994.

[30] C. Potts. Using schematic scenarios to understand user needs. In *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*, pages 247–256, 1995.

[31] M. Sabetzadeh, D. Falessi, L. C. Briand, S. D. Alesio, D. McGeorge, V. ÃĚhjem, and J. Borg. Combining goal models, expert elicitation, and probabilistic simulation for qualification of new technology. In *Proceedings of the IEEE 13th International Symposium on High-Assurance Systems Engineering*, pages 63–72, 2011.

[32] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2000.

[33] A. Sutcliffe, N. Maiden, S. Minocha, and D. Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24:1072–1088, 1998.

[34] C. K. F. Tang and E. Ternovska. Model checking abstract state machines with answer set programming. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 443–458, 2005.

[35] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[36] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirement engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[37] S. Wrobel. First order theory refinement. In L. React, editor, *Advances in Inductive Logic Programming*, pages 14–33. IOS Press, Amsterdam, 1996.