ACoRe: Automated Goal-Conflict Resolution

Luiz Carvalho[1], Renzo Degiovanni[1], Matías Brizzio[2,3], Maxime Cordy[1],
Nazareno Aguirre[4], Yves Le Traon[1], Mike Papadakis[1]

[1] SnT, University of Luxembourg, Luxembourg City, Luxembourg
{luiz.carvalho,renzo.degiovanni,maxime.cordy,yves.traon,mike.papadakis}@uni.lu
[2] IMDEA Software Institute, Madrid, Spain
matias.brizzio@imdea.org
[3] Universidad Politécnica de Madrid, Madrid, Spain
[4] Universidad Nacional de Río Cuarto and CONICET, Río Cuarto, Argentina
naguirre@dc.exa.unrc.edu.ar

Abstract. System goals are the statements that, in the context of software requirements specification, capture how the software should behave. Many times, the understanding of stakeholders on what the system should do, as captured in the goals, can lead to different problems, from clearly contradicting goals, to more subtle situations in which the satisfaction of some goals inhibits the satisfaction of others. These latter issues, called *goal divergences*, are the subject of *goal conflict analysis*, which consists of identifying, assessing, and resolving divergences, as part of a more general activity known as goal refinement.

While there exist techniques that, when requirements are expressed formally, can automatically identify and assess goal conflicts, there is currently no automated approach to support engineers in *resolving* identified divergences. In this paper, we present ACoRe, the first approach that automatically proposes potential resolutions to goal conflicts, in requirements specifications formally captured using linear-time temporal logic. ACoRe systematically explores syntactic modifications of the conflicting specifications, aiming at obtaining resolutions that disable previously identified conflicts, while preserving specification consistency. ACoRe integrates modern multi-objective search algorithms (in particular, NSGA-III, WBGA, and AMOSA) to produce resolutions that maintain coherence with the original conflicting specification, by searching for specifications that are either *syntactically* or *semantically* similar to the original specification.

We assess ACoRe on 25 requirements specifications taken from the literature. We show that ACoRe can successfully produce various conflict resolutions for each of the analyzed case studies, including resolutions that resemble specification repairs manually provided as part of conflict analyses.

1 Introduction

Many software defects that come out during software development originate from incorrect understandings of what the software being developed should do [24].

These kinds of defects are known to be among the most costly to fix, and thus it is widely acknowledged that software development methodologies must involve phases that deal with the elicitation, understanding, and precise specification of *software requirements.* Among the various approaches to systematize this requirements phase, the so-called *goal-oriented requirements engineering* (GORE) methodologies [13,55] provide techniques that organize the modeling and analysis of software requirements around the notion of *system goal.* Goals are prescriptive statements that capture how the software to be developed should behave, and in GORE methodologies are subject to various activities, including goal decomposition, refinement, and the assignment of goals [3,13,15,39,55,56].

The characterization of requirements as formally specified system goals enables tasks that can reveal flaws in the requirements. Formally specified goals allow for the analysis and identification of *goal divergences*, situations in which the satisfaction of some goals inhibits the satisfaction of others [9,16]. These divergences arise as a consequence of *goal conflicts.* A *conflict* is a condition whose satisfaction makes the goals inconsistent. Conflicts are dealt with through *goal-conflict analysis* [58], which comprises three main stages: *(i)* the *identification* stage, which involves the identification of conflicts between goals; *(ii)* the *assessment* stage, aiming at evaluating and prioritizing the identified conflicts according to their likelihood and severity; and *(iii)*, the *resolution stage*, where conflicts are resolved by providing appropriate countermeasures and, consequently, transforming the goal model, guided by the criticality level.

Goal conflict analysis has been the subject of different automated techniques to assist engineers, especially in the conflict identification and assessment phases [16,18,43,56]. However, no automated technique has been proposed for dealing with goal conflict *resolution.* In this paper, we present ACoRe, the first automated approach that deals with the goal-conflict resolution stage. ACoRe takes as input a set of goals formally expressed in Linear-Time Temporal Logic (LTL) [45], together with previously identified conflicts, also given as LTL formulas. It then searches for candidate *resolutions*, i.e., syntactic modifications to the goals that remain consistent with each other, while disabling the identified conflicts. More precisely, ACoRe employs modern search-based algorithms to efficiently explore syntactic variants of the goals, guided by a syntactic and semantic similarity with the original goals, as well as with the inhibition of the identified conflicts. This search guidance is implemented as (multi-objective) fitness functions, using Levenshtein edit distance [42] for syntactic similarity, and approximated LTL model counting [8] for semantic similarity. ACoRe exploits this fitness function to search for candidate resolutions, using various alternative search algorithms, namely a Weight-Based Genetic Algorithm (WBGA) [29], a Non-dominated Sorted Genetic Algorithm (NSGA-III) [14], an Archived Multi-Objective Simulated Annealing search (AMOSA) [6], and an unguided search approach, mainly used as a baseline in our experimental evaluations.

Our experimental evaluation considers 25 requirements specifications taken from the literature, for which goal conflicts are automatically computed [16]. The results show that ACoRe is able to successfully produce various conflict

resolutions for each of the analysed case studies, including resolutions that resemble specification repairs manually provided as part of conflict analyses. In this assessment, we measured their similarity concerning the ground-truth, i.e., to the manually written repairs, when available. The genetic algorithms are able to resemble 3 out of 8 repairs in the ground truth. Moreover, the results show that ACoRe generates more non-dominated resolutions (their finesses are not subsumed by other repairs in the output set) when adopting genetic algorithms (NSGA-III or WBGA), compared to AMOSA or unguided search, favoring genetic multi-objective search over other approaches.

2 Linear-Time Temporal Logic

2.1 Language Formalism

Linear-Time Temporal Logic (LTL) is a logical formalism widely used to specify reactive systems [45]. In addition, GORE methodologies (e.g. KAOS) have also adopted LTL to formally express requirements [55] and taken advantage of the powerful automatic analysis techniques associated with LTL to improve the quality of their specifications (e.g., to identify inconsistencies [17]).

Definition 1 (LTL Syntax). *Let AP be a set of propositional variables. LTL formulas are inductively defined using the standard logical connectives, and the temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until), as follows:*

(a) constants true and false are LTL formulas;
(b) every $p \in AP$ is an LTL formula;
(c) if φ and ψ are LTL formulas, then $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi\mathcal{U}\psi$ are also LTL formulas.

LTL formulas are interpreted over infinite traces of the form $\sigma = s_0 \, s_1 \ldots$, where each s_i is a propositional valuation on 2^{AP} (i.e., $\sigma \in 2^{AP^\omega}$).

Definition 2 (LTL Semantic). *We say that trace $\sigma = s_0, s_1, \ldots$ satisfies a formula φ, written $\sigma \models \varphi$, if and only if φ holds at the initial state of the trace, i.e. $(\sigma, 0) \models \varphi$. The last notion is inductively defined on the shape of φ as follows:*

(a) $(\sigma, i) \models p \Leftrightarrow p \in s_i$
(b) $(\sigma, i) \models (\phi \vee \psi) \Leftrightarrow (\sigma, i) \models \phi$ or $(\sigma, i) \models \psi$
(c) $(\sigma, i) \models \neg\phi \Leftrightarrow (\sigma, i) \not\models \phi$
(d) $(\sigma, i) \models \bigcirc\phi \Leftrightarrow (\sigma, i+1) \models \phi$
(e) $(\sigma, i) \models (\phi \, \mathcal{U} \, \psi) \Leftrightarrow \exists_{k \geq 0} : (\sigma, k) \models \psi$ and $\forall_{0 \leq j < k} : (\sigma, j) \models \phi$

Intuitively, formulas with no temporal operator are evaluated in the first state of the trace. Formula $\bigcirc\varphi$ is true at position i, iff φ is true in position $i+1$. Formula $\varphi\mathcal{U}\,\psi$ is true in σ iff formula φ holds at every position until ψ holds.

Definition 3 (Satisfiability). *An LTL formula φ is said* satisfiable *(SAT) iff there exists at least one trace satisfying φ.*

We also consider other typical connectives and operators, such as, $\wedge$, $\square$ (always), $\diamond$ (eventually) and $\mathcal{W}$ (weak-until), that are defined in terms of the basic ones. That is, $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\diamond\phi \equiv true\,\mathcal{U}\phi$, $\square\phi \equiv \neg\diamond\neg\phi$, and $\phi\mathcal{W}\psi \equiv (\square\phi) \vee (\phi\mathcal{U}\psi)$.

2.2   Model Counting

The *model counting* problem consists of calculating the number of models that satisfy a formula. Since the models of LTL formulas are infinite traces, it is often the case that analysis is restricted to a class of canonical *finite* representation of *infinite* traces, such as lasso traces or tree models. Notably, this is the case in bounded model checking for instance [7].

Definition 4 (Lasso Trace). *A lasso trace σ is of the form $\sigma = s_0 \dots s_i(s_{i+1} \dots s_k)^\omega$, where the states $s_0 \dots s_k$ conform the* base *of the trace, and the loop from state s_k to state s_{i+1} is the part of the trace that is repeated infinitely many times.*

For example, an LTL formula $\square(p \vee q)$ is satisfiable, and one satisfying lasso trace is $\sigma_1 = \{p\}; \{p,q\}^\omega$, wherein the first state p holds, and from the second state both p and q are valid forever. Notice that the base in the lasso trace σ_1 is the sequence containing both states $\{p\}; \{p,q\}$, while the state $\{p,q\}$ is the sequence in the loop part.

Definition 5 (LTL Model Counting). *Given an LTL formula φ and a bound k, the (bounded)* model counting *problem consists in computing how many lasso traces of at most k states exist for φ. We denote this as $\#(\varphi, k)$.*

Since existing approaches for computing the exact number of lasso traces are ineffective [25], Brizzio et. al [8] recently developed a novel model counting approach that approximates the number (of prefixes) of lasso traces satisfying an LTL formula. Intuitively, instead of counting the number of lasso traces of length k, the approach of Brizzio et. al [8] aims at approximating the number of bases of length k corresponding to some satisfying lasso trace.

Definition 6 (Approximate LTL Model Counting). *Given an LTL formula φ and a bound k, the approach of Brizzio et. al [8] approximates the number of bases $w = s_0 \dots s_k$, such that for some i, the lasso trace $\sigma = s_0 \dots (s_i \dots s_k)^\omega$ satisfies φ (notice that prefix w is the base of σ). We denote $\#\mathrm{APPROX}(\varphi, k)$ to the number computed by this approximation.*

ACoRe uses $\#\mathrm{APPROX}$ model counting to compute the semantic similarity between the original specification and the candidate goal-conflict resolutions.

3 The Goal-Conflict Resolution Problem

Goal-Oriented Requirements Engineering (GORE) [55] drives the requirements process in software development from the definition of high-level goals that state how the system to be developed should behave. Particularly, goals are prescriptive statements that the system should achieve within a given domain. The domain properties are descriptive statements that capture the domain of the problem world. Typically, GORE methodologies use a logical formalism to specify the expected system behavior, e.g., KAOS uses Linear-Time Temporal Logic for specifying requirements [55]. In this context, a *conflict* essentially represents a condition whose occurrence results in the loss of satisfaction of the goals, i.e., that makes the goals *diverge* [56,57]. Formally, it can be defined as follows.

Definition 7 (Goal Conflicts). *Let $G = \{G_1, \ldots, G_n\}$ be a set of goals, and Dom be a set of domain properties, all written in LTL. Goals in G are said to diverge if and only if there exists at least one Boundary Condition (BC), such that the following conditions hold:*

- *logical inconsistency: $\{Dom, BC, \bigwedge_{1 \leq i \leq n} G_i\} \models false$*
- *minimality: for each $1 \leq i \leq n$, $\{Dom, BC, \bigwedge_{j \neq i} G_j\} \not\models false$*
- *non-triviality: $BC \neq \neg(G_1 \wedge \ldots \wedge G_n)$*

Intuitively, a BC captures a particular combination of circumstances in which the goals cannot be satisfied. The first condition establishes that, when BC holds, the conjunction of goals $\{G_1, \ldots, G_n\}$ becomes inconsistent. The second condition states that, if any of the goals are disregarded, then consistency is recovered. The third condition prohibits a boundary condition to be simply the negation of the goals. Also, the minimality condition prohibits that BC be equals to *false* (it has to be consistent with the domain *Dom*).

Goal-conflict analysis [55,56] deals with these issues, through three main stages: (1) The goal-conflicts identification phase consists in generating boundary conditions that characterize divergences in the specification; (2) The assessment stage consists in assessing and prioritizing the identified conflicts according to their likelihood and severity; (3) The resolution stage consists in resolving the identified conflicts by providing appropriate countermeasures. Let us consider the following examples found in our empirical evaluation and commonly presented in related works.

Example 1 (Mine Pump Controller - MPC). Consider the Mine Pump Controller (MPC) widely used in related works that deal with formal requirements and reactive systems [16,35]. The MPC describes a system that is in charge of activating or deactivating a pump (p) to remove the water from the mine, in the presence of possible dangerous scenarios. The MP controller monitors environmental magnitudes related to the presence of methane (m) and the high level of water (h) in the mine. Maintaining a high level of water for a while may produce

flooding in the mine, while the methane may cause an explosion when the pump is switched on. Hence, the specification for the MPC is as follows:

$$Dom : \Box((p \wedge \bigcirc(p)) \rightarrow \bigcirc(\bigcirc(\neg h))) \quad G_1 : \Box(m \rightarrow \bigcirc(\neg p)) \quad G_2 : \Box(h \rightarrow \bigcirc(p))$$

Domain property Dom describes the impact into the environment of switching on the pump (p). For instance, when the pump is kept on for 2 unit times, then the water will decrease and the level will not be high $(\neg h)$. Goal G_1 expresses that the pump should be off when methane is detected in the mine. Goal G_2 indicates that the pump should be on when the level of water is high.

Notice that this specification is consistent, for instance, in cases in which the level of water never exceeds the high threshold. However, approaches for goal-conflict identification, such as the one of Degiovanni et al. [16], can detect a conflict between goals in this specification.

The identified goal-conflict describes a divergence situation in cases in which the level of water is high and methane is present at the same time in the environment. Switching off the pump to satisfy G_1 will result in a violation of goal G_2; while switching on the pump to satisfy G_2 will violate G_1. This divergence situation clearly evidence a conflict between goals G_1 and G_2 that is captured by a boundary condition such $BC = \Diamond(h \wedge m)$.

In the work of Letier et al. [40] two resolutions were manually proposed that precisely describe what should be the software behaviour in cases where the divergence situation is reached. The first resolution proposes to refine goal G_2, by weakening it, requiring to switch on the pump only when the level of water is high and no methane is present in the environment.

Example 2 (Resolution 1 - MPC).

$$Dom : \Box((p \wedge \bigcirc(p)) \rightarrow \bigcirc(\bigcirc(\neg h)))$$
$$G_1 : \Box(m \rightarrow \bigcirc(\neg p)) \quad G_2' : \Box(h \wedge \neg m \rightarrow \bigcirc(p))$$

With a similar analysis, the second resolution proposes to weaken G_1, requiring switching off the pump when methane is present and the level of water is not high.

Example 3 (Resolution 2 - MPC).

$$Dom : \Box((p \wedge \bigcirc(p)) \rightarrow \bigcirc(\bigcirc(\neg h)))$$
$$G_1' : \Box(m \wedge \neg h \rightarrow \bigcirc(\neg p)) \quad G_2 : \Box(h \rightarrow \bigcirc(p))$$

The *resolution* stage aims at removing the identified goal-conflicts from the specification, for which it is necessary to modify the current specification formulation. This may require weakening or strengthening the existing goals, or even removing some and adding new ones.

Definition 8 (Goal-Conflict Resolution). *Let $G = \{G_1, \ldots, G_n\}$, Dom, and BC be the set of goals, the domain properties, and an identified boundary condition, respectively written in LTL. Let $M : S_1 \times S_2 \mapsto [0,1]$ and $\epsilon \in [0,1]$ be*

a similarity metric between two specifications and a threshold, respectively. We say that a resolution $R = \{R_1, \ldots, R_m\}$ resolves goal-conflict BC, if and only if, the following conditions hold:

- *consistency:* $\{Dom, R\} \not\models false$
- *resolution:* $\{BC, R\} \not\models false$
- *similarity:* $M(G, R) < \epsilon$

Intuitively, the first condition states that the refined goals in R remain consistent within the domain properties *Dom*. The second condition states that *BC* does not lead to a divergence situation in the resolution R (i.e., refined goals in R know exactly how to deal with the situations captured by *BC*). Finally, the last condition aims at using a similarity metric M to control for the degree of changes applied to the original formulation of goals in G to produce the refined goals in resolution R.

Notice that the similarity metric M is general enough to capture similarities between G and R of different natures. For instance, $M(G, R)$ may compute the *syntactic similarity* between the text representations of the original specification of goals in G and the candidate resolution R, where the number of tokens edited from G to R is the aim. On the other hand, $M(G, R)$ may compute a semantic similarity between G and R, for instance, to favour resolutions that weaken the goals (i.e. $G \rightarrow R$), or strengthen the goals (i.e. $R \rightarrow G$) or that maintain most of the original behaviours (i.e. $\#G - \#R < \epsilon$).

Precisely, ACoRe will explore syntactic modifications of goals from G, leading to newly refined goals in R, with the aim at producing candidate resolutions that are consistent with the domain properties *Dom* and resolve conflict *BC*. Assuming that the engineer is competent and the current specification is very close to the intended one [19,1], ACoRe will integrate two similarity metrics in a multi-objective search process to produce resolutions that are syntactically and semantically similar to the original specification. Particularly, ACoRe can generate exactly the same resolutions for the MPC previously discussed, manually developed by Letier et al. [40].

4   ACoRe: Automated Goal-Conflict Resolution

ACoRe takes as input a specification $S = (Dom, G)$, composed by the domain properties *Dom*, a set of goals G, and a set $\{BC_1, \ldots, BC_k\}$ of identified boundary conditions for S. ACoRe uses search to iteratively explore variants of G to produce a set $R = \{R_1, \ldots, R_n\}$ of resolutions, where each $R_i = (Dom, G^i)$, that maintain two sorts of similarities with the original specification, namely, syntactic and semantic similarity between S and each R_i. Figure 1 shows an overview of the different steps of the search process implemented by ACoRe.

ACoRe instantiates multi-objective optimization (MOO) algorithms to efficiently and effectively explore the search space. Currently, ACoRe implements four MOO algorithms, namely, the *Non-Dominated Sorting Genetic Algorithm III* (NSGA-III) [14], a *Weight-based genetic algorithm (WBGA)* [29], an *Archived*

Multi-objective Simulated Annealing (AMOSA) [6] approach, and an *unguided search* approach we use as a baseline. Let us first describe some common components shared by the algorithms (namely, the search space, the multi-objectives, and the evolutionary operators) and then get into the particular details of each approach (such as the fitness function and selection criteria).
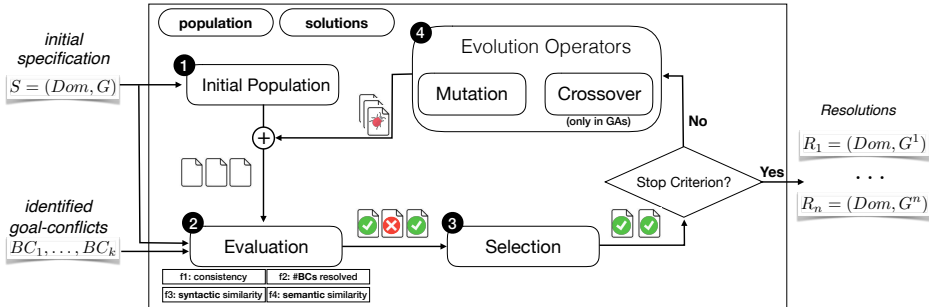

Fig. 1: Overview of ACoRe.

4.1   Search Space and Initial Population

Each individual $cR = (Dom, G')$, representing a candidate resolution, is a LTL specification over a set AP of propositional variables, where Dom captures the domain properties and G' the refined system goals. Notice that domain properties Dom are not changed through the search process since these are descriptive statements. On the other hand, ACoRe performs syntactic alterations to the original set of goals G to obtain the new set of refined goals G' that potentially resolve the conflicts given as input.

The initial population represents a sample of the search space from which the search starts. ACoRe creates one or more individuals (depending on the multi-objective algorithm being used) as the initial population by applying the mutation operator (explained below) to the specification S given as input.

4.2   Multi-Objectives: Consistency, Resolution and Similarities

ACoRe guides the search with *four objectives* that check for the validity of each of the conditions needed to be a valid goal-conflict resolution, namely, consistency, resolution and two similarity metrics (cf. Definition 8).

Given a resolution $cR = (Dom, G')$, the first objective $Consistency(cR)$ evaluates if the refined goals G' are consistent with the domain properties by using SAT solving.

$$Consistency(cR) = \begin{cases} 1 & \text{if } Dom \wedge G' \text{ is satisfiable} \\ 0.5 & \text{if } Dom \wedge G' \text{ is unsatisfiable, but } G' \text{ is satisfiable} \\ 0 & \text{if } G' \text{ is unsatisfiable} \end{cases}$$

The second objective $ResolvedBCs(cR)$ computes the ratio of boundary conditions resolved by the candidate resolution cR, among the total number of boundary conditions given as input. Hence, $ResolvedBCs(cR)$ returns values between 0 and 1, and is defined as follows:

$$ResolvedBCs(cR) = \frac{\sum_{i=1}^{k} isResolved(BC_i, G')}{k}$$

$isResolved(cR, BC_i)$ returns 1, if and only if $BC_i \wedge G'$ is satisfiable; otherwise, returns 0. Intuitively, when $BC_i \wedge G'$ is satisfiable, it means that the refined goals G' satisfies the resolution condition of Definition 8 and thus, BC_i is no longer a conflict for candidate resolution cR. In the case that cR resolves all the (k) boundary conditions, the objective $ResolvedBCs(cR)$ will return 1.

With the objective of prioritising resolutions that are in some sense similar to the original specification among the dissimilar ones, ACoRe integrates two similarity metrics. ACoRe considers one *syntactic* and one *semantic* similarity metric that will help the algorithms to focus the search in the vicinity of the specification given as input.

Precisely, objective $Syntactic(S, cR)$ refers to the distance between the text representations of the original specification S and the candidate resolution cR. To compute the syntactic similarity between LTL specifications, we use Levenshtein distance [42]. Intuitively, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other. Hence, $Syntactic(S, cR)$, is computed as:

$$Syntactic(S, cR) = \frac{maxLength - Levenshtein(S, cR)}{maxLength}$$

where $maxLength = max(length(S), length(cR))$. Intuitively, $Syntactic(S, cR)$ represents the ratio between the number of tokens changed from S to obtain cR among the maximum number of tokens corresponding to the largest specification.

On the other hand, our semantic similarity objective $Semantic(S, cR)$ refers to the system *behaviour* similarities described by the original specification and the candidate resolution. Precisely, $Semantic(S, cR)$ computes the ratio between the number of behaviours present in both, the original specification and candidate resolution, among the total number of behaviours described by the specifications. To efficiently compute the objective $Semantic(S, cR)$, ACoRe uses model counting and the approximation previously described in Definition 6. Hence, given a bound k for the lasso traces, the semantic similarity between S and cR is computed as:

$$Semantic(S, cR) = \frac{\#\text{APPROX}(S \wedge cR, k)}{\#\text{APPROX}(S \vee cR, k)}$$

Notice that, small values for $Semantic(S, cR)$ indicate that the behaviours described by S are divergent from those described by cR. In particular, in cases that S and cR are contradictory (i.e., $S \wedge cR$ is unsatisfiable), $Semantic(S, cR)$ is

0. As this value gets closer to 1, both specifications characterize an increasingly large number of common behaviors.

4.3   Evolutionary Operators

New individuals are generated through the application of the evolution operators. Particularly, our approach ACoRe implements two standard operators used for evolving LTL specifications [17,43], namely a mutation and a crossover operators. Below, we provide some examples of the application of these operators, and please refer to the complementary material for a detailed formal definition.
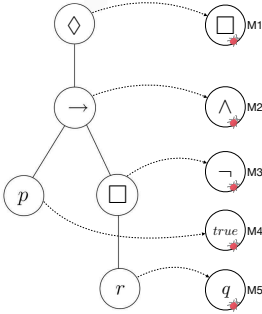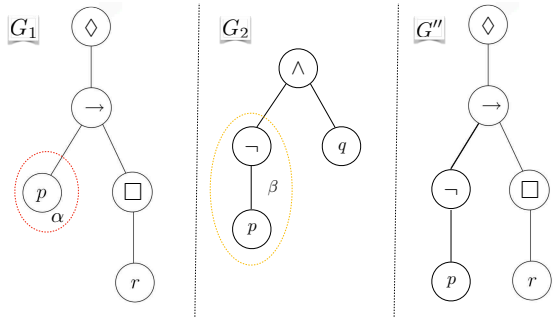

Fig. 2: Mutation operator.

Fig. 3: Crossover operator.

Given a candidate individual $cR' = (Dom, G')$, the *mutation* operator selects a goal $g' \in G'$ to mutate, leading to a new goal g'', and produces a new candidate specification $cR'' = (Dom, G'')$, where $G'' = G'[g' \mapsto g'']$, that is, G'' looks exactly as G' but goal g' is replaced by the mutated goal g''.

For instance, Figure 2 shows 5 possible mutations that we can generate for formula $\Diamond(p \to \Box r)$. Mutation M1 replaces $\Diamond$ by $\Box$, leading to $M1 : \Box(p \to \Box r)$. Mutation $M2 : \Diamond(p \land \Box r)$ replaces $\to$ by $\land$. Mutation $M3 : \Diamond(p \to \neg r)$ replaces $\Box$ by $\neg$. Mutation $M4 : \Diamond(true \to \Box r)$, reduces to $\Diamond\Box r$, replaces p by $true$. While mutation $M5 : \Diamond(p \to \Box q)$ replaces r by q.

On the contrary, the *crossover* operator takes two individuals $cR^1 = (Dom, G^1)$ and $cR^2 = (Dom, G^2)$, and produces a new candidate resolution $cR'' = (Dom, G'')$ by combining portions of both specifications. In other words, it takes one goal from each individual, i.e. $G_1 \in G^1$ and $G_2 \in G^2$, and generates a new goal G'' that is obtained by replacing a subformula α of G_1 by a subformula β taken from G_2. For instance, Figure 3 provides an illustration of how this operator works. Particularly, subformula $\alpha : p$ is selected from goal $G_1 : \Diamond(p \to \Box r)$, while subformula $\beta : \neg p$ is selected from goal $G_2 : \neg p \land q$. Hence, by replacing in G_1 subformula α by subformula β, the crossover operators generate a new goal $G'' : \Diamond(\neg p \to \Box r)$.

It is worth mentioning that the four multi-objective search algorithms implemented by ACoRe use the mutation operator to evolve the population. However, only two of the algorithms that implement two different genetic algorithms (i.e. NSGA-III and WBGA) use the crossover operator to evolve the population.

4.4   Multi-Objective Optimisation Search Algorithms

In a *multi-objective optimisation* (MOO) problem there is a set of solutions, called the *Pareto-optimal* (PO) set, which is considered to be equally important. Given two individuals x_1 and x_2 from the search-space S, and $f_1, \ldots, f_n$ a set of (maximising) fitness functions, where $f_i : S \to \mathbb{R}$, we say that x_1 dominates x_2 if (a) x_1 is not worse than x_2 in all objectives and (b) x_1 is strictly better than x_2 at least in one objective. Typically, MOO algorithms evolve the candidate population with the aim to converge to a set of *non-dominated* solutions as close to the true PO set as possible and maintain as diverse a solution set as possible. There are many variants of MOO algorithms that have been successfully applied in practice [27]. ACoRe implements four multi-objective optimization algorithms to explore the search space to generate goal-conflict resolutions.

AMOSA. The Archived Multi-objective Simulated Annealing (AMOSA) [6] is an adaptation of the simulated annealing algorithm [34] for multi-objectives. AMOSA only analyses one (current) individual per iteration, and a new individual is created by the application of the mutation operator. AMOSA has two particular features that make it promising for our purpose. During the search, it maintains an "archive" with the non-dominated candidates explored so far, that is, candidates whose fitness values are not subsumed by other generated individuals. Moreover, when a new individual is created that does not dominate the current one, it is not immediately discarded and can still be selected among the current individual with some probability that depends on the "temperature" (a function that decreases over time). At the beginning the temperature is high, then new individuals with worse fitness than the current element, are likely to be selected, but this probability decreases over the iterations. This strategy helps in avoiding local maximums and exploring more diverse potential solutions.

WBGA. ACoRe also implements a classic *Weight-based genetic algorithm (WBGA)* [29]. In this case, WBGA maintains a fixed number of individuals in each iteration (a configurable parameter), and applies both the mutation and crossover operators to generate new individuals. WBGA computes the fitness value for each objective and combines them into a single fitness f defined as:

$$f(S, cR) = \alpha * Consistency(cR) + \beta * ResolvedBCs(cR) + \\ \gamma * Syntax(S, cR) + \delta * Semantic(S, cR)$$

where weights $\alpha = 0.1$, $\beta = 0.7$, $\gamma = 0.1$, and $\delta = 0.1$ are defined by default (empirically validated), but these can be configured to other values if desired. In each iteration, WBGA sorts all the individuals according to their fitness value (descending order) and selects best ranked individuals to survive to the next

iteration (other selectors can be integrated). Finally, WBGA reports all the resolutions found during the search.

NSGA-III. ACoRe also implements the Non-Dominated Sorting Genetic Algorithm III (NSGA-III) [14] approach. It is a variant of a genetic algorithm that also uses mutation and crossover operators to evolve the population. In each iteration, it computes the fitness values for each individual and sorts the population according to the Pareto dominance relation. Then it creates a partition of the population according the level of the individuals in the Pareto dominance relation (i.e., non-dominated individuals are in Level-1, Level-2 contains the individuals dominated only by individuals in Level-1, and so on). Thus, NSGA-III selects only one individual per non-dominated level with the aim of diversifying the exploration and reducing the number of resolutions in the final Pareto-front.

ACoRe also implements an **Unguided Search** algorithm that does not use any of the objectives to guide the search. It randomly selects individuals and applies the mutation operator to evolve the population. After generating a maximum number of individuals (a given parameter of the algorithm), it checks which ones constitute a valid resolution for the goal-conflicts given as input.

5   Experimental Evaluation

We start our analysis by investigating the effectiveness of ACoRe in resolving goal-conflicts. Thus, we ask:

RQ1 *How effective is* ACoRe *at resolving goal-conflicts?*

To answer this question, we study the ability of ACoRe to generate resolutions in a set of 25 specifications for which we have identified goal-conflicts.

Then, we turn our attention to the *"quality"* of the resolution produced by ACoRe and study if ACoRe is able to replicate some of the manually written resolutions gathered from the literature (ground-truth). Thus, we ask:

RQ2 *How able is* ACoRe *to generate resolutions that match with resolutions provided by engineers (i.e. manually developed)?*

To answer RQ2, we check if ACoRe can generate resolutions that are equivalent to the ones manually developed by the engineer.

Finally, we are interested in analyzing and comparing the performance of the four search algorithms integrated by ACoRe. Thus, we ask:

RQ3 *What is the performance of* ACoRe *when adopting different search algorithms?*

To answer RQ3, we basically employ standard quality indicators (e.g. hypervolume (HV) and inverted generational distance (IGD)) to compare the Pareto-front produced by ACoRe when the different search algorithms are employed.

5.1   Experimental Procedure

We consider a total of 25 requirements specifications taken from the literature and different benchmarks. These specifications were previously used by goal-conflicts identification and assessment approaches [4,16,17,18,43,56].

Table 1: LTL Requirements Specifications and Goal-conflicts Identified.

Specification	#Dom + #Goals	#BCs		Specification	#Dom + #Goals	#BCs
minepump	3	14		rrcs	4	14
simple arbiter-v1	4	28		achieve-avoid pattern	3	16
simple arbiter-v2	4	20		retraction pattern-1	2	2
prioritized arbiter	7	11		retraction pattern-2	2	10
arbiter	3	20		RG2	2	9
detector	2	15		lily01	3	5
ltl2dba27	1	11		lily02	3	11
round robin	9	12		lily11	3	5
tcp	2	11		lily15	3	19
atm	3	24		lily16	6	38
telephone	5	4		ltl2dba theta-2	1	3
elevator	2	3		ltl2dba R-2	1	5
				simple arbiter icse2018	11	20

We start by running the approach of Degiovanni et al. [17] on each subject to identify a set of boundary conditions. Table 1 summarises, for each case, the number of domain properties and goals, and the number of boundary conditions (i.e. goal-conflicts) computed with the approach of Degiovanni et al. [17]. Notice that we use the set of "weakest"[1] boundary conditions returned by [17], in the sense that by removing all of these we are guaranteed to remove all the boundary conditions computed.

Then, we run ACoRe to generate resolutions that remove all the identified goal-conflicts. We configure ACoRe to explore a maximum number of 1000 individuals with each algorithm. We repeat this process 10 times to reduce potential threats [5] raised by the random elections of the search algorithms.

To answer RQ1, we run ACoRe and report the number of *non-dominated* resolutions produced by each implemented algorithm (i.e. those resolutions whose fitness values are not subsumed by other individuals).

To answer RQ2, we collected from the literature 8 cases in which authors reported a "buggy" version of the specification and a "fixed" version of the same specification. We take the buggy version and compute a set of boundary conditions for it that are later fed into ACoRe to automatically produce a set of resolutions. We then compare the resolutions produced by our ACoRe and the "fixed" versions we gathered from the literature. We basically analyse, by using sat solving, if any of the resolutions produced by ACoRe is equivalent to the manually developed fixed version.

To answer RQ3, we perform an objective comparison of the performance of the four search algorithms implemented by ACoRe by using two standard

---

[1] A formula A is weaker than B, if $B \land \neg A$ is unsatisfiable, i.e., if B implies A.

quality indicators: hypervolume (HV) [62] and inverted generational distance (IGD) [12]. The recent work of Wu et al. [61] indicates that quality indicators HV and IGD are the prefered ones for assessing genetic algorithms and Pareto evolutionary algorithms such as the ones ACoRe implements (NSGA-III, WBGA, and AMOSA). These quality indicators are useful to measure the *convergence*, *spread*, *uniformity*, and *cardinality* of the solutions computed by the algorithms. More precisely, hypervolume (HV) [42,54] is a *volume-based indicator*, defined by the Nadir Point [38,62], that returns a value between 0 and 1, where a value near to 1 indicates that the Pareto-front converges very well to the reference point [42] (also, high values for HV are good indicator of uniformity and spread of the Pareto-front [54]). The Inverted Generational Distance (IGD) indicator is a *distance-based indicator* that also computes convergence and spread [42,54]. In summary, IGD measures the mean distance from each reference point to the nearest element in the Pareto-optimal set [12,54]. We also perform some statistical analysis, namely, the Kruskal-Wallis H-test [37], the Mann-Whitney U-test [44], and Vargha-Delaney A measure $\hat{A}_{12}$ [59], to compare the performance of the algorithms. Intuitively, the *p-value* will tell us if the performance between the algorithms measured in terms of the HV and IGD is statistical significance, while the A-measure will tell us how frequent one algorithm obtains better indicators than the others.

ACoRe is implemented in Java into the JMetal framework [50]. It integrates the LTL satisfiability checker Polsat [41], a portfolio tool that runs in parallel with four LTL solvers, helping us to efficiently compute the fitness functions. Moreover, ACoRe uses the OwL library [36] to parse and manipulate the LTL specifications. The quality indicators also are implemented by the JMetal framework and the statistical tests by the Apache Common Math. We ran all the experiments on a cluster with nodes with Xeon E5 2.4GHz, with 5 CPUs-nodes and 8GB of RAM available per run.

Regarding the setting of the algorithms, the population size of 100 individuals was defined and the fitness evaluation was limited to a number of 1000 individuals. Moreover, the timeout of the model counting and SAT solvers were configured as 300 seconds. The probability of crossover application was 0.1, while mutation operators were always applied. A tournament selection of four solutions was used for NSGA-III, while WBGA instantiated Bolzman's selection with a decrement exponential function. The WBGA was configured to weight the fitness functions as a proportion of 0.1 in the Status, 0.7 in the ResolvedBC, 0.1 in Syntactic, and 0.1 in Semantic. The AMOSA used an archive of crowding distance, while the cooling scheme relied on a decrement exponential function.

The case studies and results are publicly available at https://sites.google.com/view/acore-goal-conflict-resolution/.

6 Experimental Results

6.1 RQ1: Effectiveness of ACoRe

Table 2 reports the average number of non-dominated resolutions produced by the algorithms in the 10 runs. First, it is worth mentioning that when ACoRe uses any of the genetic algorithms (NSGA-III or WBGA), it successfully generates at least one resolution for all the case studies. However, AMOSA fails in producing a resolution for the `lily16` and `simple arbiter icse2018` in 2 and 1 cases of the 10 runs, respectively. Despite that Unguided search succeeds in the majority of the cases, it was not able to produce any resolution for the `prioritized arbiter`, and failed in producing a resolution in 5 out of the 10 runs for the `simple-arbiter-v2`.

Table 2: Effectiveness of ACoRe in producing resolutions.

Specification	NSGA-III	WBGA	AMOSA	Unguided
minepump	5.0	**6.5**	1.8	5.1
simple arbiter-v1	**4.8**	3.1	2.0	4.1
simple arbiter-v2	3.1	**3.4**	2.3	0.5
prioritized arbiter	3.1	**3.7**	2.2	0.0
arbiter	**5.8**	2.7	3.0	5.5
detector	4.9	4.8	3.2	**6.1**
ltl2dba27	3.0	**4.2**	3.5	4.0
round robin	**7.0**	4.2	4.7	4.7
tcp	6.4	4.9	2.0	**7.4**
atm	3.9	**6.3**	3.3	4.5
telephone	**4.7**	4.4	2.2	4.5
elevator	**5.9**	**5.9**	3.6	4.8
rrcs	5.5	**5.7**	1.4	3.3
achieve pattern	5.0	**5.9**	2.5	2.8
retraction pattern-1	4.1	4.0	2.7	**4.6**
retraction pattern-2	**6.1**	4.8	2.6	6.0
RG2	3.3	**5.2**	1.5	4.3
lily01	**5.1**	**5.1**	1.5	4.1
lily02	2.4	**3.8**	1.9	1.9
lily11	**7.1**	5.0	2.2	5.8
lily15	**6.1**	4.1	1.2	5.8
lily16	**3.5**	3.2	0.8	3.8
ltl2dba theta-2	1.9	**2.8**	1.9	1.2
ltl2dba R-2	1.0	**2.1**	1.9	**2.1**
simple arbiter icse2018	**3.8**	3.7	0.9	3.5

Table 3: ACoRe effectiveness in producing an exact or more general resolution than the manually written one.

Specification	NSGA-III	WBGA	AMOSA	Unguided
minepump	✓	✓	✓	✓
simple arbiter-v1				
simple arbiter-v2	✓	✓		
prioritized arbiter				
arbiter				
detector	✓	✓		✓
ltl2dba27				
round robin				

Second, the genetic algorithms (NSGA-III and WBGA) generate on average more (non-dominated) resolutions than AMOSA and unguided search. The results point out that WBGA generates more (non-dominated) resolutions than others in 13 out of the 25 cases, and NSGA-III is the one that produces more (non-dominated) resolutions in 11 cases. Considering the genetic algorithms together, we can observe that they outperform the AMOSA and unguided search in 21 out of the 25 cases, and coincide in one case (`ltl2dba R-2`). Finally, the Unguided Search generates more resolutions in 3 cases, namely, `detector`, TCP,

and `retraction-pattern-1`. Interestingly, the different algorithms of ACoRe produce on average between 1 and 8 non-dominated resolutions, which we consider is a reasonable number of options that the engineer can manually inspect and validate to select the most appropriate one.

ACoRe generates more non-dominated resolutions when adopting genetic algorithms. On average, ACoRe produces between 1 and 8 non-dominated resolutions that can be presented to the engineer for analysis and validation.

6.2   RQ2: Comparison with the Ground-truth

Table 3 presents the effectiveness of ACoRe in generating a resolution that is equivalent or more general than the ones manually developed by engineers. Overall, ACoRe is able to reproduce same resolutions in 3 out of 8 of the cases, namely, for the `minepump` (our running example), `simple arbiter-v2`, and `detector`. Like for RQ1, the genetic algorithms outperform AMOSA and unguided search in this respect. Particularly, the Unguided Search can replicate the resolution for the `detector` case, in which AMOSA fails.

Overall, the genetic algorithms can produce same or more general resolutions than the ground-truth in 3 out of the 8 cases, outperforming AMOSA (1 out of the 8) and unguided search (2 out of the 8).

6.3   RQ3: Comparing the Multi-objective Optimization Algorithms

For each set of non-dominated resolutions generated by the different algorithms, we compute the quality indicators HV and IGD for the syntactic and semantic similarity values. The reference point is the best possible value for each objective which is 1. These will allow us to determine which algorithm converges the most to the reference point and produces more diverse and optimal resolutions.
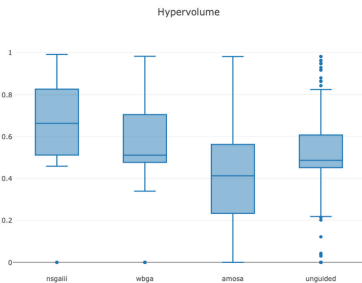

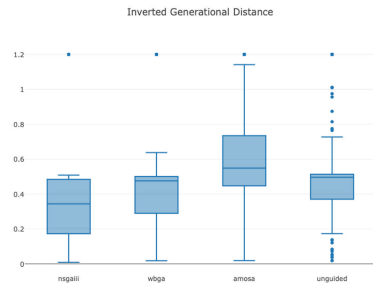Fig. 4: HV of the Pareto-optimal sets generated by ACoRe.

Fig. 5: IGD of the Pareto-optimal sets generated by ACoRe.

Figures 4 and 5 show the boxplots for each quality indicator. NSGA-III obtains on average much better HV and IGD than the rest of the algorithms. Precisely, it obtains on average 0.66 of HV (while higher the better) and 0.34 of IGD (while lower the better), outperforming the other algorithms.

To confirm this result we compare the quality indicators in terms of non-parametric statistical tests: (i) Kruskal–Wallis test by ranks and (ii) the Mann-Whitney U-test. The α value defined in the Kruskal-Wallis test by ranks is 0.05 and the Mann-Whitney U-test is 0.0125. Moreover, we also complete our assessment by using Vargha and Delaney's $\hat{A}_{12}$, a non-parametric effect size measurement. Table 4 summarises the results when we compare pair-wise each one of the approaches. We can observe that NSGA-III in near 80% of the cases obtains resolutions with better quality indicators than AMOSA and Unguided search (and the differences are statistically significant). We can also observe that NSGA-III obtains higher HV (IGD) than WBGA in 66% (65%) of the cases. From Table 4 we can also observe that WBGA outperforms both AMOSA and unguided search. Moreover, we can observe that AMOSA is the worse performing algorithm according to the considered quality indicators.

Table 4: HV and IGD quality indicators for the generated resolutions.

		WBGA		AMOSA		Unguided	
		HV	IGD	HV	IGD	HV	IGD
NSGAIII	p-value	< 0.00001	< 0.00001	< 0.00001	< 0.00001	< 0.00001	< 0.00001
	$\hat{A}_{12}$	0.66	0.65	0.84	0.83	0.80	0.76
WBGA	p-value	-	-	< 0.00001	< 0.00001	< 0.00001	< 0.00001
	$\hat{A}_{12}$	-	-	0.74	0.74	0.64	0.61
AMOSA	p-value	-	-	-	-	< 0.00001	< 0.00001
	$\hat{A}_{12}$	-	-	-	-	0.36	0.36

Overall, both statistical tests evidence that NSGA-III leads to a set of resolutions with better quality indicators (HV and IGD) than the rest of the algorithms. WBGA is the one in the second place, outperforming the unguided search and AMOSA. While AMOSA shows the lowest performance based on the quality indicators, even worse than the unguided search in several cases.

7 Related Work

Several manual approaches have been proposed to identify inconsistencies between goals and resolve them once the requirements were specified. Among them, Murukannaiah *et al.* [49] compares a genuine analysis of competing hypotheses against modified procedures that include requirements engineer thought process. The empirical evaluation shows that the modified version presents higher completeness and coverage. Despite the increase in quality, the approach is limited to manual applicability performed by engineers as well previous approaches [56].

Various informal and semi-formal approaches [28,32,33], as well as more formal approaches [21,23,26,30,51,53], have been proposed for detecting logically

inconsistent requirements, a strong kind of conflicts, as opposed to this work that focuses on a weak form of conflict, called divergences (cf. Section 3).

Moreover, recent approaches have been introduced to automatically identify goal-conflicts. Degiovanni *et al.* [18] introduced an automated approach where boundary conditions are automatically computed using a tableaux-based LTL satisfiability checking procedure. Since it exhibits serious scaliability issues, the work of Degiovanni *et al.* [17] proposes a genetic algorithm that mutates the LTL formulas in order to find boundary conditions for the goal specifications. The output of this approach can be fed into ACoRe to produce potential resolutions for the identified conflicts (as shown in the experimental evaluation).

Regarding specification repair approaches, Wang *et al.* [60] introduced ARepair, an automated tool to repair a faulty model formally specified in Alloy [31]. ARepair takes a faulty Alloy model and a set of failing tests and applies mutations to the model until all failing tests become passing. In the case of ACoRe, the identified goal conflicts are the ones that guide the search, and candidates are aimed to be syntactic and semantically similar to the original specification.

In the context of reactive synthesis [22,46,52], some approaches were proposed to repair imperfections in the LTL specifications that make the unrealisable (i.e., no implementation that satisfies the specification can be synthesized). The majority of the approaches focus on learning missing assumptions about the environment that make them unrealisable [4,10,11,48]. A more recent approach [8], published in a technical report, proposes to mutate both the assumptions and guarantees (goals) until the specification becomes realisable. Precisely, we use the novel model counting approximation algorithm from Brizzio et. al [8] to compute the semantic similarity between the original buggy specification and the resolutions. However, the notion of repair for Brizzio et. al [8] requires a realizable specification, which is very general and does not necessarily lead to quality synthesized controllers [20,47]. In this work, the definition of resolution is fine-grained and focused on removing the identified conflicts, which potentially leads to interesting repairs as we showed in our empirical evaluation.

Alrajeh *et al.* [2] introduced an automated approach to refine a goal model when the environmental context changes. That is, if the domain properties are changed, then this approach will propose changes in the goals to make them consistent with the new domain. The adapted goal model is generated using a new counterexample-guided learning procedure that ensures the correctness of the updated goal model, preferring more local adaptations and more similar goal models. In our work, the domain properties are not changed and the adaptions are made to resolve the identified inconsistencies, and instead of counterexamples, our search is guided by syntactic and semantic similarity metrics.

8   Conclusion

In this paper, we presented ACoRe, the first automated approach for goal-conflict resolution. Overall, ACoRe takes a goal specification and a set of conflicts previously identified, expressed in LTL, and computes a set of reso-

lutions that removes such conflicts. To assess and implement ACoRe that is a search-based approach, we adopted three multi-objective algorithms (NSGA-III, AMOSA, and WBGA) that simultaneously optimize and deal with the trade-off among the objectives. We evaluated ACoRe in 25 specifications that were written in LTL and extracted from the related literature. The evaluation showed that the genetic algorithms (NSGA-III and WBGA) typically generate more (non-dominated) resolutions than AMOSA and an Unguided Search we implemented as a baseline in our evaluation. Moreover, the algorithms generate on average between 1 and 8 resolutions per specification, which may allow the engineer to manually inspect and select the most appropriate resolutions. We also observed that the genetic algorithms (NSGA-III and WBGA) outperform AMOSA and Unguided Search in terms of several quality indicators: number of (non-dominated) resolutions and standard quality indicators (HV and IGD) for multi-objective algorithms.

References

1. Allen Troy Acree, Timothy Alan Budd, Richard A. DeMillo, Richard J. Lipton, and Frederick Gerald Sayward. Mutation analysis. techreport GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, Georgia, 1979.
2. Dalal Alrajeh, Antoine Cailliau, and Axel van Lamsweerde. Adapting requirements models to varying environments. In *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, May 23-29, 2020*, 2020.
3. Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastin Uchitel. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 265–275, Washington, DC, USA, 2009. IEEE Computer Society.
4. Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 26–33, 2013.
5. Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, page 1–10, New York, NY, USA, 2011. Association for Computing Machinery.
6. Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Trans. Evol. Comput.*, 12(3):269–283, 2008.
7. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.

8. Matías Brizzio, Renzo Degiovanni, Maxime Cordy, Mike Papadakis, and Nazareno Aguirre. Automated repair of unrealisable LTL specifications guided by model counting. *CoRR*, abs/2105.12595, 2021.

9. Antoine Cailliau and Axel van Lamsweerde. Handling knowledge uncertainty in risk-based requirements engineering. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*, pages 106–115, 2015.

10. Davide G. Cavezza and Dalal Alrajeh. Interpolation-based GR(1) assumptions refinement. *CoRR*, abs/1611.07803, 2016.

11. Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory*, pages 147–161, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

12. Carlos A. Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa, editors, *MICAI 2004: Advances in Artificial Intelligence*, pages 688–697, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

13. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. In *SCIENCE OF COMPUTER PROGRAMMING*, pages 3–50, 1993.

14. Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

15. Renzo Degiovanni, Dalal Alrajeh, Nazareno Aguirre, and Sebastián Uchitel. Automated goal operationalisation based on interpolation and sat solving. In *ICSE*, pages 129–139, 2014.

16. Renzo Degiovanni, Pablo F. Castro, Marcelo Arroyo, Marcelo Ruiz, Nazareno Aguirre, and Marcelo F. Frias. Goal-conflict likelihood assessment based on model counting. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 1125–1135, 2018.

17. Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. A genetic algorithm for goal-conflict identification. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 520–531, 2018.

18. Renzo Degiovanni, Nicolás Ricci, Dalal Alrajeh, Pablo F. Castro, and Nazareno Aguirre. Goal-conflict detection based on temporal satisfiability checking. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 507–518, 2016.

19. Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, 1978.

20. Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.*, 22(1):9, 2013.

21. Christian Ellen, Sven Sieverding, and Hardi Hungar. Detecting consistencies and inconsistencies of pattern-based functional requirements. In *Proc. of the 19th Intl. Conf. on Formal Methods for Industrial Critical Systems*, pages 155–169, 2014.

22. E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
23. Neil A. Ernst, Alexander Borgida, John Mylopoulos, and Ivan J. Jureta. Agile requirements evolution via paraconsistent reasoning. In *Proc. of the 24th Intl. Conf. on Advanced Information Systems Engineering*, pages 382–397, 2012.
24. Daniel Méndez Fernández, Stefan Wagner, Marcos Kalinowski, Michael Felderer, Priscilla Mafra, Antonio Vetro, Tayana Conte, M.-T. Christiansson, Des Greer, Casper Lassenius, Tomi Männistö, M. Nayabi, Markku Oivo, Birgit Penzenstadler, Dietmar Pfahl, Rafael Prikladnicki, Günther Ruhe, André Schekelmann, Sagar Sen, Rodrigo O. Spínola, Ahmet Tuzcu, Jose Luis de la Vara, and Roel Wieringa. Naming the pain in requirements engineering - contemporary problems, causes, and effects in practice. *Empirical Software Engineering*, 22(5):2298–2338, 2017.
25. Bernd Finkbeiner and Hazem Torfah. Counting models of linear-time temporal logic. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 360–371. Springer, 2014.
26. David Harel, Hillel Kugler, and Amir Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. In *Formal Methods in Software and Systems Modeling: Essays Dedicated to Hartmut Ehrig on the Occasion of His 60th Birthday*, pages 309–324, 2005.
27. Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, December 2012.
28. J.H. Hausmann, R. Heckel, and G. Taentzer. Detection of conflicting functional requirements in a use case-driven approach. In *ICSE*, pages 105–115, 2002.
29. John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992.
30. Anthony Hunter and Bashar Nuseibeh. Managing inconsistent specifications: Reasoning, analysis, and action. *ACM TOSEM*, 7(4):335–367, 1998.
31. Daniel Jackson. *Software Abstractions - Logic, Language, and Analysis.* MIT Press, 2006.
32. M. Kamalrudin. Automated software tool support for checking the inconsistency of requirements. In *ASE*, pages 693–697, 2009.
33. Massila Kamalrudin, John Hosking, and John Grundy. Improving requirements quality using essential use case interaction patterns. In *ICSE*, pages 531–540, 2011.
34. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.
35. J. Kramer, J. Magee, M. Sloman, and A. Lister. CONIC: an integrated approach to distributed computer control systems. *Computers and Digital Techniques, IEE Proceedings E*, 130(1):1+, 1983.
36. Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018.
37. William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

38. Maciej Laszczyk and Paweł B. Myszkowski. Survey of quality measures for multi-objective optimization: Construction of complementary set of multi-objective quality measures. *Swarm and Evolutionary Computation*, 48:109–133, 2019.
39. Emanuel Letier. Goal-oriented elaboration of requirements for a safety injection control system. Technical report, Université catholique de Louvain, 2002.
40. Emmanuel Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université catholique de Louvain, 2001.
41. Jianwen Li, Geguang Pu, Lijun Zhang, Yinbo Yao, Moshe Y. Vardi, and Jifeng He. Polsat: A portfolio LTL satisfiability solver. *CoRR*, abs/1311.1602, 2013.
42. Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Comput. Surv.*, 52(2), mar 2019.
43. Weilin Luo, Hai Wan, Xiaotong Song, Binhao Yang, Hongzhen Zhong, and Yin Chen. How to identify boundary conditions with contrasty metric? In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 1473–1484. IEEE, 2021.
44. H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.
45. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
46. Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984.
47. Shahar Maoz and Jan Oliver Ringert. On well-separation of GR(1) specifications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 362–372, 2016.
48. Shahar Maoz, Jan Oliver Ringert, and Rafi Shalom. Symbolic repairs for GR(1) specifications. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 1016–1026, 2019.
49. P.K. Murukannaiah, A.K. Kalia, P.R. Telangy, and M.P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *Proc. 23rd IEEE Int. Requirements Engineering Conf.*, pages 156–165, 2015.
50. Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. Redesigning the jmetal multi-objective optimization framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, page 1093–1100, New York, NY, USA, 2015. Association for Computing Machinery.
51. Tuong Huan Nguyen, Bao Quoc Vo, Markus Lumpe, and John Grundy. KBRE: a framework for knowledge-based requirements engineering. *Software Quality Journal*, 22(1):87–119, 2013.
52. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.
53. George Spanoudakis and Anthony Finkelstein. Reconciling requirements: a method for managing interference, inconsistency and conflict. *Annals of Software Engineering*, 3(1):433–457, 1997.
54. Ryoji Tanabe and Hisao Ishibuchi. An analysis of quality indicators using approximated optimal distributions in a 3-d objective space. *IEEE Trans. Evol. Comput.*, 24(5):853–867, 2020.

55. Axel van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications.* Wiley, 2009.
56. Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. Software Eng.*, 24(11):908–926, 1998.
57. Axel van Lamsweerde and Emmanuel Letier. Integrating obstacles in goal-driven requirements engineering. In *Proceedings of the 20th International Conference on Software Engineering*, ICSE '98, pages 53–62, Washington, DC, USA, 1998. IEEE Computer Society.
58. Axel van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, October 2000.
59. András Vargha and Harold D. Delaney. A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.
60. Kaiyuan Wang, Allison Sullivan, and Sarfraz Khurshid. Arepair: A repair framework for alloy. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 103–106, 2019.
61. Jiahui Wu, Paolo Arcaini, Tao Yue, Shaukat Ali, and Huihui Zhang. On the preferences of quality indicators for multi-objective search algorithms in search-based software engineering. *Empirical Softw. Engg.*, 27(6), nov 2022.
62. Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.