Università della Svizzera italiana | **Faculty of Informatics**

USI Technical Report Series in Informatics

Automated Discovery of Simulation Between Programs

Grigory Fedyukovich[1], Arie Gurfinkel[2], Natasha Sharygina[1]

[1] Formal Verification Lab of the Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
[2] SEI/CMU, USA

Abstract

The paper presents SIMABS, the first fully automated SMT-based approach to synthesize an abstraction of one program (called target) that simulates another program (called source). SIMABS iteratively traverses the search space of existential abstractions of the target and choses the strongest abstraction among them that simulates the source. Deciding whether a given relation is a simulation relation is reduced to solving validity of ∀∃-formulas iteratively. We present a novel algorithm for dealing with such formulas using an incremental SMT solver. In addition to deciding validity, our algorithm extracts witnessing Skolem relations which further drive simulation synthesis in SIMABS. Our evaluation confirms that SIMABS is able to efficiently discover both, simulations and abstractions, for C programs from the Software Verification Competition.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense. This material has been approved for public release and unlimited distribution. DM-0001771

Report Info

Published
October 2014

Updated
July 2015

Number
USI-INF-TR-2014-10

Institution
Faculty of Informatics
Università
della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 Introduction

There is a growing interest to the problems of regression verification and program equivalence checking [22, 24, 12, 15, 8, 21, 7, 27, 17]. In general, the problem is to identify the condition under which two programs (referred to as the *source* (S) and the *target* (T)) are equivalent (i.e., satisfy the same properties) and to check it. These approaches prevent the wasted efforts in re-analyzing equivalent parts of the programs. For instance, while proving safety of two closely related programs, obtaining a proof of one program and adapting it to another program can be more efficient than proving each program from scratch (e.g., [8, 7]).

For example, [7] applied the idea of adapting proofs to analyze whether compiler optimizations preserve safety properties. While efficient, this method had a number of limitations. The most crucial one is that it required a *mapping* between variables of S and T that was either guessed automatically or provided by the user. The simplest mapping that equates variables of S and T based on their names (i.e., variable a of S is mapped to a variable a of T) used in [7] is often insufficient. In practice, it sometimes results in no interesting facts lifted from S to T. As an example, consider compiler spilling that may introduce many new variable names in T that did not appear in S.

In another example application [20, 21, 11], it is shown that a simulation relation is the most general mapping to transfer proofs between S and T. However, discovering a simulation relation is difficult (e.g., [21] expects the candidate relation to be provided by the user). Moreover, their result only applies when T actually simulates S. The tasks of regression verification rely on cases when T is obtained by a small modification from S, but might not simulate S.

In this article, we address two problems: (1) the challenge of automatically constructing a simulation relation between two arbitrary programs, and (2) if the target T does not simulate the source S, the challenge of finding a strong abstraction of the target T that simulates the source S.

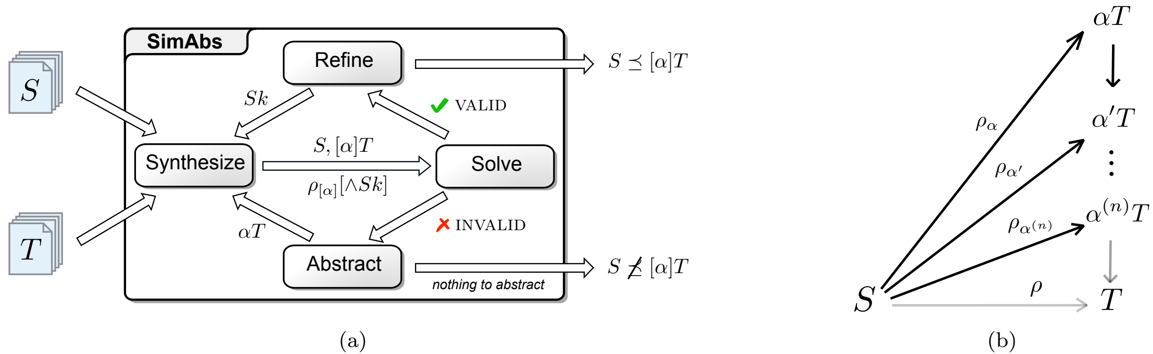


Figure 1: (a) SIMABS and (b) its search space.

Our main contribution is an iterative abstraction-refinement approach called SIMABS, illustrated in Fig. 1(a). The inputs are the source and the target programs S and T, respectively. The output is an abstraction of T (possibly T itself) that simulates S and a simulation relation ρ, or a step of S that cannot be simulated by any abstraction of T (in the considered space of abstractions). SIMABS first guesses a relation ρ between S and T (**Synthesize** step). Second, it checks whether ρ is a simulation relation between S and the current abstraction of T (**Solve** step). Third, if the check succeeds, it refines the current abstraction of T (**Refine** step) and synthesizes new relation ρ. Otherwise, it looks for a better abstraction α of T (**Abstract** step). The algorithm terminates when either no refinement or no abstraction is possible. The search space of the algorithm is shown in Fig. 1(b). SIMABS explores the space of abstractions of T, starting with the most general abstraction (called αT in Fig. 1(b)) that simulates S, and iteratively refines it until no further refinement is possible.

In contrast to existing algorithms for checking whether a given relation ρ is a simulation relation (e.g., [19, 14, 22]), we reduce the problem to deciding validity of $\forall\exists$-formula. Intuitively, the formula says "for each behavior of S there exists a corresponding behavior of $[\alpha]T$". Our second contribution is a novel decision procedure, AE-VAL, for deciding validity of $\forall\exists$-formulas over Linear Real Arithmetic (LRA). Our procedure is similar to the QE_SAT algorithm of [25]. However, in addition to deciding validity, we also extract a Skolem relation to witness the existential quantifier. This Skolem relation (called *Sk* in Fig. 1(a)) is the key to the **Refine** step of SIMABS.

We implemented SIMABS and AE-VAL on the top of the UFO framework [1, 16] and an SMT solver Z3 [5], respectively. We evaluated SIMABS by discovering simulation relations between programs and their LLVM optimizations. Our results show that SIMABS is able to efficiently find simulation relations between original and optimized program versions in both directions.

In the rest of the article, is structured as follows. We list the related work in Sect. 2. Then, after defining notation in Sect. 3, we describe how to reduce the problem of checking a simulation relation to a validity check of a specific $\forall\exists$-formula in Sect. 4. The algorithm AE-VAL designed to solve such formulas and extract Skolem relation is presented in Sect. 5. Sect. 6 provides implementation details for the algorithm SIMABS. Sect. 7 presents an evaluation of our implementation of SIMABS. Finally, Sect. 9 concludes the article.

2 Related Work

The notion of simulation relation between programs dates back to Milner [19]. Apart from the first algebraic formalization of simulation, this work also proposed an idea of abstracting some details from two programs in order to prove that they realize the same algorithm. Since then, this concept has been widely used in verification and other areas of computer science.

The first symbolic automatic construction of simulation relations was proposed by Dill et al. in [6]. However, that work was based on BDDs, so quantifiers are eliminated directly. We target to solve this problem by exploiting recent advancements in SMT and thus allowing synthesis of non-trivial simulation relations.

The classical approach to check simulation relations is *game-theoretic*: the state space of the source and the target programs is traversed by the evader and the pursuer players. For instance, Henzinger et al. [14] apply it to prove validity of a simulation relation between infinite graphs. In our setting, this result can be

used to extend SIMABS to deal with programs with different CPGs.

The problem of constructing and checking simulation relation arises in the area of translation validation. Necula [22] proposes to check automatically correctness of compiler optimizations by checking a simulation relation. He also proposes a simple heuristic to construct simulation relations. Namjoshi et al. in [21, 11] propose a more precise way to construct simulation relations, which requires augmenting a particular optimizer. In contrast, our SMT-based approach combines constructing and checking simulation relations in the same procedure, and is not restricted to any specific program transformer.

Ciobâcă et al. [4] develop a parametric proof system for proving mutual simulation between programs written in different programming languages. Strichman et al. [12] perform a program equivalence check in alternative way, based on Bounded Model Checking. Both approaches do not consider cases when there is only an abstract simulation, and when there is some other form of simulation relation rather than identity. Our approach goes beyond these limitations.

Simulation relation is a sort of relative specifications: it describes how the behaviors of programs relate to each other, but not how they behave individually. Inferring other types of relative specifications were studied in [17, 9]. Lahiri et al. [17] propose to search for *differential errors*: whether there exist two behaviors of S and T starting from the same input, such that the former is non-failing and the latter is failing. The proposed solution is by composing S and T into one program and running an off-the-shelf invariant generator on it. Felsing et.al [9] aims at synthesizing *coupling predicates*, a stronger relationship than a simulation relation, since it does not allow programs to have unmatched behaviors. In contrast to SIMABS, their synthesis method is restricted to deal only with deterministic and terminating programs and does not require quantifier elimination.

Apart from discovering simulation between programs, there exist other symbolic techniques to prove equivalence between the programs. For example, [3] checks equivalence of a Verilog circuit and a C program through encoding and solving quantifier-free SAT formula. [12, 15] attempt to perform the localized check on the function-call-tree level of C programs. While checking absolute equivalence is hard, there is a group of techniques [27, 8, 7] that check partial (or relative) equivalence with respect to some assertion (or a set of assertions). For example, [7] tries to adapt safe inductive invariants corresponding to the nodes of the program's Cut Point Graph (CPG) [13]. In Sect. 6 of the paper, we adapt the discovery of inductive simulation relation for the CPG edges, providing the mapping between variables to be potentially useful in [7].

3 Background and Notation

In this section, we give the necessary definitions of a program, a transition system, and a simulation relation. We use vector notation to denote sets of variables and set-theoretic operations (e.g., subset $\vec{u} \subseteq \vec{x}$, complement $\vec{x}_{\vec{u}} = \vec{x} \setminus \vec{u}$, union $\vec{x} = \vec{u} \cup \vec{x}_{\vec{u}}$). Throughout the article we assume all free variables are implicitly universally quantified. For example, we write $\varphi(\vec{x})$ for $\forall \vec{x} \,.\, \varphi(\vec{x})$.

Definition 1. *A* program *P is a tuple $\langle Var, Init, Tr\rangle$, where $Var \equiv V \cup L \cup V'$ is a set of* input, local, *and* output variables*; $Init$ is a formula over V encoding the* initial states*; and Tr is a formula over Var encoding the* transition relation.

A state $\vec{s} \in \mathscr{S}$ is a valuation to all variables in V. While Tr encodes entire computation between initial and final states, the values of variables in L explicitly capture all intermediate states along the computation. If for states $\vec{s}, \vec{s}'$, there exists a valuation $\vec{l}$ to local variables in L, such that $\vec{s}, \vec{l}, \vec{s}' \models Tr$, we call the pair $(\vec{s}, \vec{s}')$ *computable*. V' is used to denote the values of variables of V at the end of the computation. Throughout, we write $\vec{s}'$ for $\vec{s}(x')$ and $\mathscr{S}'$ for $\{\vec{s}' \mid \vec{s} \in \mathscr{S}\}$.

Definition 2. *Given a program $P = \langle Var, Init, Tr\rangle$, a* transition system *$\mathscr{T}(P) = \langle \mathscr{S}, \mathscr{I}, \mathscr{R}\rangle$, where $\mathscr{I} = \{\vec{s} \in \mathscr{S} \mid \vec{s} \models Init\}$ is the set of* initial states*, $\mathscr{R} = \{(\vec{s}, \vec{s}') \mid \vec{s} \in \mathscr{S}, \vec{s}' \in \mathscr{S}' \,.\, (\vec{s}, \vec{s}')$ is computable$\}$ is a* transition relation.

Throughout, we use programs and their transition systems interchangeably.

Definition 3. *Program $P_1 = \langle V_1 \cup L_1 \cup V', Init_1, Tr_1\rangle$ is an* abstraction *of program $P_2 = \langle V_2 \cup L_2 \cup V_2', Init_2, Tr_2\rangle$ iff* (1) $V_2 \subseteq V_1$, (2) $Init_2 \implies Init_1$, (3) *each $(\vec{s}, \vec{s}')$, that computable in Tr_2, is also computable in Tr_1.*

An example of abstraction is shown in Fig. 2(b)-2(c), where non-determinism is introduced by making an input variable of the while-loop local.

```
int a = *; int b = *;
while(*){
  a = a + b;
}
```

(a) The source

```
int a = *; int b = *;
while(*){
  int c = a - b;
  a = c;
}
```

(b) The target

```
int a = *;
while(*){
  int b = *;
  int c = a - b;
  a = c;
}
```

(c) Abstraction of the target

Figure 2: Three programs in C.

Definition 4. *Given transition systems S and T, a* left-total *relation $\rho \subseteq \mathscr{S}_S \times \mathscr{S}_T$ is a* simulation relation *if* (1) *every state in $\mathscr{I}_S$ is related by ρ to some state in $\mathscr{I}_T$, and* (2) *for all states $\vec{s}$, $\vec{s}'$ and $\vec{t}$, such that $(\vec{s},\vec{t}) \in \rho$ and $(\vec{s},\vec{s}') \in \mathscr{R}_S$ there is some state $\vec{t}'$, such that $(\vec{t},\vec{t}') \in \mathscr{R}_T$ and $(\vec{s}',\vec{t}') \in \rho$.*

We write, $T_1 \preceq_\rho T_2$, to denote that a transition system T_1 is simulated by a transition system T_2 via a simulation relation ρ. We write $T_1 \preceq T_2$ to indicate existence of a simulation between T_1 and T_2. Identity relation *id* i.e., pairwise-equivalence of values of common variables, is an example of ρ.

Lemma 1. *If P_2 is an abstraction of P_1 then $P_1 \preceq_{id} P_2$.*

Lemma 2. *If $P_1 \preceq P_2$ and $P_2 \preceq P_3$ then $P_1 \preceq P_3$.*

Thus, each program S is simulated by the *universal abstraction* $\mathbb{U}$ of any other program T (which is in fact the only common abstraction to all possible programs). In our approach we will algorithmically disqualify such cases and aim at synthesizing total simulation relations to provide the strong level of equivalence between S and (an abstraction of) T.

Note that the programs in scope of the article are not required to have an *error* location. Consequently, the approach proposed in the following sections is not limited to dealing only with safe programs.

4 From Simulation to Validity

In this section, we show that deciding whether a given relation ρ is a simulation relation is reducible to deciding validity of $\forall\exists$-formulas. We then show how Skolem functions witnessing the validity of the quantifiers are used to refine ρ.

4.1 Deciding Simulation Symbolically

Let $S(\vec{s},\vec{x},\vec{s}')$ and $T(\vec{t},\vec{y},\vec{t}')$ be formulas encoding transition relations of two programs, where $\vec{s}$ and $\vec{t}$, $\vec{s}'$ and $\vec{t}'$, $\vec{x}$ and $\vec{y}$ are input, output, and local variables, respectively. Let $Init_S(\vec{s})$ and $Init_T(\vec{t})$ be formulas encoding the initial states of S and T, respectively. Let $\rho(\vec{s},\vec{t})$ denote a left-total relation between variables of S and T. For simplicity, we omit the arguments and simply write S, T, and ρ, when the arguments are clear from the context.

Lemma 3. *A relation ρ is a simulation relation between S and T iff*

$$Init_S(\vec{s}) \implies \exists \vec{t} \,.\, \rho(\vec{s},\vec{t}) \land Init_T(\vec{t}) \tag{1}$$

$$\rho(\vec{s},\vec{t}) \land \exists \vec{x} \,.\, S(\vec{s},\vec{x},\vec{s}') \implies \exists \vec{t}', \vec{y} \,.\, T(\vec{t},\vec{y},\vec{t}') \land \rho(\vec{s}',\vec{t}') \tag{2}$$

Implication (1) reflects the matching of initial states of S and T by ρ. The left-hand-side of implication (2) reflects the set of all behaviors of S and the set of all input conditions matched by ρ. The right-hand-side of (2) reflects the existence of a behavior in T and an output condition matched by ρ.

Example 1. *Consider two programs in Fig. 2(a) and Fig. 2(b). For demonstration purposes, we focus on the corresponding loop bodies in an arbitrary iteration.*[1] *Assume, the input variables are assigned to some*

[1] Looking ahead, note that checking simulation relation between generic programs with non-trivial Control-Flow-Graphs in our approach is performed in several phases. While Example 1 demonstrates the phase of checking simulation relation between loop bodies, the whole-program analysis is demonstrated in Example 5 in Sect. 6.

constants (X, Y), as in (3), so the computation starts at the identical states. The fragments of the transition relation corresponding to the single loop body are encoded into (4):

$$Init_S \equiv (a_S = X) \wedge (b_S = Y) \qquad Init_T \equiv (a_T = X) \wedge (b_T = Y) \tag{3}$$

$$S \equiv (a'_S = a_S + b_S) \qquad T \equiv (c_T = a_T - b_T) \wedge (a'_T = c_T) \tag{4}$$

where the subscript indicates in which program the variables are defined.

Let ρ be a relation between variables of S and T, defined in (5):

$$\rho \equiv (a_S = a_T) \wedge (b_S = b_T) \qquad \rho' \equiv (a'_S = a'_T) \wedge (b_S = b_T) \tag{5}$$

ρ is a simulation relation iff the two following formulas are valid:

$$Init_S \implies \exists a_T, b_T \,.\, Init_T \wedge \rho \qquad \rho \wedge S \implies \exists c_T, a'_T \,.\, T \wedge \rho' \tag{6}$$

Note that since T is deterministic, the existential quantifiers in (6) are eliminated trivially by substitution. In our example, the left implication of (6) is valid, but the right implication of (6) simplifies to $0 = 1$. Hence, $S \not\preceq_\rho T$.

4.2 Abstract Simulation

There is a practical significance to check whether a program S is simulated by an abstraction αT of a program T via relation ρ_α in cases when the complete simulation is not proven. Our key result is to show that such abstract-simulation checking can be done without constructing an abstraction explicitly.

We restrict our attention to existential abstraction [2], although the results extend to predicate abstraction as well. An *existential abstraction*, $\alpha^{\exists}_{\vec{u}}$, of T abstracts a subset of variables $\vec{u}$ from T. Formally, given $\vec{u} \subseteq \vec{t}$, $Init_{\alpha^{\exists}_{\vec{u}}(T)} \equiv \exists \vec{u} \,.\, Init_T(\vec{t})$, and $\alpha^{\exists}_{\vec{u}}(T) \equiv \exists \vec{u}, \vec{u}' \,.\, T(\vec{t}, \vec{y}, \vec{t'})$. For example, the program in Fig. 2(c) is an existential abstraction of the program in Fig. 2(b), where $\vec{u} = \{b\}$.

Deciding whether S is simulated by $\alpha^{\exists}_{\vec{u}}(T)$ via $\rho_\alpha(\vec{s}, \vec{t}_{\vec{u}})$ (where $\vec{t}_{\vec{u}}$ is the complement of $\vec{u}$ in $\vec{t}$) can be done if the variables $\vec{u}$ are treated as locals in T.

Lemma 4. *A relation ρ_α is a simulation relation between S and $\alpha^{\exists}_{\vec{u}}(T)$ iff*

$$Init_S(\vec{s}) \implies \exists \vec{t}_{\vec{u}}, \vec{u} \,.\, \rho_\alpha(\vec{s}, \vec{t}_{\vec{u}}) \wedge Init_T(\vec{t}) \tag{7}$$

$$\rho_\alpha(\vec{s}, \vec{t}_{\vec{u}}) \wedge \exists \vec{x} \,.\, S(\vec{s}, \vec{x}, \vec{s'}) \implies \exists \vec{u}, \vec{y}, \vec{t'}_{\vec{u}'}, \vec{u}' \,.\, T(\vec{t}, \vec{y}, \vec{t'}) \wedge \rho_\alpha(\vec{s'}, \vec{t'}_{\vec{u}'}) \tag{8}$$

Recall that in Example 1, the loop-body S was shown to be not simulated by the loop-body T via identity relation. Interestingly, this result is still useful to obtain a simulation relation between S and T by creating an implicit abstraction of T and further refining it. We demonstrate this 2-steps procedure in Example 2.

Example 2. *As the first (*abstraction*) step, we create the abstraction of T by choosing a variable (say b) to be existentially quantified. Note that the produced abstraction is equivalent to the program in Fig. 2(c). Instead of encoding initial states $Init_{\alpha T}$ and a transition system αT from scratch (similarly to (3) and (4)), we let $Init_{\alpha T} \equiv \exists b_T \,.\, Init_T$ and $\alpha T \equiv \exists b_T \,.\, T$.*

Relation ρ (disproven to be a simulation between S and T) is abstracted away in correspondence with αT:

$$\rho_\alpha \equiv (a_S = a_T) \qquad \rho'_\alpha \equiv (a'_S = a'_T) \tag{9}$$

ρ_α is a simulation relation between S and αT iff the following formulas are valid:

$$Init_S \implies \exists a_T, b_T \,.\, Init_T \wedge \rho_\alpha \qquad \rho_\alpha \wedge S \implies \exists c_T, a'_T, b_T \,.\, T \wedge \rho'_\alpha \tag{10}$$

Clearly, (10) are valid iff there is a Skolem function for the existentially quantified variable b_T. Note that $sk_{b_T}(b_S) = -b_S$ is such function:

$$Init_S \implies (b_T = -b_S) \implies \exists a_T \,.\, Init_T \wedge \rho_\alpha \tag{11}$$

$$\rho_\alpha \wedge S \implies (b_T = -b_S) \implies \exists c_T, a'_T \,.\, T \wedge \rho'_\alpha$$

*As the second (*refinement*) step, sk_{b_T} is used to strengthen the simulation relation ρ_α between S and αT to become (12).*

$$\rho^{ext}_\alpha \equiv (a_S = a_T) \wedge (b_S = -b_T) \qquad \rho'^{ext}_\alpha \equiv (a'_S = a'_T) \wedge (b_S = -b_T) \tag{12}$$

Note that ρ^{ext}_α is a simulation relation between S and T.

The detailed definition of the Skolem function, its generalization and application to the simulation-relation-checking problem is given in Sect. 4.3.

4.3 Refining Simulation by Skolem Relations

We begin with a definition of a Skolem relation:

Definition 5. *Given a formula* $\exists y\,.\,f(x,y)$*, a relation* $Sk_y(x,y)$ *is a* Skolem relation *for* y *iff* (1) $Sk_y(x,y) \implies f(x,y)$, (2) $\exists y\,.\,Sk_y(x,y) \iff \exists y\,.\,f(x,y)$.

In Def. 5, we allow Sk_y to be a relation between x and y such that (1) Sk_y maps each x to a value of y that makes f true, (2.1) if for a given x, Sk_y maps x to some value of y then there is a value of y that makes f valid for this value of x, (2.2) if for a given x, there is a value of y such that f holds, then Sk_y is not empty. A Skolem relation Sk_y is *functional* iff it is given by the form $Sk_y(x,y) \equiv y = f_y(x)$ (also known as *Skolem function*, as in [26]). $Sk_{\vec{y}}$ is *Cartesian* if it is a Cartesian product of Skolem relations of individual variables from $\vec{y}$. $Sk_{\vec{y}}$ is guarded if it is a guarded disjunction of Cartesian Skolem relations.

In other words, validity of a ∀∃-formula is equivalent to existence of an appropriate total Skolem relation. As sketched in Example 2, our use of a Skolem relation Sk witnessing the validity of the formulas (7,8) is to refine an abstract simulation relation ρ_α to $\rho_\alpha^{ext} = \rho_\alpha \wedge Sk$. However, ρ_α^{ext} is guaranteed to be a simulation relation only in case if the corresponding formulas (7,8) are valid, thus requiring an extra simulation-check.

Theorem 1. *Let* $S(\vec{s},\vec{x},\vec{s}')$ *and* $T(\vec{t},\vec{y},\vec{t}')$ *be two programs, such that* $S \preceq_\rho T$*. Let* $\vec{u}$ *be a subset of variables in* T *(i.e.,* $\vec{u} \subseteq \vec{t}$*), such that* $S \preceq_{\rho_\alpha} \alpha^{\exists}_{\vec{u}}(T)$ *and* $\rho \implies \rho_\alpha$*. Then, there exists a relation* $Sk(\vec{s},\vec{u})$ *such that* (1) $\rho_\alpha \wedge Sk$ *is a simulation relation between* S *and* T *and* (2) Sk *is a Skolem relation for* $\vec{u}$ *in* (7) *and* (8).

Proof. Let $\vec{t}_{\vec{u}}$ be the complement of $\vec{u}$ in $\vec{t}$, and $\vec{t}'_{\vec{u}'}$ be the complement of $\vec{u}'$ in $\vec{t}'$. Having in mind $\rho \implies \rho_\alpha$, let Sk be a left-total relation over $\vec{s}$, $\vec{s}'$, $\vec{u}$ and $\vec{u}'$, such that:

$$\begin{aligned} \rho(\vec{s},\vec{t}) &\equiv \rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge Sk(\vec{s},\vec{u}) \\ \rho(\vec{s}',\vec{t}') &\equiv \rho_\alpha(\vec{s}',\vec{t}'_{\vec{u}}) \wedge Sk(\vec{s}',\vec{u}') \end{aligned} \tag{13}$$

Substituting (13) into (1) and (2), we get (14) and (15) respectively:

$$Init_S(\vec{s}) \implies \exists \vec{t}_{\vec{u}},\vec{u}\,.\,\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge Sk(\vec{s},\vec{u}) \wedge Init_T(\vec{t}) \tag{14}$$

$$\begin{aligned} &\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge Sk(\vec{s},\vec{u}) \wedge S(\vec{s},\vec{s}') \implies \\ &\quad \exists \vec{t}'_{\vec{u}},\vec{u}',\vec{y}\,.\,T(\vec{t}_{\vec{u}},\vec{u},\vec{y},\vec{t}'_{\vec{u}'},\vec{u}') \wedge \rho_\alpha(\vec{s}',\vec{t}'_{\vec{u}'}) \wedge Sk(\vec{s}',\vec{u}') \end{aligned} \tag{15}$$

Then from (14) and (15), it easy to see that $Sk(\vec{s},\vec{u})$ satisfies Def. 5 (it is a Skolem relation) for $\vec{u}$, respectively in (16) and in (17).

$$Init_S(\vec{s}) \implies \exists \vec{t}_{\vec{u}},\vec{u}\,.\,\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge Sk(\vec{s},\vec{u}) \wedge Init_T(\vec{t}) \tag{16}$$

$$\begin{aligned} &\exists \vec{u}\,.\,\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge S(\vec{s},\vec{s}') \implies \\ &\quad \exists \vec{t}'_{\vec{u}},\vec{u}',\vec{y}\,.\,T(\vec{t}_{\vec{u}},\vec{u},\vec{y},\vec{t}'_{\vec{u}'},\vec{u}') \wedge \rho_\alpha(\vec{s}',\vec{t}'_{\vec{u}'}) \wedge Sk(\vec{s}',\vec{u}') \end{aligned} \tag{17}$$

Consequently, $Sk(\vec{s},\vec{u})$ is a Skolem relation for $\vec{u}$ in (19), that is logically weaker than (17); and in (18), that is logically weaker than (16).

$$Init_S(\vec{s}) \implies \exists \vec{t}_{\vec{u}},\vec{u}\,.\,\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge Init_T(\vec{t}) \tag{18}$$

$$\exists \vec{u}\,.\,\rho_\alpha(\vec{s},\vec{t}_{\vec{u}}) \wedge S(\vec{s},\vec{s}') \implies \exists \vec{t}'_{\vec{u}'},\vec{u},\vec{u}',\vec{y}\,.\,T(\vec{t}_{\vec{u}},\vec{u},\vec{y},\vec{t}'_{\vec{u}},\vec{u}') \wedge \rho_\alpha(\vec{s}',\vec{t}'_{\vec{u}}) \tag{19}$$

Finally, (18) and (19) are respectively equivalent to the simulation-checking-formulas (7) and (8) for S and $\alpha^{\exists}_{\vec{u}}(T)$.

In the next section we will show how formulas (1), (2), (7), and (8) can be solved by iterative quantifier elimination. On the top of it, we will demonstrate the procedure for discovering guarded Skolem relations.

5 Validity and Skolem Extraction

We present a novel algorithm, AE-VAL, for deciding validity of ∀∃-formulas and constructing witnessing Skolem relations. Without loss of generality, we restrict the input formula to have the form $S(\vec{x}) \implies \exists \vec{y} \,.\, T(\vec{x}, \vec{y})$, where S has no universal quantifiers, and T is quantifier-free.

5.1 MBP for Linear Rational Arithmetic

Our algorithm is based on a notion of Model-Based Projection (MBP), introduced in [16], that under-approximates existential quantification. An *MBP* is a function from models of the formula $T(\vec{x}, \vec{y})$ to quantifier-free formulas $T_{\vec{y}}(\vec{x})$ iff:

$$\forall m \models T(\vec{x}, \vec{y}) \,.\, m \models MBP(m) \tag{20}$$

$$\forall m \models T(\vec{x}, \vec{y}) \,.\, MBP(m) \implies \exists \vec{y} \,.\, T(\vec{x}, \vec{y}) \tag{21}$$

That is, the only $\vec{x}$ variables appear in the image $MBP(m)$, m is a model of $MBP(m)$, and $MBP(m)$ is an implicant of $\exists \vec{y} \,.\, T(\vec{x}, \vec{y})$. Furthermore, for fixed $\vec{y}$ and T, the function *MBP* is finite. Consequently, there are finitely many projections $T_{\vec{y}_1}(\vec{x}), T_{\vec{y}_2}(\vec{x}), \ldots, T_{\vec{y}_n}(\vec{x})$, such that $\exists \vec{y} \,.\, T(\vec{x}, \vec{y}) = \bigvee_{i=1}^{n} T_{\vec{y}_i}(\vec{x})$ for some n. In our implementation, we are using an *MBP* function from [16] for Linear Rational Arithmetic (LRA) that is based on Loos-Weispfenning [18] quantifier elimination.

Loos-Weispfenning (LW) method [18] applies for quantifier elimination in LRA (22). Consider formula $\exists \vec{y} \,.\, T(\vec{x}, \vec{y})$, where T is quantifier-free. Following the simplified presentation by Nipkow [23], $\vec{y}$ is singleton, T is in Negation Normal Form, and y appears in the literals only of the form $y = e$, $l < y$ or $y < u$, where l, u, e are y-free.

$$\exists y \,.\, T(\vec{x}) \equiv \Big(\bigvee_{(y=e)\in lits(T)} T[e] \vee \bigvee_{(l<y)\in lits(T)} T[l+\epsilon] \vee T[-\infty] \Big) \tag{22}$$

In (22), $lits(T)$ denote the set of literals of T, $T[\cdot]$ stands for a *virtual substitution* for the literals containing y. In particular, $T[e]$ substitutes exact values of y ($y = e$), $T[l+\epsilon]$ substitutes the intervals ($l < y$) of possible values of y, $T[-\infty]$ substitutes the rest of the literals. Consequently, a function $LRAProj_T$ is an implementation of the *MBP* function for (22):

$$LRAProj_T(M) = \begin{cases} T[e], & \text{if } (y = e) \in lits(T) \wedge M \models (y = e) \\ T[l+\epsilon], & \text{else if } (l < y) \in lits(T) \wedge M \models (l < y) \wedge \\ & \forall (l' < y) \in lits(T) \,.\, M \models ((l' < y) \implies (l' \leq l)) \\ T[-\infty], & \text{otherwise} \end{cases} \tag{23}$$

Additionally, we assume that for each projection $T_{\vec{y}_i}$, the *MBP* procedure gives a condition ϕ_i under which T is equisatisfiable with $T_{\vec{y}_i}$:

$$\phi_i(\vec{x}, \vec{y}) \implies \big(T_{\vec{y}_i}(\vec{x}) \iff T(\vec{x}, \vec{y})\big) \tag{24}$$

Such a relation ϕ_i is a natural by-product of the *MBP* algorithm in [16]. Intuitively, each ϕ_i captures the substitutions made in T to produce $T_{\vec{y}_i}$. We assume that each ϕ_i is in the Cartesian form, i.e., a conjunction of terms, in which each $y \in \vec{y}$ appears at most once. That is, for $y \in \vec{y}$ and $\sim \in \{<, \leq, =, \geq, >\}$,

$$\phi_i(\vec{x}, \vec{y}) = \bigwedge_{y\in\vec{y}} \big(y \sim f_y(\vec{x})\big) \tag{25}$$

We write $(T_{\vec{y}_i}, \phi_i) \leftarrow \text{GETMBP}(\vec{y}, m, T(\vec{x}, \vec{y}))$ for an *MBP* algorithm that takes a formula T, a model m of T and a vector of variables $\vec{y}$, and returns a projection $T_{\vec{y}_i}$ of T that covers m and the corresponding relation ϕ_i.

5.2 Deciding Validity of ∀∃-formulas

AE-VAL is shown in Alg. 1. Given two formulas $S(\vec{x})$ and $\exists \vec{y} \,.\, T(\vec{x}, \vec{y})$, it decides validity of $S(\vec{x}) \implies \exists \vec{y} \,.\, T(\vec{x}, \vec{y})$. AE-VAL enumerates the models of $S \wedge T$ and blocks them from S. In each iteration, it first checks whether S is non-empty (line 3) and then looks for a model m of $S \wedge T$ (line 7). If m is found, AE-VAL gets a projection

Algorithm 1: AE-VAL $\left(S(\vec{x}), \exists \vec{y} \,.\, T(\vec{x},\vec{y})\right)$

Input: $S(\vec{x}), \exists \vec{y} \,.\, T(\vec{x},\vec{y})$
Output: $res \in \{$VALID, INVALID$\}$ of $S(\vec{x}) \Longrightarrow \exists \vec{y} \,.\, T(\vec{x},\vec{y})$
Data: Incremental SMTSOLVER, model m, MBP $T_{\vec{y}}(\vec{x})$, condition $\phi(\vec{x},\vec{y})$

1 SMTADD($S(\vec{x})$);
2 **forever do**
3   **if** (ISUNSAT(SMTSOLVE())) **then return** VALID;
4   SMTPUSH();
5   SMTADD($T(\vec{x},\vec{y})$);
6   **if** (ISUNSAT(SMTSOLVE())) **then return** INVALID;
7     $m \leftarrow$ SMTGETMODEL();
8     $(T_{\vec{y}}, \phi(\vec{x},\vec{y})) \leftarrow$ GETMBP($\vec{y}, m, T(\vec{x},\vec{y})$));
9   SMTPOP();
10   SMTADD($\neg T_{\vec{y}}$);

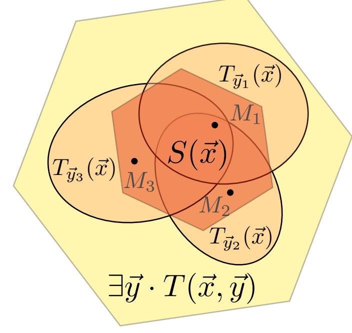


Figure 3: A Venn diagram for formulas from Example 3 ($S, \exists T$ – hexagons, MBPs – ovals, models – points).

$T_{\vec{y}}$ of T based on m (line 8) and blocks all models contained in $T_{\vec{y}}$ from S (line 10). The algorithm iterates until either it finds a model of S that can not be extended to a model of T (line 6), or all models of S are blocked (line 3). In the first case, the input formula is invalid. In the second case, every model of S has been extended to some model of T, and the formula is valid.

Possible three iterations of AE-VAL are depicted graphically in Fig. 3. In the first iteration, AE-VAL selects a model m_1 and generalizes it to a projection $MBP(m_1) = T_{\vec{y}_1}$. Then, it picks a model m_2 that is not contained in $T_{\vec{y}_1}$ and generalizes it to $MBP(m_2) = T_{\vec{y}_2}$. Finally, it picks a model m_3 that is contained neither in $T_{\vec{y}_1}$ nor in $T_{\vec{y}_2}$, and generalizes it to $MBP(m_3) = T_{\vec{y}_3}$. At this point, all models of S are covered by $\vec{y}$-free implicants of $\exists \vec{y} \,.\, T(\vec{x},\vec{y})$, and the algorithm terminates. We demonstrate this further in the following example.

Example 3. *Let $\vec{x} \equiv \{a, b\}$, $\vec{y} \equiv \{a', b'\}$, and S and T be defined as follows:*

$$S \equiv (a = b + 2)$$
$$T \equiv (a' > a) \wedge (b = 1 \implies b' = b) \wedge (b = 2 \implies b' > b) \wedge (b = 3 \implies b' < b)$$

We use Φ_i to denote the formula in the SMT context at the beginning of iteration i of AE-VAL. *Initially, $\Phi_1 = S$. The first model is $m_1 \equiv \{a = 0, b = -2, a' = 1/2, b' = -5/2\}$.* GETMBP$(\vec{y}, m_1, T)$ *returns:*

$$T_1 \equiv (b \neq 1) \wedge (b \neq 2) \qquad\qquad \phi_1 \equiv (a' > a) \wedge (b' < b)$$

In the second iteration, $\Phi_2 = \Phi_1 \wedge \neg T_1$, $m_2 \equiv \{a = 4, b = 2, a' = 9/2, b' = 5/2\}$, and GETMBP$(\vec{y}, m_2, T)$ *returns:*

$$T_2 \equiv (b \neq 1) \wedge (b \neq 3) \qquad\qquad \phi_2 \equiv (a' > a) \wedge (b' > b)$$

In the third iteration, $\Phi_3 = \Phi_2 \wedge \neg T_2$, $m_3 \equiv \{a = 3, b = 1, a' = 7/2, b' = 1\}$, and GETMBP$(\vec{y}, m_3, T)$ *returns:*

$$T_1 \equiv (b \neq 2) \wedge (b \neq 3) \qquad\qquad \phi_3 \equiv (a' > a) \wedge (b' = b)$$

Since $\Phi_4 = \Phi_3 \wedge \neg T_3$ is unsatisfiable, AE-VAL *returns* VALID *and terminates.*

5.3 Extracting Skolem Relation

AE-VAL is designed to construct a Skolem relation $Sk_{\vec{y}}(\vec{x},\vec{y})$, that maps each model of $S(\vec{x})$ to a corresponding model of $T(\vec{x},\vec{y})$. We use a set of projections $\{T_{\vec{y}_i}(\vec{x})\}$ for $T(\vec{x},\vec{y})$ and the set of conditions $\{\phi_i(\vec{x},\vec{y})\}$ that make the corresponding projections equisatisfiable with $T(\vec{x},\vec{y})$.

Lemma 5. *For each i, the relation $\phi_i(\vec{x},\vec{y})$ is a Skolem relation for $\vec{y}$ in formula $S(\vec{x}) \wedge T_{\vec{y}_i}(\vec{x}) \implies \exists \vec{y} \,.\, T(\vec{x},\vec{y})$.*

Proof. Follows from (21), (24), and (25). Recall, by definition (24), $\phi_i(\vec{x},\vec{y}) \implies \left(T_{\vec{y}_i}(\vec{x}) \iff T(\vec{x},\vec{y})\right)$. Unfolding Def. 5, we need to show that:
1) $\phi_i(\vec{x},\vec{y}) \implies (S(\vec{x}) \wedge T_{\vec{y}_i}(\vec{x}) \implies T(\vec{x},\vec{y}))$ – by definition (24).
2) $\exists y \,.\, \phi_i(\vec{x},\vec{y}) \iff (S(\vec{x}) \wedge T_{\vec{y}_i}(\vec{x}) \implies \exists \vec{y} \,.\, T(\vec{x},\vec{y}))$ – since by construction (25), ϕ_i is total relation, and an implication in the definition of MBP (21) is valid.

Intuitively, ϕ_i maps each model of the subset $S \wedge T_{\vec{y}_i}$ of S to T. Moreover, ϕ_i are not disjoint (e.g., see Fig. 3). Thus, to define the Skolem relation *Sk*, we need to address two issues: (1) we need to find a partitioning $\{I_i\}_{i=1}^n$ of S, and (2) each partition must be associated with an appropriate ϕ_i.

The constraints on the partitions I_i are as follows. First, a partition I_i must cover all models of $T_{\vec{y}_i}$ that are not already covered by other elements of I_i. Second, it should not include any models that are not contained in $T_{\vec{y}_i}$. Writing these requirements formally, we get the following system of constraints:

$$\begin{cases} S(\vec{x}) \wedge T_{\vec{y}_1}(\vec{x}) \implies I_1(\vec{x}) \\ S(\vec{x}) \wedge T_{\vec{y}_2}(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \implies I_2(\vec{x}) \\ \ldots \\ S(\vec{x}) \wedge T_{\vec{y}_n}(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \wedge \neg T_{\vec{y}_2}(\vec{x}) \wedge \ldots \wedge \neg T_{\vec{y}_{n-1}}(\vec{x}) \implies I_n(\vec{x}) \\ S(\vec{x}) \wedge I_1(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \implies \bot \\ \ldots \\ S(\vec{x}) \wedge I_n(\vec{x}) \wedge \neg T_{\vec{y}_n}(\vec{x}) \implies \bot \end{cases} \tag{26}$$

Note that in (26), S and $\{T_{\vec{y}_i}\}$ are first-order formulas, and $\{I_i\}$ are uninterpreted predicates. The set of constrains corresponds to a set of recursion-free Horn clauses. Thus, we can find an interpretation of the predicates $\{I_i\}$ using a Horn-clause solver. In our implementation, we use the solver of Z3, but other solutions, for example, based on interpolation, are also possible.

We now define the guarded Skolem relation $Sk_{\vec{y}}(\vec{x}, \vec{y})$ as follows:

$$Sk_{\vec{y}}(\vec{x}, \vec{y}) \equiv \begin{cases} \phi_{\vec{y}_1}(\vec{x}, \vec{y}) & \text{if } I_1(\vec{x}) \\ \phi_{\vec{y}_2}(\vec{x}, \vec{y}) & \text{else if } I_2(\vec{x}) \\ \cdots & \cdots \\ \phi_{\vec{y}_n}(\vec{x}, \vec{y}) & \text{else } I_n(\vec{x}) \end{cases} \tag{27}$$

The following theorem states that for the chosen model $\vec{x}$, $Sk_{\vec{y}}(\vec{x}, \vec{y})$ satisfies Def. 5, and that $Sk_{\vec{y}}(\vec{x}, \vec{y})$ is defined for all models of $\vec{x}$.

Theorem 2 (Soundness and Completeness of Skolem Relation)**.** *If the set $\{I_i(\vec{x})\}$ is a solution to (26), and $Sk_{\vec{y}}(\vec{x}, \vec{y})$ is as in (27) then:* (1) *$Sk_{\vec{y}}(\vec{x}, \vec{y})$ is a Skolem relation for $\vec{y}$ in formula $S(\vec{x}) \implies \exists \vec{y} \,.\, T(\vec{x}, \vec{y})$,* (2) *$S(\vec{x}) \implies \bigvee_i^n I_i(\vec{x})$.*

Proof. (1) follows immediately from Lemma 5 the last n constraints of (26). (2) follows immediately from the first n constraints of (26).

Example 4. *A partitioning I_1, I_2, I_3 that determines a Skolem relation for Example 3 is: $I_1 \equiv (b \neq 1) \wedge (b \neq 2)$, $I_2 \equiv b \geq 2$, and $I_3 \equiv b = 1$.*

Constructing a Minimal Skolem relation. Any solution to (26) creates a Skolem relation. But not all Skolem relations are equal. In practice, we often like a Skolem relation that minimizes the number of variables on which each partition depends. For example, in Example 4, we have chosen a partition that only depends on the variable b alone. A simple way to find a minimal solution is to iteratively restrict the number of variables in each partition in (26) until no smaller solution can be found. We leave the problem of finding the minimum partitioning for future work.

6 Simulation-Abstraction-Refinement Loop

This section generalizes the approach of the symbolic simulation discovery to programs with non-trivial Control Flow Graphs. Assuming that programs S and T are loop-free, αT is an abstraction of T and ρ_α encodes a relation between S and αT (as in Sect. 4), the check $S \preceq_{\rho_\alpha} \alpha T$ is done by constructing two $\forall\exists$-formulas of the form (7) and (8), and by applying the AE-VAL algorithm to decide validity (as in Sect. 5.1). Furthermore, ρ_α and αT might be refined by means of a Skolem relation (which extraction is shown in Sect. 5.3).

In contrast to the programs with simplified structure considered in Sect. 4, realistic programs involve communication of two or more components with independent transition relations. Simulation relation

Algorithm 2: SIMABS(S, T)

Input: programs S and T, quality metric $\mathcal{Q} : \alpha \to \{\top, \bot\}$
Output: an abstraction $\alpha^{ext} T$,
and a simulation relation ρ_α^{ext}, such that $S \preceq_{\rho_\alpha^{ext}} \alpha^{ext} T$

```
1  α^ext T ← T;
2  forever do
3      α_pre T ← α^ext T;
4      αT, ρ_α ← FINDABS(S, T);
5      if (αT ≠ 𝕌) then
6          return 𝕌, ∅
7      α^ext T, ρ_α^ext ← EXTEND(S, αT, ρ_α);
8      if (𝒬(α^ext T) ∨ (α_pre T = α^ext T)) then
9          return α^ext T, ρ_α^ext
```

Algorithm 3: FINDABS(S, T)

Input: programs S and T
Output: an abstraction αT, a simulation relation ρ_α, such that $S \preceq_{\rho_\alpha} \alpha T$

```
1  for each (u, v) ∈ E do
2      ρ(v) ← SYNTHESIZE(τ_S(u, v), τ_T(u, v));
3      if (τ_S(u, v) ⋠_ρ τ_T(u, v)) then
4          αT ← WEAKEN(T, Var(u) ∪ Var(v));
5          if (αT ≠ 𝕌) then
6              return FINDABS(S, αT);
7          else
8              return 𝕌, ∅;
9  return T, ρ;
```

Algorithm 4: EXTEND($S, \alpha T, \rho_\alpha$)

Input: program S, abstraction αT, simulation relation ρ_α
Output: abstraction $\alpha^{ext} T$, simulation relation ρ_α^{ext}

```
1  ρ_α^ext ← ρ_α;
2  α^ext T ← αT;
3  WL ← E;
4  while (WL ≠ ∅) do
5      (u, v) ← GETEDGE(WL);
6      WL ← WL \ {(u, v)};
7      if (τ_S(u, v) ⪯_{ρ_α^ext} τ_{α^ext(T)}(u, v)) then
8          ρ_α^ext(u) ← ρ_α^ext(u) ∧ SKOLEM(u, v, ρ_α^ext);
9          α^ext T ← STRENGTHEN(αT, Sk);
10         WL ← WL ∪ {(u, x) ∈ E | x ∈ CP} ∪ {(y, u) ∈ E | y ∈ CP};
11     else return αT, ρ_α;
12 return α^ext T, ρ_α^ext;
```

Algorithm 5: SYNTHESIZE(S, T)

Input: loop-free programs S, T
Output: a candidate relation ρ

```
1  return ⋀_{a'_S ∈ V'_S, a'_T ∈ V'_T} (a'_S = a'_T);
```

Algorithm 6: WEAKEN(T, U)

Input: program T, $U \subseteq Var_T$
Output: abstraction αT

```
1  guess U' ⊆ U;
2  return α^∃_{U'}(T);
```

Algorithm 7: STRENGTHEN($\alpha T, Sk$)

Input: abstraction αT, relation Sk
Output: abstraction $\alpha^{ext} T$

```
1  U^ext ← Var(Sk);
2  return α^∃_{U^ext}(T);
```

should be discovered independently for each pair of the matched components and then inductively checked for the compatibility with the pairs of the remaining components. We propose an algorithm SIMABS that implements a complete *Simulation-Abstraction-Refinement Loop* and enables such inductive reasoning.

We use the Cut Point Graph (CPG) representation of a program and treat each program as a graph $\langle \mathrm{CP}, E \rangle$. Here CP is the set of locations which represent heads of the loops (called cutpoints), and $E \subseteq \mathrm{CP} \times \mathrm{CP}$ is the set of longest loop-free program fragments. For example, CPGs of programs in Fig. 2(a)-2(b) are shown in Fig. 4(a)-4(b) respectively. We assume that the considered programs S and T share the graph $\langle \mathrm{CP}, E \rangle$, but might have different labeling of the edges by first-order formulas $\tau_S, \tau_T : E \to Expr$ encoding the transition relations of S and T respectively: $\tau_S \neq \tau_T$.

SIMABS gets as input two programs S and T. The output is an abstraction αT of T and a simulation relation ρ_α^{ext}, such that $S \preceq_{\rho_\alpha^{ext}} T$, if such abstraction exists. In the presentation, we assume that S and T share the set of cutpoints CP, but might have different interpretation of control-flow edges by loop-free program fragments. However, our algorithm extends to the general case where S and T have different cutpoints as well.

SIMABS uses FINDABS (which in turn uses SYNTHESIZE) to guess an initial relation ρ. The initial guess can be an arbitrary total relation between the CPGs of S and T. In our implementation, for every cutpoint, we take ρ to be the identity relation between the live variables[2] of S and T at that cutpoint, that have identical names. Then, FINDABS checks whether there exists an abstraction ρ_α of ρ (including ρ itself) that is a simulation relation between S and a corresponding abstraction αT of T. If this succeeds, the method EXTEND is used to refine the abstraction αT to $\alpha^{ext} T$ and the simulation relation ρ_α to ρ_α^{ext}. This process continues until the abstraction is satisfied by some quality metric $\mathcal{Q}$ (line 8) (for example, if it is sufficient to (dis-)prove some safety property of T) or it is equivalent to an abstraction produced in the previous iteration (line 3). Otherwise, SIMABS goes into the next iteration and finds another abstraction. If the only *universal abstraction* $\mathbb{U}$ can be constructed, SIMABS terminates with a negative result (line 5).

FINDABS (shown in Alg. 3) iteratively checks for each edge $(u, v) \in E$ (line 3), whether a synthesized ρ is a simulation relation between the loop-free program fragment $\tau_S(u, v)$ labeling the (u, v)-edge in S and the corresponding loop-free program fragment $\tau_T(u, v)$ labeling the (u, v)-edge in T. If the check succeeds for all edges, ρ is returned to the user. Otherwise, FINDABS chooses an abstraction αT of T using the method

[2] Informally, a variable is *live* if its current value is used in the program computations.

Figure 4: CPGs of programs in Fig. 2(a)-2(b).

WEAKEN, and repeats the check for S and αT. WEAKEN introduces non-determinism to the interpretation of control-flow edges of T. In our implementation, WEAKEN existentially abstracts of a subset of input variables (and its input variants) of the edge (u, v) for which the simulation check has failed. For each iteration of SIMABS, FINDABS is given a concrete program T, and always constructs a new abstraction from scratch.

EXTEND (shown in Alg. 4) gets as input S, αT, and ρ_α and constructs a refinement ρ_α^{ext} of simulation relation ρ_α, and the corresponding strengthening $\alpha^{ext} T$ of αT. EXTEND maintains a work-list *WL* of CPG edges to be processed. Initially, *WL* is populated with all the CPG edges E (line 3). In each iteration, EXTEND refines simulation relation by adding a Skolem relation *Sk* to ρ_α of the destination node of the current CPG edge (line 8). *Sk* is produced for the existentially abstracted input variables of αT on the current CPG edge and is also used to strengthen αT (line 9). Finally, EXTEND updates *WL* with the outgoing edges from u and other incoming edges to u (line 10) and iterates until *WL* is empty (line 12). If in some iteration, a strengthening is impossible, EXTEND returns the last successful values for ρ_α^{ext} and $\alpha^{ext} T$.

It is worth reminding that for every iteration of FINDABS, as well as for every iteration of EXTEND, there is a need to decide validity of simulation-abstraction-checking formulas (7) and (8). For this goal, we invoke AE-VAL (Alg. 1). Algorithm SKOLEM, used as a subroutine of EXTEND, is an essential extension AE-VAL that produces a Skolem relation as in (27). Notably, it does not extract Skolem relation for local variables, but does it only for existentially abstracted output variables of the current CPG egde.

Recall the source and the target programs from Fig. 2(a)-2(b). In Example 2, we found simulation relation between their loop bodies. In the following, we show how SIMABS is used to discover a simulation relation between whole-programs.

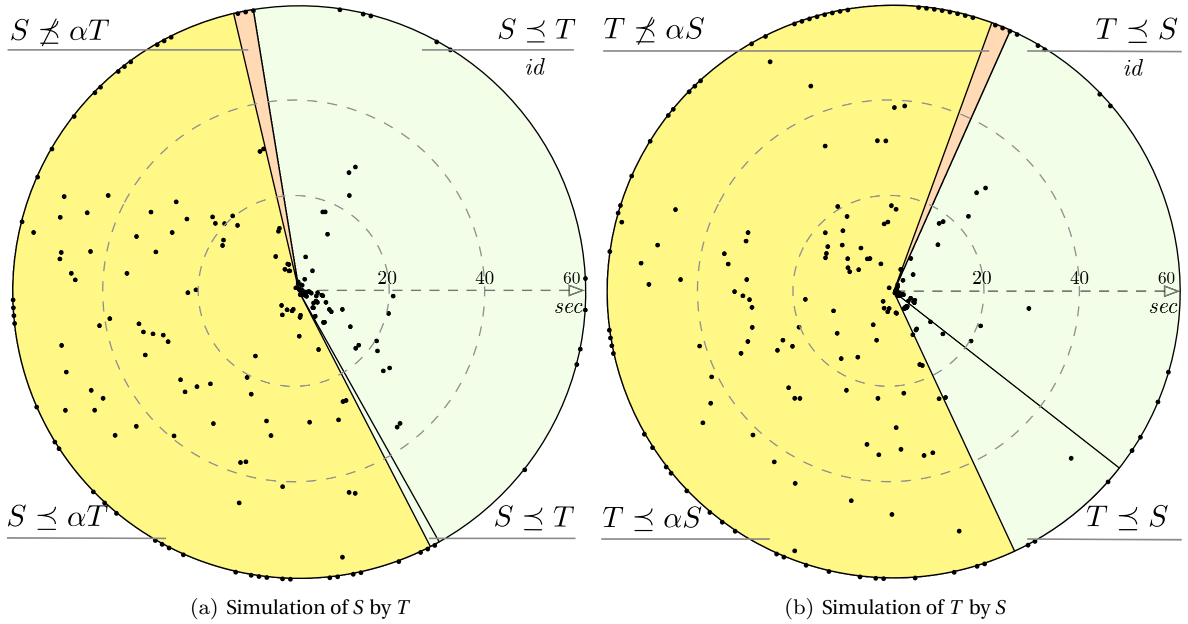
Example 5. *The programs from the example share the set of cutpoints* CP $= \{en, CP_0\}$, *and the set of CPG edges* $E = \{(en, CP_0), (CP_0, CP_0)\}$. *First,* FINDABS *considers the CPG edge* (en, CP_0), *and synthesises an identity relation* $\rho \equiv (a_S = a_T) \wedge (b_S = b_T)$ *which is then proven to be a simulation relation for the current edge. Then* FINDABS *considers the CPG edge* (CP_0, CP_0) *and checks whether* ρ *(and its variant* ρ'*) is an inductive simulation relation for the current edge. Since this check does not succeed (recall Example 1),* WEAKEN *produces an implicit abstraction of the target, by eliminating a (preferably, minimal) subset of variables (e.g.,* $\{b\} \subset \{a, b\}$*), and* FINDABS *recursively calls itself.*

In the second iteration of FINDABS, E *is traversed again, and relation* $\rho_\alpha \equiv (a_S = a_T)$ *is checked w.r.t. the source and the abstraction of the target. Since the check succeeds for all two edges,* EXTEND *extracts Skolem relation and uses it to create* $\rho_\alpha^{ext} \equiv (a_S = a_T) \wedge (b_S = -b_T)$, *and* STRENGTHEN *the abstraction of the target to become the target itself.* SIMABS *successfully terminates, and discovery of simulation relation is completed for the whole-programs.*

7 Evaluating SimAbs and AE-VAL

We implemented SIMABS and AE-VAL in the UFO framework. As an SMT and Horn solver AE-VAL uses Z3. We evaluated them on the Software Verification Competition (SVCOMP) benchmarks and `constprop`, `globalopt`, `instcombine`, `simplifycfg`, `adce`, and `mem2reg` optimizations of LLVM.[3] The `constprop` performs constant propagation, the `globalopt` transforms global variables, the `instcombine` simplifies local arithmetic operations, the `simplifycfg` performs dead code elimination and basic block merging, the `adce` performs aggressive dead code elimination, and `mem2reg` promotes memory references to be register references. Notably, combinations of the optimizations provide more aggressive optimizations than each individual optimization, thus increasing a *semantic gap* between the original and the optimized programs. In our evaluation, we aim at synthesizing simulation relations for programs with a bigger semantic gap and empirically demonstrate the power of AE-VAL (that is expected to have a higher number of AE-VAL iterations in such cases).

[3]LLVM optimizations are documented at `http://llvm.org/docs/Passes.html`.



(a) Simulation of S by T

(b) Simulation of T by S

Figure 5: Pie chart and running times in the spherical coordinate system.

For each source program (S) (300 - 5000 lines of source code), we created an optimized program (T) by combination of all optimizations and applied SIMABS to discover two simulations: $S \preceq T$ and $T \preceq S$. We chose 228 benchmarks for which SIMABS terminates within 10 minutes. We present the results in two diagrams in Fig. 5. Each diagram is a pie chart and a collection of SIMABS execution times for each benchmark in the spherical coordinate system. The pie chart in Fig. 5(a) represents a proportion of four main classes of SIMABS results:

- ◷ : T simulates S via identity relation;
- ◶ : T simulates S via some Skolem-relation-based ρ;
- ◵ : some abstraction αT simulates S;
- ◐ : we did not find an abstraction αT that simulates S.

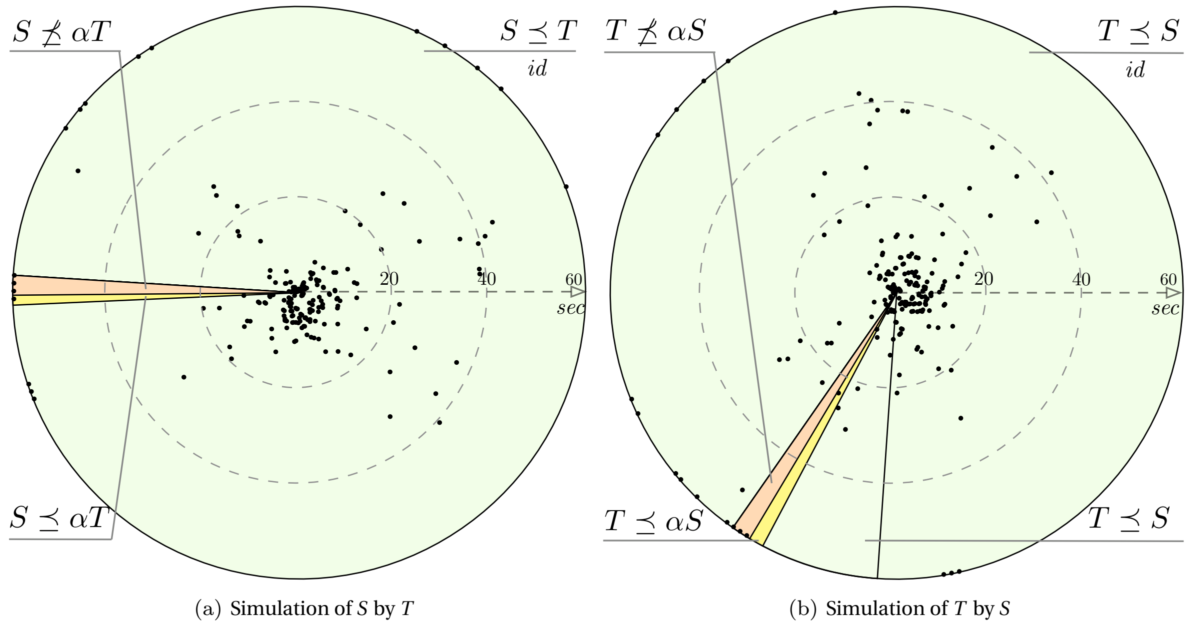
Each dot represents a runtime of SIMABS on a single benchmark. It is placed in one of the circular sectors, ◷, ◶, ◵ or ◐, with respect to the outcome, and is assigned the unique polar angle and the radial distance to represent time in seconds. For example, a benchmark on which $S \preceq_{id} T$ solved in 20 seconds is placed in the sector ◷ in a distance 20 from the center. Being closer to the center means being faster. Runs that took longer than 60 seconds are placed on the boundary. Fig. 5(b) is structured similarly, but with inverse order of S and T.

The experiment shows that SIMABS is able to discover simulation relations between S and T in both directions. While in many cases (101 in Fig. 5(a), and 65 in Fig. 5(b)) it proved simulation by identity, in the remaining cases SIMABS goes deeper into the simulation-abstraction-refinement loop and in 124 and 160 cases respectively delivers either a complete simulation or an abstract simulation. SIMABS is effective in both directions of $S \preceq T$, and terminates with a positive result in all, but 3 pairs of programs (this cases correspond to the disqualified simulations by the universal abstraction).

The core solving engine, AE-VAL, invoked in the low level of SIMABS was shown to be effective while eliminating quantifiers. Overall, it solved 84587 formulas (each formula contains up to 1055 existentially quantified variables, and requires up to 617 iterations to decide validity), and extracted 3503 Skolem relations.

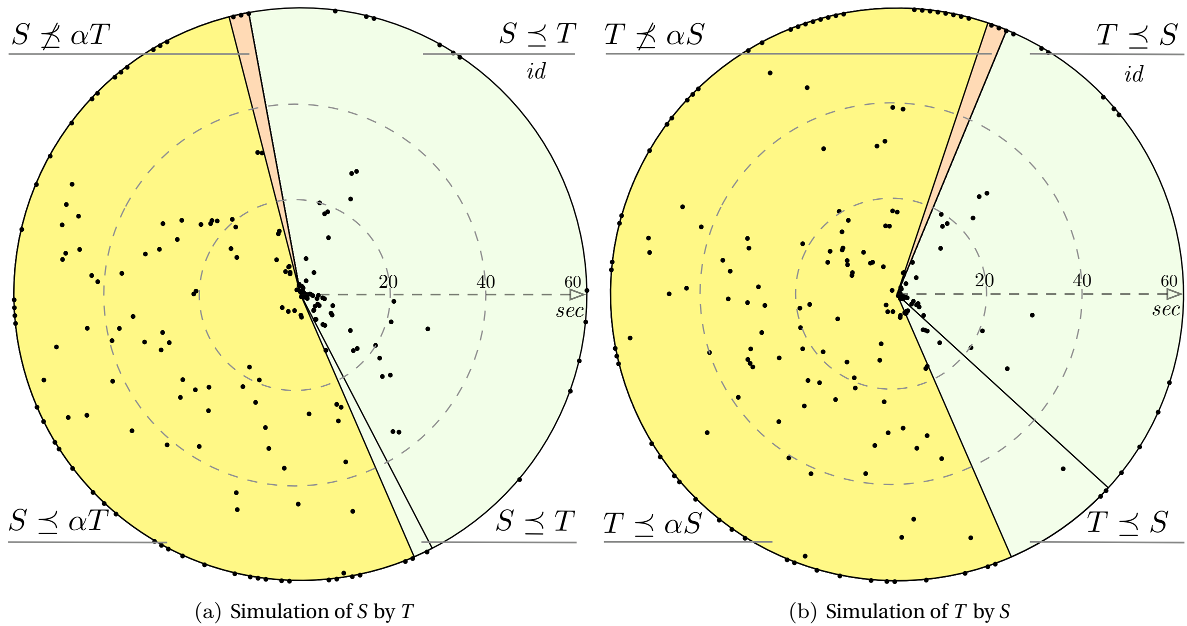
7.1 Evaluating other optimizations

In addition to simulation discovery for the combination of optimizations (described in Sect. 7 and depicted in Fig. 5), we applied SIMABS to discover simulation relations between the source program and the target program resulted from individual (and therefore, less aggressive) optimizations.



(a) Simulation of S by T (b) Simulation of T by S

Figure 6: `constprop, globalopt, adce`



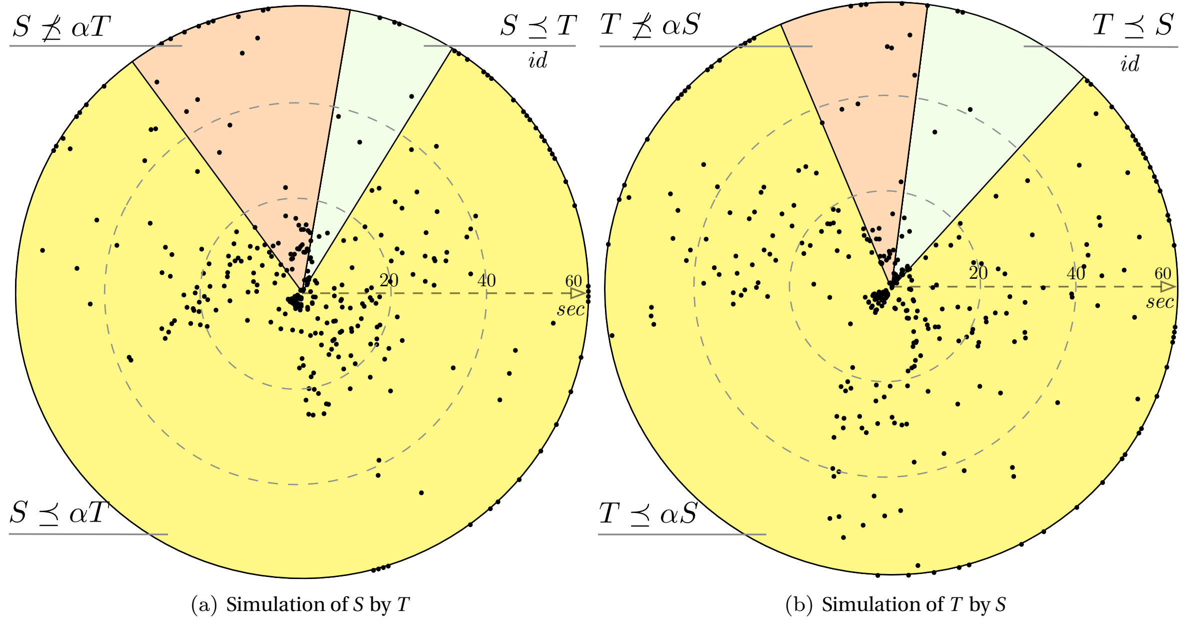
(a) Simulation of S by T (b) Simulation of T by S

Figure 7: `instcombine, simplifycfg`

Fig. 6 depicts the effect of optimizations `constprop, globalopt, adce`. SIMABS was able to discover mutual simulation relation between S and T in 223 cases out of 228 ones. This can be explained by our intuition that the considered optimizations are relatively lightweight.

Fig. 7 depicts the effect of optimizations `instcombine, simplifycfg`. SIMABS performs similarly to the experiment in Fig. 5, but it was able to discover 3 more total simulation relations in the direction $S \preceq T$ and 2 more total simulation relations in the direction $T \preceq S$.

Our experiments for discovering simulation relations individually for `instcombine`, were identical to the case of `instcombine, simplifycfg`. We therefore omit presenting the corresponding figures.



(a) Simulation of S by T

(b) Simulation of T by S

Figure 8: Discovering simulation for program mutants.

7.2 Application to mutation testing

We demonstrate the applicability of SIMABS to a rather different scenario from proving correctness of compiler optimizations. *Mutation testing* is used to evaluate the quality of program specification. The idea is to generate a so called *program mutant* using a small random change. It could change arithmetic operations (e.g., "+" to "-", as in Fig. 2(a)-2(b)). Finally, the goal of testing is to *kill* such mutant (i.e., to prove that it is inconsistent with the given specification), and therefore to demonstrate that the change brakes the specification.

We expand this idea and move it into the verification domain. More specifically, for a given program S, we generate a mutant program T and then proceed with synthesizing a simulation relation between S and T. The results of the experiment are shown in Fig. 8. SIMABS was applied to 352 pairs of S and T, but succeeded in synthesis of total simulation relation (i.e., $S \preceq T$) less than in 10% of cases (21 cases for $S \preceq T$ and 34 cases for $T \preceq S$). On the the other hand, SIMABS was unable to find any abstraction αT, such that $S \preceq \alpha T$ in more than in 8% cases (45 cases for $S \not\preceq \alpha T$ and 31 cases for $T \not\preceq \alpha S$). Both numbers significantly differ from the correspondent ones in the experiments from Sect. 7.1: number of $\star \preceq \star$ results was much larger, and the number of $\star \not\preceq \alpha\star$ was much lower. Intuitively, this means that our mutation was successful, and the correspondent program mutants were killed.

8 Experimental data

In this section, we present the raw data from our experiment, explained in details in Sect. 7. There are 2 tables corresponding to the following cases:

- Table 1 for S and T mutually simulated (via identity relation or some synthesised relation);
- Table 2 for S and T mutually simulated on abstraction level.

Each Table gathers the experimental data corresponding to the SIMABS run for different optimization passes. The more aggressive optimizations are highlighted by the darker color. The columns with the less dark color (the leftmost and the rightmost) correspond to the less aggressive optimizations: `constprop, globalopt, adce` (depicted in Fig. 6). The columns with the middle dark color correspond to `instcombine, simplifycfg` optimizations (depicted in Fig. 7). Finally, the columns with the darkest color (two central ones) correspond to the most aggressive optimizations (combination of all, depicted in Fig. 5). For all 6 experiments, we provide the result of simulation discovery and the total execution time. Full results are available at http://www.inf.usi.ch/phd/fedyukovich/niagara.

Benchmark acronym	-constprop -globalopt -adce		-instcombine -simplifycfg		-constprop -globalopt -adce -instcombine -simplifycfg		-constprop -globalopt -adce -instcombine -simplifycfg		-instcombine -simplifycfg		-constprop -globalopt -adce	
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
module_drbd_u0	$S \preceq_{id} T$	1.04	$S \preceq_{id} T$	1.03	$S \preceq_{id} T$	1.07	$T \preceq_{id} S$	1	$T \preceq_{id} S$	1.06	$T \preceq_{id} S$	1.07
module_drivers_u3	$S \preceq_{id} T$	1.09	$S \preceq_{id} T$	1.05	$S \preceq_{id} T$	0.99	$T \preceq_{id} S$	1.1	$T \preceq_{id} S$	1.04	$T \preceq_{id} S$	1.04
usb_s0	$S \preceq_{id} T$	235.51	$S \preceq_{id} T$	262.28	$S \preceq_{id} T$	258.88	$T \preceq_{id} S$	207.08	$T \preceq_{id} S$	211.8	$T \preceq_{id} S$	236.39
32_vp7045_main0_s3	$S \preceq_{id} T$	19.03	$S \preceq_{id} T$	19.79	$S \preceq_{id} T$	19.16	$T \preceq_{id} S$	19.2	$T \preceq_{id} S$	19.04	$T \preceq_{id} S$	19.1
32_vp7045_main1_s3	$S \preceq_{id} T$	95.43	$S \preceq_{id} T$	95.69	$S \preceq_{id} T$	95.53	$T \preceq_{id} S$	95.13	$T \preceq_{id} S$	96.34	$T \preceq_{id} S$	94.9
32_tps6105x_s0	$S \preceq_{id} T$	3.68	$S \preceq_{id} T$	3.52	$S \preceq_{id} T$	3.59	$T \preceq_{id} S$	3.62	$T \preceq_{id} S$	3.62	$T \preceq_{id} S$	3.57
32_tps6105x_s3	$S \preceq_{id} T$	1.54	$S \preceq_{id} T$	1.35	$S \preceq_{id} T$	1.3	$T \preceq_{id} S$	1.52	$T \preceq_{id} S$	1.54	$T \preceq_{id} S$	1.56
32_isdn_u0	$S \preceq_{id} T$	4.86	$S \preceq_{id} T$	5.08	$S \preceq_{id} T$	4.93	$T \preceq_{id} S$	4.92	$T \preceq_{id} S$	5.3	$T \preceq_{id} S$	4.97
32_isdn_u3	$S \preceq_{id} T$	4.54	$S \preceq_{id} T$	4.67	$S \preceq_{id} T$	4.69	$T \preceq_{id} S$	4.64	$T \preceq_{id} S$	4.61	$T \preceq_{id} S$	4.72
32_vivi_u0	$S \preceq_{id} T$	291.23	$S \preceq_{id} T$	291.21	$S \preceq_{id} T$	291.38	$T \preceq_{id} S$	289.51	$T \preceq_{id} S$	294.87	$T \preceq_{id} S$	293.4
32_scsi_u0	$S \preceq_{id} T$	19.56	$S \preceq_{id} T$	19.59	$S \preceq_{id} T$	19.65	$T \preceq_{id} S$	19.59	$T \preceq_{id} S$	19.45	$T \preceq_{id} S$	19.43
32_keucr_u0	$S \preceq_{id} T$	4.76	$S \preceq_{id} T$	2.58	$S \preceq_{id} T$	2.49	$T \preceq_{id} S$	4.77	$T \preceq_{id} S$	4.8	$T \preceq_{id} S$	4.71
32_usb_u0	$S \preceq_{id} T$	7.1	$S \preceq_{id} T$	3.74	$S \preceq_{id} T$	3.81	$T \preceq_{id} S$	3.82	$T \preceq_{id} S$	3.71	$T \preceq_{id} S$	7.03
diskperf_simpl1_s0	$S \preceq_{id} T$	82.15	$S \preceq_{id} T$	78.69	$S \preceq_{id} T$	78.07	$T \preceq_{id} S$	79.25	$T \preceq_{id} S$	80.96	$T \preceq_{id} S$	80.63
kbfiltr_simpl2_u0	$S \preceq_{id} T$	2.26	$S \preceq_{id} T$	2.1	$S \preceq_{id} T$	2.14	$T \preceq_{id} S$	2.2	$T \preceq_{id} S$	2.27	$T \preceq_{id} S$	2.22
kbfiltr_simpl2_u3	$S \preceq_{id} T$	1.94	$S \preceq_{id} T$	1.97	$S \preceq_{id} T$	1.9	$T \preceq_{id} S$	1.9	$T \preceq_{id} S$	1.91	$T \preceq_{id} S$	1.9
kbfiltr_simpl1_s0	$S \preceq_{id} T$	0.54	$S \preceq_{id} T$	0.42	$S \preceq_{id} T$	0.4	$T \preceq_{id} S$	0.54	$T \preceq_{id} S$	0.51	$T \preceq_{id} S$	0.46
kbfiltr_simpl2_s0	$S \preceq_{id} T$	0.34	$S \preceq_{id} T$	0.31	$S \preceq_{id} T$	0.38	$T \preceq_{id} S$	0.34	$T \preceq_{id} S$	0.34	$T \preceq_{id} S$	0.35
diskperf_s3	$S \preceq_{id} T$	1.46	$S \preceq_{id} T$	1.22	$S \preceq_{id} T$	1.18	$T \preceq_{id} S$	1.43	$T \preceq_{id} S$	1.45	$T \preceq_{id} S$	1.42
kundu_s3	$S \preceq_{id} T$	290.52	$S \preceq_{id} T$	260.55	$S \preceq_{id} T$	256.66	$T \preceq_{id} S$	290.03	$T \preceq_{id} S$	297.13	$T \preceq_{id} S$	287.81
kundu1_u3	$S \preceq_{id} T$	5.34	$S \preceq_{id} T$	5.51	$S \preceq_{id} T$	5.42	$T \preceq_{id} S$	5.62	$T \preceq_{id} S$	5.53	$T \preceq_{id} S$	5.28
kundu2_u3	$S \preceq_{id} T$	225.29	$S \preceq_{id} T$	219.85	$S \preceq_{id} T$	220.45	$T \preceq_{id} S$	221.87	$T \preceq_{id} S$	225.33	$T \preceq_{id} S$	221.92
pc_sfifo_1_s3	$S \preceq_{id} T$	22.43	$S \preceq_{id} T$	22.72	$S \preceq_{id} T$	22.98	$T \preceq_{id} S$	22.44	$T \preceq_{id} S$	22.78	$T \preceq_{id} S$	22.75
pc_sfifo_1_u3	$S \preceq_{id} T$	13.64	$S \preceq_{id} T$	13.39	$S \preceq_{id} T$	13.68	$T \preceq_{id} S$	13.72	$T \preceq_{id} S$	13.49	$T \preceq_{id} S$	13.8
pc_sfifo_2_s3	$S \preceq_{id} T$	28.71	$S \preceq_{id} T$	28.47	$S \preceq_{id} T$	28.92	$T \preceq_{id} S$	29.23	$T \preceq_{id} S$	28.57	$T \preceq_{id} S$	28.47
pc_sfifo_2_u3	$S \preceq_{id} T$	26.91	$S \preceq_{id} T$	27.66	$S \preceq_{id} T$	27.31	$T \preceq_{id} S$	27.26	$T \preceq_{id} S$	26.97	$T \preceq_{id} S$	26.87
token_ring.01_s3	$S \preceq_{id} T$	3.02	$S \preceq_{id} T$	3.23	$S \preceq_{id} T$	3.12	$T \preceq_{id} S$	3.28	$T \preceq_{id} S$	3.14	$T \preceq_{id} S$	3.1
token_ring.01_u3	$S \preceq_{id} T$	1.62	$S \preceq_{id} T$	1.6	$S \preceq_{id} T$	1.74	$T \preceq_{id} S$	1.54	$T \preceq_{id} S$	1.66	$T \preceq_{id} S$	1.69
token_ring.02_s3	$S \preceq_{id} T$	18.34	$S \preceq_{id} T$	18.37	$S \preceq_{id} T$	17.86	$T \preceq_{id} S$	18.18	$T \preceq_{id} S$	18.41	$T \preceq_{id} S$	18.16
token_ring.02_u3	$S \preceq_{id} T$	17.48	$S \preceq_{id} T$	17.78	$S \preceq_{id} T$	17.7	$T \preceq_{id} S$	17.4	$T \preceq_{id} S$	17.37	$T \preceq_{id} S$	17.47
token_ring.03_s3	$S \preceq_{id} T$	169.52	$S \preceq_{id} T$	171.06	$S \preceq_{id} T$	170.95	$T \preceq_{id} S$	171.27	$T \preceq_{id} S$	170.48	$T \preceq_{id} S$	170.67
token_ring.03_u3	$S \preceq_{id} T$	174	$S \preceq_{id} T$	175.24	$S \preceq_{id} T$	176.84	$T \preceq_{id} S$	176.25	$T \preceq_{id} S$	175.79	$T \preceq_{id} S$	176.13
transmitter.01_u3	$S \preceq_{id} T$	1.87	$S \preceq_{id} T$	1.84	$S \preceq_{id} T$	1.77	$T \preceq_{id} S$	1.88	$T \preceq_{id} S$	1.86	$T \preceq_{id} S$	1.84
transmitter.02_u3	$S \preceq_{id} T$	7.84	$S \preceq_{id} T$	7.75	$S \preceq_{id} T$	7.7	$T \preceq_{id} S$	8.05	$T \preceq_{id} S$	7.68	$T \preceq_{id} S$	7.49
transmitter.03_u3	$S \preceq_{id} T$	78.9	$S \preceq_{id} T$	78.1	$S \preceq_{id} T$	77.78	$T \preceq_{id} S$	77.2	$T \preceq_{id} S$	77.64	$T \preceq_{id} S$	78.67
mps3_01_u3	$S \preceq_{id} T$	0.58	$S \preceq_{id} T$	0.58	$S \preceq_{id} T$	0.65	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.58	$T \preceq_{id} S$	0.6
mps3_02_u0	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.64	$S \preceq_{id} T$	0.65	$T \preceq_{id} S$	0.59	$T \preceq_{id} S$	0.54	$T \preceq_{id} S$	0.62
mps3_04_u0	$S \preceq_{id} T$	0.76	$S \preceq_{id} T$	0.8	$S \preceq_{id} T$	0.79	$T \preceq_{id} S$	0.8	$T \preceq_{id} S$	0.81	$T \preceq_{id} S$	0.77
mps3_05_u0	$S \preceq_{id} T$	0.64	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.59	$T \preceq_{id} S$	0.63	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.61
mps3_05_u3	$S \preceq_{id} T$	0.63	$S \preceq_{id} T$	0.56	$S \preceq_{id} T$	0.58	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.61	$T \preceq_{id} S$	0.56
mps3_08_u0	$S \preceq_{id} T$	0.8	$S \preceq_{id} T$	0.75	$S \preceq_{id} T$	0.81	$T \preceq_{id} S$	0.72	$T \preceq_{id} S$	0.77	$T \preceq_{id} S$	0.86
mps3_08_u3	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.79	$T \preceq_{id} S$	0.77	$T \preceq_{id} S$	0.77	$T \preceq_{id} S$	0.75
mps3_10_u0	$S \preceq_{id} T$	0.59	$S \preceq_{id} T$	0.58	$S \preceq_{id} T$	0.61	$T \preceq_{id} S$	0.64	$T \preceq_{id} S$	0.61	$T \preceq_{id} S$	0.62
mps3_10_u3	$S \preceq_{id} T$	0.61	$S \preceq_{id} T$	0.59	$S \preceq_{id} T$	0.61	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.56
mps3_11_u3	$S \preceq_{id} T$	0.75	$S \preceq_{id} T$	0.75	$S \preceq_{id} T$	0.7	$T \preceq_{id} S$	0.87	$T \preceq_{id} S$	0.78	$T \preceq_{id} S$	0.76
mps3_12_u0	$S \preceq_{id} T$	0.8	$S \preceq_{id} T$	0.73	$S \preceq_{id} T$	0.76	$T \preceq_{id} S$	0.81	$T \preceq_{id} S$	0.8	$T \preceq_{id} S$	0.8
mps3_12_u3	$S \preceq_{id} T$	0.78	$S \preceq_{id} T$	0.83	$S \preceq_{id} T$	0.8	$T \preceq_{id} S$	0.92	$T \preceq_{id} S$	0.83	$T \preceq_{id} S$	0.81
mps3_14_u0	$S \preceq_{id} T$	0.65	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.62	$T \preceq_{id} S$	0.67	$T \preceq_{id} S$	0.62	$T \preceq_{id} S$	0.59
mps3_15_u3	$S \preceq_{id} T$	0.72	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.76	$T \preceq_{id} S$	0.87	$T \preceq_{id} S$	0.73	$T \preceq_{id} S$	0.84
mps3_16_u3	$S \preceq_{id} T$	0.73	$S \preceq_{id} T$	0.84	$S \preceq_{id} T$	0.74	$T \preceq_{id} S$	0.78	$T \preceq_{id} S$	0.8	$T \preceq_{id} S$	0.83
mps3_17_u0	$S \preceq_{id} T$	0.61	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.62	$T \preceq_{id} S$	0.64	$T \preceq_{id} S$	0.62	$T \preceq_{id} S$	0.56
mps3_17_u3	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.59	$S \preceq_{id} T$	0.61	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.62	$T \preceq_{id} S$	0.64
mps3_18_u3	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.58	$S \preceq_{id} T$	0.65	$T \preceq_{id} S$	0.62	$T \preceq_{id} S$	0.63	$T \preceq_{id} S$	0.6
mps3_21_u0	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.58	$S \preceq_{id} T$	0.61	$T \preceq_{id} S$	0.65	$T \preceq_{id} S$	0.58	$T \preceq_{id} S$	0.66
mps3_22_u0	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.66	$S \preceq_{id} T$	0.6	$T \preceq_{id} S$	0.61	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.59
mps3_22_u3	$S \preceq_{id} T$	0.59	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.59	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.61	$T \preceq_{id} S$	0.6
mps3_24_u0	$S \preceq_{id} T$	0.85	$S \preceq_{id} T$	0.78	$S \preceq_{id} T$	0.78	$T \preceq_{id} S$	0.79	$T \preceq_{id} S$	0.72	$T \preceq_{id} S$	0.74
mps3_24_u3	$S \preceq_{id} T$	0.75	$S \preceq_{id} T$	0.84	$S \preceq_{id} T$	0.77	$T \preceq_{id} S$	0.85	$T \preceq_{id} S$	0.81	$T \preceq_{id} S$	0.86
mps3_26_u3	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.65	$S \preceq_{id} T$	0.62	$T \preceq_{id} S$	0.63	$T \preceq_{id} S$	0.6	$T \preceq_{id} S$	0.66
mps3_27_u3	$S \preceq_{id} T$	0.81	$S \preceq_{id} T$	0.83	$S \preceq_{id} T$	0.8	$T \preceq_{id} S$	0.75	$T \preceq_{id} S$	0.78	$T \preceq_{id} S$	0.76
mps3_28_u0	$S \preceq_{id} T$	0.77	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.8	$T \preceq_{id} S$	0.73	$T \preceq_{id} S$	0.85	$T \preceq_{id} S$	0.8
mps3_28_u3	$S \preceq_{id} T$	0.84	$S \preceq_{id} T$	0.76	$S \preceq_{id} T$	0.74	$T \preceq_{id} S$	0.74	$T \preceq_{id} S$	0.73	$T \preceq_{id} S$	0.79
mps3_29_u0	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.62	$S \preceq_{id} T$	0.64	$T \preceq_{id} S$	0.59	$T \preceq_{id} S$	0.63	$T \preceq_{id} S$	0.65
mps3_30_u3	$S \preceq_{id} T$	0.63	$S \preceq_{id} T$	0.6	$S \preceq_{id} T$	0.56	$T \preceq_{id} S$	0.61	$T \preceq_{id} S$	0.58	$T \preceq_{id} S$	0.6
mps3_03_u0	$S \preceq_{id} T$	0.93	$S \preceq_{id} T$	0.88	$S \preceq_{id} T$	0.76	$T \preceq_{\rho_\alpha \wedge sk} S$	4.01	$T \preceq_{\rho_\alpha \wedge sk} S$	4.25	$T \preceq_{\rho_\alpha \wedge sk} S$	3.92
mps3_07_u0	$S \preceq_{id} T$	0.94	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.77	$T \preceq_{\rho_\alpha \wedge sk} S$	4.06	$T \preceq_{\rho_\alpha \wedge sk} S$	4.04	$T \preceq_{\rho_\alpha \wedge sk} S$	4.23
mps3_11_u0	$S \preceq_{id} T$	0.96	$S \preceq_{id} T$	0.79	$S \preceq_{id} T$	0.74	$T \preceq_{\rho_\alpha \wedge sk} S$	4.15	$T \preceq_{\rho_\alpha \wedge sk} S$	3.98	$T \preceq_{\rho_\alpha \wedge sk} S$	4.05
mps3_15_u0	$S \preceq_{id} T$	0.97	$S \preceq_{id} T$	0.78	$S \preceq_{id} T$	0.76	$T \preceq_{\rho_\alpha \wedge sk} S$	3.95	$T \preceq_{\rho_\alpha \wedge sk} S$	4	$T \preceq_{\rho_\alpha \wedge sk} S$	4.16
mps3_19_u0	$S \preceq_{id} T$	0.91	$S \preceq_{id} T$	0.74	$S \preceq_{id} T$	0.76	$T \preceq_{\rho_\alpha \wedge sk} S$	4.03	$T \preceq_{\rho_\alpha \wedge sk} S$	3.98	$T \preceq_{\rho_\alpha \wedge sk} S$	4
mps3_23_u0	$S \preceq_{id} T$	0.84	$S \preceq_{id} T$	0.76	$S \preceq_{id} T$	0.76	$T \preceq_{\rho_\alpha \wedge sk} S$	4.22	$T \preceq_{\rho_\alpha \wedge sk} S$	3.97	$T \preceq_{\rho_\alpha \wedge sk} S$	3.95
mps3_27_u0	$S \preceq_{id} T$	1	$S \preceq_{id} T$	0.74	$S \preceq_{id} T$	0.82	$T \preceq_{\rho_\alpha \wedge sk} S$	3.98	$T \preceq_{\rho_\alpha \wedge sk} S$	4.23	$T \preceq_{\rho_\alpha \wedge sk} S$	4.12
mps3_31_u0	$S \preceq_{id} T$	0.91	$S \preceq_{id} T$	0.76	$S \preceq_{id} T$	0.78	$T \preceq_{\rho_\alpha \wedge sk} S$	4.14	$T \preceq_{\rho_\alpha \wedge sk} S$	3.9	$T \preceq_{\rho_\alpha \wedge sk} S$	4.2
pc_sfifo_1_s0	$S \preceq_{id} T$	2.37	$S \preceq_{id} T$	2.3	$S \preceq_{id} T$	2.37	$T \preceq_{\rho_\alpha \wedge sk} S$	13.53	$T \preceq_{\rho_\alpha \wedge sk} S$	13.4	$T \preceq_{\rho_\alpha \wedge sk} S$	13.32
pc_sfifo_1_u0	$S \preceq_{id} T$	1.88	$S \preceq_{id} T$	1.95	$S \preceq_{id} T$	1.92	$T \preceq_{\rho_\alpha \wedge sk} S$	9.92	$T \preceq_{\rho_\alpha \wedge sk} S$	9.85	$T \preceq_{\rho_\alpha \wedge sk} S$	9.95

Table 1: Mutual simulation for 3 types of optimizations.

9 Conclusion and Future Work

We proposed a solution to the problem of automated discovery of simulations between programs. Our main contribution is SIMABS, a novel algorithm to automatically synthesize both, abstractions and simulations, between the source and the target programs. If the target does not simulate the source, SIMABS iteratively performs abstraction-refinement reasoning to detect an abstraction of the target that simulates the source. In contrast to existing techniques, SIMABS operates by deciding validity of ∀∃-formulas iteratively. The second contribution of the paper is AE-VAL, a novel decision procedure that extracts Skolem

Benchmark acronym	-constprop -globalopt -adce		-instcombine -simplifycfg		-constprop -globalopt -adce -instcombine -simplifycfg		-constprop -globalopt -adce -instcombine -simplifycfg		-instcombine -simplifycfg		-constprop -globalopt -adce	
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
module_mISDN_cil0	$S \preceq_{id} T$	6.44	$S \preceq \alpha T$	30.87	$S \preceq \alpha T$	31.14	$T \preceq \alpha S$	99.1	$T \preceq \alpha S$	97.97	$T \preceq_{id} S$	6.32
module_mISDN_0	$S \preceq_{id} T$	6.37	$S \preceq \alpha T$	31.23	$S \preceq \alpha T$	30.83	$T \preceq \alpha S$	98.75	$T \preceq \alpha S$	98.23	$T \preceq_{id} S$	6.36
usb_usbmouse0	$S \preceq_{id} T$	20.54	$S \preceq \alpha T$	14.15	$S \preceq \alpha T$	14.22	$T \preceq \alpha S$	113.44	$T \preceq \alpha S$	111.81	$T \preceq_{id} S$	21.17
usb_usbhid0	$S \preceq_{id} T$	17.23	$S \preceq \alpha T$	13.68	$S \preceq \alpha T$	13.69	$T \preceq \alpha S$	115.1	$T \preceq \alpha S$	113.97	$T \preceq_{id} S$	17.39
usb_keyspan_cil0	$S \preceq_{id} T$	8.99	$S \preceq \alpha T$	6.48	$S \preceq \alpha T$	6.4	$T \preceq \alpha S$	39.19	$T \preceq \alpha S$	39.07	$T \preceq_{id} S$	8.92
usb_keyspan_0	$S \preceq_{id} T$	8.81	$S \preceq \alpha T$	6.41	$S \preceq \alpha T$	6.58	$T \preceq \alpha S$	38.79	$T \preceq \alpha S$	39.41	$T \preceq_{id} S$	8.72
usb_tablet_cil0	$S \preceq_{id} T$	7.7	$S \preceq \alpha T$	5.26	$S \preceq \alpha T$	5.08	$T \preceq \alpha S$	31.84	$T \preceq \alpha S$	32.41	$T \preceq_{id} S$	7.73
usb_tablet_0	$S \preceq_{id} T$	7.55	$S \preceq \alpha T$	4.98	$S \preceq \alpha T$	5.13	$T \preceq \alpha S$	32.01	$T \preceq \alpha S$	31.57	$T \preceq_{id} S$	7.87
usb_ttusb_cil3	$S \preceq_{id} T$	289.11	$S \preceq \alpha T$	516.36	$S \preceq \alpha T$	505.87	$T \preceq \alpha S$	315.9	$T \preceq \alpha S$	323.52	$T \preceq_{id} S$	291.79
usb_ttusb_3	$S \preceq_{id} T$	291.58	$S \preceq \alpha T$	513.93	$S \preceq \alpha T$	510.35	$T \preceq \alpha S$	318.29	$T \preceq \alpha S$	317.8	$T \preceq_{id} S$	290.5
usb_i14800	$S \preceq_{id} T$	128.21	$S \preceq \alpha T$	151.89	$S \preceq \alpha T$	152.07	$T \preceq \alpha S$	158.85	$T \preceq \alpha S$	159.11	$T \preceq_{id} S$	128.19
32_vp7045_main0_s0	$S \preceq_{id} T$	7.12	$S \preceq_{id} T$	7.1	$S \preceq \alpha T$	12.48	$T \preceq_{\rho_\alpha \wedge sk} S$	24.57	$T \preceq_{id} S$	7.04	$T \preceq_{id} S$	7.06
32_vp7045_main1_s0	$S \preceq_{id} T$	27.06	$S \preceq_{id} T$	27.58	$S \preceq \alpha T$	67.05	$T \preceq_{\rho_\alpha \wedge sk} S$	104.16	$T \preceq_{id} S$	27.8	$T \preceq_{id} S$	26.83
kbfiltr_u3	$S \preceq_{id} T$	28.72	$S \preceq \alpha T$	27.53	$S \preceq \alpha T$	27.8	$T \preceq_{id} S$	28.6	$T \preceq_{id} S$	28.71	$T \preceq_{id} S$	29.17
mps1_35_u0	$S \preceq_{id} T$	12.55	$S \preceq \alpha T$	136.76	$S \preceq \alpha T$	134.1	$T \preceq \alpha S$	106.9	$T \preceq \alpha S$	104.75	$T \preceq_{id} S$	12.69
mps1_35_u3	$S \preceq_{id} T$	12.05	$S \preceq \alpha T$	121.56	$S \preceq \alpha T$	109.38	$T \preceq \alpha S$	74.78	$T \preceq \alpha S$	83.87	$T \preceq_{id} S$	12.36
mps1_36_u3	$S \preceq_{id} T$	13.68	$S \preceq \alpha T$	149.49	$S \preceq \alpha T$	153.29	$T \preceq \alpha S$	150.43	$T \preceq \alpha S$	142.55	$T \preceq_{id} S$	13.58
mps1_40_u0	$S \preceq_{id} T$	30.11	$S \preceq \alpha T$	305.11	$S \preceq \alpha T$	372.42	$T \preceq \alpha S$	179.68	$T \preceq \alpha S$	152.76	$T \preceq_{id} S$	29.61
mps1_40_u3	$S \preceq_{id} T$	18.63	$S \preceq \alpha T$	210.03	$S \preceq \alpha T$	201.54	$T \preceq \alpha S$	176.14	$T \preceq \alpha S$	148.22	$T \preceq_{id} S$	18.27
mps1_44_u0	$S \preceq_{id} T$	19.96	$S \preceq \alpha T$	190.6	$S \preceq \alpha T$	194.84	$T \preceq \alpha S$	91.4	$T \preceq \alpha S$	90.21	$T \preceq_{id} S$	19.9
mps1_45_s0	$S \preceq_{id} T$	5.04	$S \preceq \alpha T$	25.59	$S \preceq \alpha T$	32.13	$T \preceq \alpha S$	16.23	$T \preceq \alpha S$	22.47	$T \preceq_{id} S$	5.08
mps1_46_s0	$S \preceq_{id} T$	5.03	$S \preceq \alpha T$	29.42	$S \preceq \alpha T$	28.23	$T \preceq \alpha S$	15.96	$T \preceq \alpha S$	16.02	$T \preceq_{id} S$	5.22
mps1_47_s3	$S \preceq_{id} T$	9.66	$S \preceq \alpha T$	94.59	$S \preceq \alpha T$	102.35	$T \preceq \alpha S$	63.32	$T \preceq \alpha S$	48.54	$T \preceq_{id} S$	9.79
mps1_50_u0	$S \preceq_{id} T$	5.89	$S \preceq \alpha T$	53.21	$S \preceq \alpha T$	44.99	$T \preceq \alpha S$	48.83	$T \preceq \alpha S$	43.38	$T \preceq_{id} S$	5.9
mps1_51_u3	$S \preceq_{id} T$	11.24	$S \preceq \alpha T$	117.74	$S \preceq \alpha T$	105.49	$T \preceq \alpha S$	107.44	$T \preceq \alpha S$	118.9	$T \preceq_{id} S$	11.36
mps1_53_u0	$S \preceq_{id} T$	6.32	$S \preceq \alpha T$	52.62	$S \preceq \alpha T$	53.28	$T \preceq \alpha S$	64.14	$T \preceq \alpha S$	68.74	$T \preceq_{id} S$	6.22
mps1_57_s0	$S \preceq_{id} T$	6.64	$S \preceq \alpha T$	49.37	$S \preceq \alpha T$	47.54	$T \preceq \alpha S$	58.76	$T \preceq \alpha S$	53.36	$T \preceq_{id} S$	6.62
mps1_58_s0	$S \preceq_{id} T$	6.58	$S \preceq \alpha T$	52.07	$S \preceq \alpha T$	52.57	$T \preceq \alpha S$	50.36	$T \preceq \alpha S$	52.53	$T \preceq_{id} S$	6.54
mps1_59_s3	$S \preceq_{id} T$	13.4	$S \preceq \alpha T$	133.44	$S \preceq \alpha T$	129.44	$T \preceq \alpha S$	84.63	$T \preceq \alpha S$	69.41	$T \preceq_{id} S$	13.65
mps1_61_s0	$S \preceq_{id} T$	7.88	$S \preceq \alpha T$	47.26	$S \preceq \alpha T$	57.12	$T \preceq \alpha S$	53.19	$T \preceq \alpha S$	61.27	$T \preceq_{id} S$	7.78
mps2_37_s0	$S \preceq_{id} T$	5.18	$S \preceq \alpha T$	49.98	$S \preceq \alpha T$	50.62	$T \preceq \alpha S$	44.64	$T \preceq \alpha S$	44.05	$T \preceq_{id} S$	5.16
mps2_38_s0	$S \preceq_{id} T$	5.02	$S \preceq \alpha T$	50.42	$S \preceq \alpha T$	50.69	$T \preceq \alpha S$	51.22	$T \preceq \alpha S$	47.66	$T \preceq_{id} S$	5.29
mps2_41_u0	$S \preceq_{id} T$	6.45	$S \preceq \alpha T$	43.95	$S \preceq \alpha T$	47.97	$T \preceq \alpha S$	31.51	$T \preceq \alpha S$	31.28	$T \preceq_{id} S$	6.52
mps2_42_u0	$S \preceq_{id} T$	6.37	$S \preceq \alpha T$	42.58	$S \preceq \alpha T$	47.04	$T \preceq \alpha S$	30.15	$T \preceq \alpha S$	27.69	$T \preceq_{id} S$	6.34
mps2_49_s3	$S \preceq_{id} T$	6.28	$S \preceq \alpha T$	62.74	$S \preceq \alpha T$	61.59	$T \preceq \alpha S$	64.71	$T \preceq \alpha S$	56.55	$T \preceq_{id} S$	6.27
mps2_57_s0	$S \preceq_{id} T$	10.32	$S \preceq \alpha T$	80	$S \preceq \alpha T$	85.82	$T \preceq \alpha S$	88.55	$T \preceq \alpha S$	93.79	$T \preceq_{id} S$	10.17
mps2_58_s0	$S \preceq_{id} T$	10.11	$S \preceq \alpha T$	92.56	$S \preceq \alpha T$	76.19	$T \preceq \alpha S$	76.64	$T \preceq \alpha S$	86.53	$T \preceq_{id} S$	10.3
mps2_58_s3	$S \preceq_{id} T$	6.32	$S \preceq \alpha T$	61.04	$S \preceq \alpha T$	62.16	$T \preceq \alpha S$	65.21	$T \preceq \alpha S$	70.27	$T \preceq_{id} S$	6.44
mps3_33_s3	$S \preceq_{id} T$	5.27	$S \preceq \alpha T$	35.37	$S \preceq \alpha T$	40.08	$T \preceq \alpha S$	30.43	$T \preceq \alpha S$	29.32	$T \preceq_{id} S$	5.23
mps3_34_s0	$S \preceq_{id} T$	5.96	$S \preceq \alpha T$	45.22	$S \preceq \alpha T$	42.44	$T \preceq \alpha S$	42.09	$T \preceq \alpha S$	40.99	$T \preceq_{id} S$	5.89
mps3_38_s0	$S \preceq_{id} T$	7.36	$S \preceq \alpha T$	52.28	$S \preceq \alpha T$	51.01	$T \preceq \alpha S$	32.14	$T \preceq \alpha S$	32.57	$T \preceq_{id} S$	7.12
mps3_41_s3	$S \preceq_{id} T$	7.06	$S \preceq \alpha T$	29.94	$S \preceq \alpha T$	31.84	$T \preceq \alpha S$	34.34	$T \preceq \alpha S$	33.95	$T \preceq_{id} S$	6.99
mps3_42_s3	$S \preceq_{id} T$	7.1	$S \preceq \alpha T$	34.12	$S \preceq \alpha T$	29.85	$T \preceq \alpha S$	25.18	$T \preceq \alpha S$	33.84	$T \preceq_{id} S$	7.14
mps3_45_s0	$S \preceq_{id} T$	8.93	$S \preceq \alpha T$	56.64	$S \preceq \alpha T$	51.56	$T \preceq \alpha S$	41.95	$T \preceq \alpha S$	33.66	$T \preceq_{id} S$	8.82
mps3_45_s3	$S \preceq_{id} T$	5.38	$S \preceq \alpha T$	29.25	$S \preceq \alpha T$	29.33	$T \preceq \alpha S$	27.11	$T \preceq \alpha S$	27.88	$T \preceq_{id} S$	5.33
mps3_46_s3	$S \preceq_{id} T$	5.33	$S \preceq \alpha T$	31.39	$S \preceq \alpha T$	34.8	$T \preceq \alpha S$	23.93	$T \preceq \alpha S$	22.65	$T \preceq_{id} S$	5.37
mps3_49_s0	$S \preceq_{id} T$	9.59	$S \preceq \alpha T$	71	$S \preceq \alpha T$	68.14	$T \preceq \alpha S$	73.4	$T \preceq \alpha S$	74.45	$T \preceq_{id} S$	9.46
mps3_49_s3	$S \preceq_{id} T$	5.86	$S \preceq \alpha T$	43.3	$S \preceq \alpha T$	48.14	$T \preceq \alpha S$	44.73	$T \preceq \alpha S$	39.46	$T \preceq_{id} S$	5.89
mps3_50_s0	$S \preceq_{id} T$	9.91	$S \preceq \alpha T$	64.25	$S \preceq \alpha T$	54.84	$T \preceq \alpha S$	84.53	$T \preceq \alpha S$	62.79	$T \preceq_{id} S$	9.7
mps3_50_s3	$S \preceq_{id} T$	6.39	$S \preceq \alpha T$	55.01	$S \preceq \alpha T$	46.67	$T \preceq \alpha S$	48.33	$T \preceq \alpha S$	49.02	$T \preceq_{id} S$	6
mps3_54_s3	$S \preceq_{id} T$	5.97	$S \preceq \alpha T$	51.36	$S \preceq \alpha T$	49.42	$T \preceq \alpha S$	14.52	$T \preceq \alpha S$	19.42	$T \preceq_{id} S$	5.71
mps3_58_s0	$S \preceq_{id} T$	10.44	$S \preceq \alpha T$	78.93	$S \preceq \alpha T$	95.22	$T \preceq \alpha S$	68.7	$T \preceq \alpha S$	92.07	$T \preceq_{id} S$	10.18
mps3_62_s0	$S \preceq_{id} T$	13.45	$S \preceq \alpha T$	91.56	$S \preceq \alpha T$	92.41	$T \preceq \alpha S$	87.8	$T \preceq \alpha S$	85.97	$T \preceq_{id} S$	13.1
mps3_63_u3	$S \preceq_{id} T$	24.67	$S \preceq \alpha T$	194.53	$S \preceq \alpha T$	196.87	$T \preceq_{\rho_\alpha \wedge sk} S$	50.94	$T \preceq_{\rho_\alpha \wedge sk} S$	50.38	$T \preceq_{id} S$	24.64
mps3_64_u3	$S \preceq_{id} T$	53.1	$S \preceq \alpha T$	457.96	$S \preceq \alpha T$	484.61	$T \preceq_{\rho_\alpha \wedge sk} S$	166.27	$T \preceq_{\rho_\alpha \wedge sk} S$	169.04	$T \preceq_{id} S$	52.67
mps4_36_u0	$S \preceq_{id} T$	7.22	$S \preceq \alpha T$	217.19	$S \preceq \alpha T$	48.78	$T \preceq \alpha S$	89.67	$T \preceq \alpha S$	164.98	$T \preceq_{\rho_\alpha \wedge sk} S$	27.27
mps4_37_u0	$S \preceq_{id} T$	6.73	$S \preceq \alpha T$	45.33	$S \preceq \alpha T$	44.13	$T \preceq \alpha S$	41.89	$T \preceq \alpha S$	36.19	$T \preceq_{id} S$	6.59
mps4_39_u3	$S \preceq_{id} T$	14.64	$S \preceq \alpha T$	125.57	$S \preceq \alpha T$	120.03	$T \preceq \alpha S$	84.97	$T \preceq \alpha S$	92.06	$T \preceq_{id} S$	14.68
mps4_44_u3	$S \preceq_{id} T$	32.07	$S \preceq \alpha T$	165.83	$S \preceq \alpha T$	216.61	$T \preceq \alpha S$	109.93	$T \preceq \alpha S$	109.13	$T \preceq_{id} S$	32.72
mps4_46_u0	$S \preceq_{id} T$	8.3	$S \preceq \alpha T$	42.03	$S \preceq \alpha T$	39.82	$T \preceq \alpha S$	43.37	$T \preceq \alpha S$	40.87	$T \preceq_{id} S$	8.27
mps4_47_u3	$S \preceq_{id} T$	16.79	$S \preceq \alpha T$	130.55	$S \preceq \alpha T$	147.91	$T \preceq \alpha S$	119.25	$T \preceq \alpha S$	97.76	$T \preceq_{id} S$	16.58
mps4_52_s3	$S \preceq_{id} T$	5.51	$S \preceq \alpha T$	38.16	$S \preceq \alpha T$	32.75	$T \preceq \alpha S$	44.96	$T \preceq \alpha S$	41.46	$T \preceq_{id} S$	5.39
mps4_60_s0	$S \preceq_{id} T$	39.91	$S \preceq \alpha T$	365.53	$S \preceq \alpha T$	353	$T \preceq \alpha S$	344.47	$T \preceq \alpha S$	302.85	$T \preceq_{id} S$	40.84
mps4_60_s3	$S \preceq_{id} T$	25.09	$S \preceq \alpha T$	249.36	$S \preceq \alpha T$	243.1	$T \preceq \alpha S$	159.49	$T \preceq \alpha S$	166.34	$T \preceq_{id} S$	25.24
mps4_61_s0	$S \preceq_{id} T$	10.24	$S \preceq \alpha T$	65.14	$S \preceq \alpha T$	68.52	$T \preceq \alpha S$	57.94	$T \preceq \alpha S$	57.92	$T \preceq_{id} S$	10.32
mps4_64_s3	$S \preceq_{id} T$	35.41	$S \preceq \alpha T$	267.25	$S \preceq \alpha T$	251.15	$T \preceq \alpha S$	61.64	$T \preceq \alpha S$	60.15	$T \preceq_{id} S$	34.84
mps5_35_s0	$S \preceq_{id} T$	5.6	$S \preceq \alpha T$	135.36	$S \preceq \alpha T$	37.59	$T \preceq \alpha S$	66.19	$T \preceq \alpha S$	172.94	$T \preceq_{\rho_\alpha \wedge sk} S$	21.89
mps5_36_s3	$S \preceq_{id} T$	5.66	$S \preceq \alpha T$	43.54	$S \preceq \alpha T$	37.09	$T \preceq \alpha S$	44.5	$T \preceq \alpha S$	37.91	$T \preceq_{id} S$	5.77
mps5_37_s0	$S \preceq_{id} T$	5.37	$S \preceq \alpha T$	46.87	$S \preceq \alpha T$	45.75	$T \preceq \alpha S$	32.29	$T \preceq \alpha S$	31.79	$T \preceq_{id} S$	5.26
mps5_39_s3	$S \preceq_{id} T$	15.78	$S \preceq \alpha T$	104.21	$S \preceq \alpha T$	89.5	$T \preceq \alpha S$	74.92	$T \preceq \alpha S$	102.1	$T \preceq_{id} S$	15.43
mps5_41_s0	$S \preceq_{id} T$	5.3	$S \preceq \alpha T$	32.81	$S \preceq \alpha T$	30.45	$T \preceq \alpha S$	34.13	$T \preceq \alpha S$	28.85	$T \preceq_{id} S$	5.16
mps5_43_s3	$S \preceq_{id} T$	26.65	$S \preceq \alpha T$	200.42	$S \preceq \alpha T$	210.09	$T \preceq \alpha S$	157.04	$T \preceq \alpha S$	132.25	$T \preceq_{id} S$	26.5
mps5_44_s0	$S \preceq_{id} T$	35.87	$S \preceq \alpha T$	283.5	$S \preceq \alpha T$	291.14	$T \preceq \alpha S$	274.21	$T \preceq \alpha S$	303.94	$T \preceq_{id} S$	36.26
mps5_45_s0	$S \preceq_{id} T$	8.44	$S \preceq \alpha T$	66.99	$S \preceq \alpha T$	69.62	$T \preceq \alpha S$	46.51	$T \preceq \alpha S$	50.73	$T \preceq_{id} S$	8.38
mps5_46_s0	$S \preceq_{id} T$	8.36	$S \preceq \alpha T$	61.68	$S \preceq \alpha T$	66.03	$T \preceq \alpha S$	32.89	$T \preceq \alpha S$	47.04	$T \preceq_{id} S$	8.26
mps5_47_s3	$S \preceq_{id} T$	20.51	$S \preceq \alpha T$	173.93	$S \preceq \alpha T$	178.49	$T \preceq \alpha S$	110.35	$T \preceq \alpha S$	110.08	$T \preceq_{id} S$	20.43
mps5_52_s3	$S \preceq_{id} T$	8.51	$S \preceq \alpha T$	63.57	$S \preceq \alpha T$	71.86	$T \preceq \alpha S$	71.36	$T \preceq \alpha S$	70.13	$T \preceq_{id} S$	8.58
mps5_53_s0	$S \preceq_{id} T$	7.77	$S \preceq \alpha T$	71.27	$S \preceq \alpha T$	66.18	$T \preceq \alpha S$	34.58	$T \preceq \alpha S$	35.49	$T \preceq_{id} S$	7.87
mps5_57_s0	$S \preceq_{id} T$	7.89	$S \preceq \alpha T$	58.96	$S \preceq \alpha T$	56.08	$T \preceq \alpha S$	51.87	$T \preceq \alpha S$	53.13	$T \preceq_{id} S$	7.89
mps5_59_s3	$S \preceq_{id} T$	20.95	$S \preceq \alpha T$	189.17	$S \preceq \alpha T$	174.64	$T \preceq \alpha S$	186.28	$T \preceq \alpha S$	187.45	$T \preceq_{id} S$	20.79
mps5_61_s0	$S \preceq_{id} T$	11.88	$S \preceq \alpha T$	89.11	$S \preceq \alpha T$	80.67	$T \preceq \alpha S$	82.29	$T \preceq \alpha S$	92.25	$T \preceq_{id} S$	12.04
mps5_61_s3	$S \preceq_{id} T$	6.17	$S \preceq \alpha T$	46.61	$S \preceq \alpha T$	43.05	$T \preceq \alpha S$	16.06	$T \preceq \alpha S$	16.49	$T \preceq_{id} S$	6.06
mps5_62_s3	$S \preceq_{id} T$	6.09	$S \preceq \alpha T$	36.21	$S \preceq \alpha T$	43.56	$T \preceq \alpha S$	16.45	$T \preceq \alpha S$	16.19	$T \preceq_{id} S$	5.93

Table 2: Abstract simulation for 3 types of optimizations (fragment).

relations from valid ∀∃-formulas in Linear Rational Arithmetic. The third contribution is our implementation of SIMABS and AE-VAL that supports (and not limited by) different LLVM optimizer and mutator passes.

In future, we plan to use simulation relations produced by SIMABS to migrate the safe inductive invari-

ants across program transformations, thus achieving a *property-directed equivalence* between programs [7]. AE-VAL can be used as a stand-alone solver. We believe, it has a great potential, and in future we plan to apply it for other tasks (e.g., [10]) of realizability and synthesis.

References

[1] A. Albarghouthi, A. Gurfinkel, and M. Chechik. UFO: A Framework for Abstraction- and Interpolation-Based Software Verification. In *CAV*, volume 7358 of *LNCS*, pages 672–678. Springer, 2012.

[2] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[3] E. M. Clarke, D. Kroening, and K. Yorav. Behavioral consistency of C and Verilog programs using bounded model checking. In *DAC*, pages 368–371. ACM, 2003.

[4] Ştefan Ciobâcă, D. Lucanu, V. Rusu, and G. Rosu. A language-independent proof system for mutual program equivalence. In *ICFEM*, volume 8829 of *LNCS*, pages 75–90. Springer, 2014.

[5] L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[6] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation preorders. In *CAV*, volume 575 of *LNCS*, pages 255–265. Springer, 1991.

[7] G. Fedyukovich, A. Gurfinkel, and N. Sharygina. Incremental verification of compiler optimizations. In *NFM*, volume 8430 of *LNCS*, pages 300–306. Springer, 2014.

[8] G. Fedyukovich, O. Sery, and N. Sharygina. eVolCheck: Incremental Upgrade Checker for C. In *TACAS*, volume 7795 of *LNCS*, pages 292–307. Springer, 2013.

[9] D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich. Automating regression verification. In *ASE*, pages 349–360. ACM, 2014.

[10] A. Gacek, A. Katis, M. W. Whalen, J. Backes, and D. D. Cofer. Towards Realizability Checking of Contracts Using Theories. In *NFM*, volume 9058 of *LNCS*, pages 173–187. Springer, 2015.

[11] R. Gjomemo, K. S. Namjoshi, P. H. Phung, V. N. Venkatakrishnan, and L. D. Zuck. From verification to optimizations. In *VMCAI*, volume 8931, pages 300–317. Springer, 2015.

[12] B. Godlin and O. Strichman. Regression verification. In *DAC*, pages 466–471. ACM, 2009.

[13] A. Gurfinkel, S. Chaki, and S. Sapra. Efficient Predicate Abstraction of Program Summaries. In *NFM*, volume 6617 of *LNCS*, pages 131–145. Springer, 2011.

[14] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462, 1995.

[15] M. Kawaguchi, S. K. Lahiri, and H. Rebelo. Conditional equivalence. Technical Report MSR-TR-2010-119, Microsoft Research, 2010.

[16] A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-Based Model Checking for Recursive Programs. In *CAV*, volume 8559 of *LNCS*, pages 17–34, 2014.

[17] S. K. Lahiri, K. L. McMillan, R. Sharma, and C. Hawblitzel. Differential assertion checking. In *FSE*, pages 345–355. ACM, 2013.

[18] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.

[19] R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.

[20] K. S. Namjoshi. Lifting Temporal Proofs through Abstractions. In *VMCAI*, volume 2575 of *LNCS*, pages 174–188. Springer, 2003.

[21] K. S. Namjoshi and L. D. Zuck. Witnessing program transformations. In *SAS*, volume 7935 of *LNCS*, pages 304–323. Springer, 2013.

[22] G. C. Necula. Translation validation for an optimizing compiler. In *PLDI*, pages 83–94. ACM, 2000.

[23] T. Nipkow. Linear quantifier elimination. *J. Autom. Reasoning*, 45(2):189–212, 2010.

[24] S. Person, M. B. Dwyer, S. G. Elbaum, and C. S. Pasareanu. Differential symbolic execution. In *FSE*, pages 226–237. ACM, 2008.

[25] A.-D. Phan, N. Bjørner, and D. Monniaux. Anatomy of alternating quantifier satisfiability (work in progress). In *SMT*, volume 20 of *EPiC Series*, pages 120–130. EasyChair, 2012.

[26] T. Skolem. Über die mathematische logik. In *Norsk Matematisk Tidsskrift 10*, pages 125–142, 1928.

[27] G. Yang, S. Khurshid, S. Person, and N. Rungta. Property differencing for incremental checking. In *ICSE*, pages 1059–1070. ACM, 2014.