# Automated Discovery of Simulation Between Programs*

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

[1] Formal Verification Lab, University of Lugano, Switzerland,
{grigory.fedyukovich,natasha.sharygina}@usi.ch
[2] SEI/CMU, USA, arie@cmu.com

**Abstract.** Deciding equivalence between two programs (called a source and a target program) is often reduced to finding a simulation relation between them. This is computationally expensive and often requires a manual guidance. In this paper, we propose an abstraction-refinement-guided approach, called SimAbs, to automatically construct a simulation relation between the source program and an abstraction of the target program. In our approach both the abstraction and the simulation relation are discovered automatically, and deciding whether a given relation is a simulation relation is reduced to deciding validity of a $\forall\exists$-formula. We present a novel algorithm for deciding such formulas using an SMT-solver. In addition to deciding validity, our algorithm constructs a witnessing Skolem relation. These relations enable the refinement-step of SimAbs. We have implemented SimAbs using UFO framework and Z3 SMT-solver and applied it to finding simulation relations between C programs from the Software Verification Competition. Our empirical results show that SimAbs is efficient at finding a simulation relation.

## 1 Introduction

There is a growing interest in the problems of regression verification and program equivalence checking [21, 22, 11, 14, 10, 20, 9]. In general, the problem is to identify (and check) the condition under which two programs (referred to as the *source* ($S$) and the *target* ($T$)) are equivalent (i.e., satisfy the same properties). 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., [10, 9]).

For example, in [9] we 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

* 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 Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **18 OCT 2014** | **N/A** | |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **Automated Discovery of Simulation Between Programs** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| **Arie Gurfinkel Grigory Fedyukovich /Natasha Sharygina** | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| **Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213** | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release, distribution unlimited.**

13. SUPPLEMENTARY NOTES
**The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | |
| **unclassified** | **unclassified** | **unclassified** | **SAR** | **19** | |

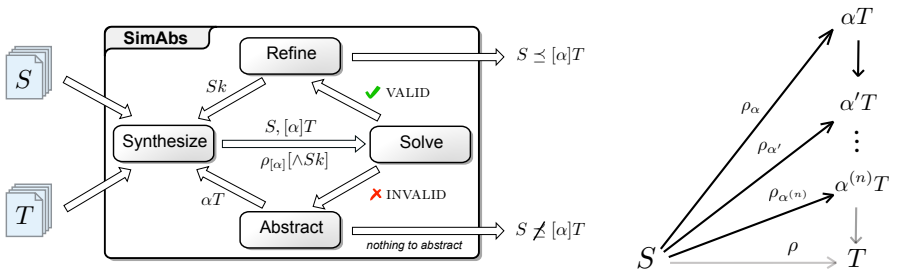



Fig. 1: (a) SimAbs and (b) its search space.

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 [9] 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 $S$.

Namjoshi et al. [19, 20] show 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., [20] expects the relation to be provided by the user). Moreover, their result only applies when $T$ actually simulates $S$, whereas we are interested in cases where $T$ is obtained by a small modification from $S$, but might not simulate $S$.

In this paper, we address two problems: (1) the challenge of automatically constructing a simulation relation for 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 $S$.

Our main contribution is an iterative abstraction-refinement approach called SimAbs, illustrated in Fig. 1a. 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 $\rho$, 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 $\rho$ between $S$ and $T$ (**Synthesize** step). Second, it checks whether $\rho$ 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 $T$ and synthesizes new relation $\rho$ (**Refine** step). Otherwise, it looks for a better abstraction $\alpha$ 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. 1b. SimAbs explores the space of abstractions of $T$, starting with the most general abstraction (called $\alpha T$ in Fig. 1b) that simulates $S$, and iteratively refines it until no further refinement is possible.

In contrast to existing algorithms for checking whether a given relation $\rho$ is a simulation relation (e.g., [18, 13, 21]), we reduce the problem to deciding va-

lidity of a $\forall\exists$-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 [23]. However, in addition to deciding validity, we also extract a Skolem relation to witness the existential quantifier. This Skolem relation (called *Sk* in Fig. 1a) is the key to the **Refine** step of SimAbs.

We implemented SimAbs and AE-VAL on the top of the UFO framework [1, 15] and an SMT-solver Z3 [8], respectively. We have evaluated SimAbs by discovering simulation relations between programs and their optimizations (done by LLVM). Our results show that SimAbs is able to efficiently discover nontrivial simulation relations between the original and the optimized versions for most of the benchmarks.

The rest of the paper is structured as follows. After defining notation in Sect. 2, we describe how to reduce the problem of checking a simulation relation to a validity check of a specific $\forall\exists$-formula in Sect. 3. The algorithm AE-VAL designed to solve such formulas and extract Skolem relation is presented in Sect. 4. Sect. 5 provides implementation details for the algorithm SimAbs. Sect. 6 presents an evaluation of our implementation of SimAbs. Sect. 7 compares our work with the related one, and Sect. **??** concludes the paper.

## 2 Background

In this section, we give the necessary definitions of a program, a transition system, and a simulation relation.

**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 current, next-state, and local* variables*, respectively; $Init$ is a formula over $V$ that defines the* initial state*, and $Tr$ is a formula over $Var$ that denotes the* transition relation*.*

**Definition 2.** *A program $P = \langle Var, Init, Tr\rangle$ induces a* transition system *$\mathcal{T} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}\rangle$, where $\mathcal{S}$ is a set of valuations to all variables in $V$ (i.e.,* states*), $\mathcal{I} = \{\vec{s} \in \mathcal{S} \mid \vec{s} \models Init\}$ is the set of* initial states*, $\mathcal{R} = \{(\vec{s}, \vec{t}) \mid \vec{s}, \vec{t} \in \mathcal{S}, \exists \vec{l} \in L \cdot Tr(\vec{s}, \vec{l}, \vec{t'})\}$ is a* transition relation*. Throughout, we write $\mathcal{S}'$ for $\{\vec{s'} \mid \vec{s} \in \mathcal{S}\}$, and $\vec{t'}$ for $\vec{t}(x')$.*

**Definition 3.** *Program $P_1 = \langle V \cup L_1 \cup V', Init_1, Tr_1\rangle$ is an* abstraction *of program $P_2 = \langle V \cup L_2 \cup V', Init_2, Tr_2\rangle$ iff*

$$Init_2 \implies Init_1 \quad (\exists \vec{l_2} \in L_2 \cdot Tr_2(\vec{s}, \vec{l_2}, \vec{s'})) \implies (\exists \vec{l_1} \in L_1 \cdot Tr_1(\vec{s}, \vec{l_1}, \vec{s'})) \quad (1)$$

An example of abstraction `AQ` of `Q` is shown on Fig. 2b-2c. In `AQ`, nondeterminism is introduced on the level go the input parameters.

**Definition 4 (cf.[20]).** *Given two transition systems $S$ and $T$, a relation $\rho \subseteq \mathcal{S}_S \times \mathcal{S}_T$ is a* simulation relation *if (1) every state in $\mathcal{I}_S$ is related by $\rho$ to some state in $\mathcal{I}_T$, and (2) for all states $\vec{s}$, $\vec{s'}$ and $\vec{t}$, such that $(\vec{s}, \vec{t}) \in \rho$ and $(\vec{s}, \vec{s'}) \in \mathcal{R}_S$ there is some state in $\mathcal{S}_T$ such that $(\vec{t}, \vec{t'}) \in \mathcal{R}_T$ and $(\vec{s'}, \vec{t'}) \in \rho$.*

```
int P (int a,
    int b, int c)
{
  int m = b + c;
  int ret = m + a;
  return ret;

}
```

(a) Source program `P`

```
int Q (int a,
    int b, int c)
{
  int m = a + b;
  int n = c + 1;
  int ret = m - n;
  return ret;
}
```

(b) Target program `Q`

```
int AQ (int a,
    int b, * )
{
  int m = a + b;
  int n = c + 1;
  int ret = m - n;
  return ret;
}
```

(c) Abstraction of `Q`

Fig. 2: Fragments of the three programs in C

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 $\rho$. We write $T_1 \preceq T_2$ to indicate existence of a simulation between $T_1$ and $T_2$. We extend the notion of a simulation from transition systems to programs in the usual way: a program $P_1$ is simulated by a program $P_2$ iff their corresponding transition systems simulate each other.

**Lemma 1.** *If $P_1$ is an abstraction of $P_2$ then the corresponding transition systems $T_2 \preceq_{id} T_1$, where id is the identity relation.*

**Lemma 2.** *If $T_1 \preceq T_2$ and $T_2 \preceq T_3$ then $T_1 \preceq T_3$.*

## 3 From Simulation to Satisfiability

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

### 3.1 Deciding Simulation Symbolically

Let $S(\vec{s}, \vec{x}, \vec{s'})$ and $T(\vec{t}, \vec{y}, \vec{t'})$ be formulas denoting transition relations of two programs, where $\vec{s}$ and $\vec{t}$, $\vec{s'}$ and $\vec{t'}$, $\vec{x}$ and $\vec{y}$ are current-state, next-state, and local variables, respectively. Let $\rho(\vec{s}, \vec{t})$ denote a relation between state variables of $S$ and $T$. For simplicity, we omit the arguments and simply write $S$, $T$, and $\rho$, when the arguments are clear from the context.

**Lemma 3.** *Given programs $S$ and $T$, a relation $\rho$ is a simulation relation between $S$ and $T$ iff*

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

The left-hand-side of implication (2) represents the set of all behaviors of $S$ and the set of all matched input conditions. The right-hand-side of (2) represents the existence of a matching behavior in $T$.

*Example 1.* Consider two programs `P` and `Q` shown on Fig. 2a and Fig. 2b, respectively. Their corresponding transition relations are shown in (3):

$$P \equiv m_P = b_P + c_P \wedge ret_P = m_P + a_P \tag{3}$$
$$Q \equiv m_Q = a_Q + b_Q \wedge n_Q = c_Q + 1 \wedge ret_Q = m_Q + n_Q$$

where the subscript indicates which program the corresponding variables is defined in.

Let $\mathcal{C}$ and $\mathcal{N}$ be relations between current and next-state variables of $P$ and $Q$, respectively, defined as follows:

$$\mathcal{C} \equiv a_P = a_Q \wedge b_P = b_Q \wedge c_P = c_Q \qquad \mathcal{N}' \equiv ret_P = ret_Q \tag{4}$$

Note that in general, unlike in our simplified definition in Section 2, current and next-state variables of a program can differ, requiring us to split the simulation relation into two components.

$\mathcal{C}$ and $\mathcal{N}$ is a simulation relation iff the following formula is valid:

$$\mathcal{C} \wedge (\exists m_P \cdot P) \implies \exists ret_Q, m_Q, n_Q \cdot Q \wedge \mathcal{N} \tag{5}$$

Note that since `Q` is deterministic, the existential quantifiers in (5) are eliminated trivially by substitution. In our example, (5) simplifies to $0 = 1$, hence $\mathcal{C}$ and $\mathcal{N}$ is *not* a simulation relation between $P$ and $Q$. □

### 3.2 Abstract Simulation

In this section, we show how to check whether a given relation $\rho$ is a simulation relation between a program $S$ and an abstraction $\alpha T$ of a program $T$. We restrict our attention to existential abstraction [6], although the results extend easily to predicate abstraction as well. Our key result is to show that simulation checking can be done without constructing an abstraction explicitly.

Let $T$ be a transition relation of a program, and $U$ a sub-set of the state-variables of $T$. An existential abstraction, $\alpha^{\exists}_U$, of $T$ abstracts from $T$ all variables in $U$. Formally, $\alpha^{\exists}_U(T) \equiv \exists U, U' \cdot T(\vec{t}, \vec{y}, \vec{t'})$, where $U \subseteq \vec{t}$. For example, the program `AQ` in Fig. 2c is an existential abstraction of the program `Q` in Fig. 2b, where $U = \{\mathtt{c}\}$.

Deciding whether a given relation $\rho_\alpha$ is a simulation between a concrete program $S$ and an abstract program $\alpha T$ can be done without computing the abstraction explicitly. Intuitively, the variables that are abstracted away are simply treated as local variables of $T$.

**Lemma 4.** *Let $S(\vec{s}, \vec{x}, \vec{s'})$ and $T(\vec{t}, \vec{y}, \vec{t'})$ be two programs. Let $U \subseteq \vec{t}$ be the set of abstracted variables and $\vec{t_1} = \vec{t} \setminus U$. A relation $\rho_\alpha$ is a simulation relation between $S$ and $\alpha^{\exists}_U(T)$ iff*

$$\rho_\alpha(\vec{s}, \vec{t_1}) \wedge S(\vec{s}, \vec{x}, \vec{s'}) \implies \exists \vec{t'}_1, \vec{y}, U, U' \cdot T(\vec{t}, \vec{y}, \vec{t'}) \wedge \rho_\alpha(\vec{s'}, \vec{t'_1}) \tag{6}$$

*Proof.* Immediate from (2) and the definition of existential abstraction. □

Recall that in Example 1, program P was shown to be not simulated by Q via identity relation. Interestingly, this result is still usefull to obtain a valid simulation relation between P and Q by creating implicit abstraction of Q and further refining it. We will demonstrate this 2-steps procedure in Example 2.

*Example 2.* As a first (*abstraction*) step, we create an implicit abstraction of Q by choosing a state-variable (let it be c) to be existentially quantified. Note that the produced abstraction is equivalent to AQ. Instead of encoding a transition system $AQ$ from scratch (similarly to (3)), we let it be equal to $AQ \equiv \exists c_Q \cdot Q$

Relation $\mathcal{C}$ and $\mathcal{N}$ (disproven to be a simulation relation for P and Q) are abstracted away in correspondence with $AQ$:

$$\mathcal{C}_\alpha \equiv a_P = a_Q \wedge b_P = b_Q \qquad \mathcal{N}_\alpha \equiv ret_P = ret_Q \tag{7}$$

$\mathcal{C}$ and $\mathcal{N}$ is a simulation relation between P and AQ iff the following formula is valid:

$$\mathcal{C} \wedge (\exists m_P \cdot P) \implies \exists c_Q, ret_Q, m_Q, n_Q \cdot Q \wedge \mathcal{N} \tag{8}$$

Clearly, (8) is valid iff there is a Skolem function for the existentially quantified variable $c_Q$. Note that $sk_{c_Q}(c_P) = -c_P - 1$ is such a function:

$$\mathcal{C} \wedge (\exists m_P \cdot P) \implies (c_Q = -c_P - 1 \implies \exists ret_Q, m_Q, n_Q \cdot Q \wedge \mathcal{N}) \tag{9}$$

As a second (*refinement*) step, $sk_{c_Q}$ is used to strengthen a simulation relation $\mathcal{C}$ and $\mathcal{N}$ between P and AQ.

$$\mathcal{C}^{ext}_\alpha \equiv a_P = a_Q \wedge b_P = b_Q \wedge c_Q = -c_P - 1 \qquad \mathcal{N}^{ext}_\alpha \equiv ret_P = ret_Q \tag{10}$$

Note that $\mathcal{C}^{ext}_\alpha$ and $\mathcal{N}^{ext}_\alpha$ is a simulation relation between P and Q. □

Detailed definition of the Skolem function, its generalization and application to the simulation-relation-checking problem is given in Sect. 3.3.

### 3.3 Refining Simulation by Skolem Relations

In this section, we show how to use a Skolem relation that is witnessing the validity of the abstract simulation check (6) to refine an abstract simulation relation. We begin with the classical definition of a Skolem function:

**Definition 5.** *Given a valid formula* $\forall x \cdot \exists y \cdot f(x, y)$*, a* Skolem function *for* $y$*,* $sk_y(x)$ *is a function such that* $\exists y \cdot f(x, y) \iff f(x, sk_y(x))$.

We now relax the definition by allowing the relationship between $y$ and $x$ to be an arbitrary relation:

**Definition 6.** *Given a valid formula* $\forall x \cdot \exists y \cdot f(x, y)$*, a* Skolem relation *for* $y$ *is a relation* $Sk_y(x, y)$ *such that* $\exists y \cdot f(x, y) \iff (Sk_y(x, y) \implies f(x, y))$.

To see that Def. 6 is a generalization of Def. 5, let $sk_y(x)$ be a Skolem function of $y$ in $\exists y \cdot f(x,y)$. Then, $Sk_y(x,y) \equiv (y = sk_y(x))$ is the corresponding Skolem relation.

$$\Big[Sk_y(x,y) \implies f(x,y)\Big] \equiv \Big[y = sk_y(x) \implies f(x,y)\Big] \equiv \Big[f(x, sk_y(x))\Big] \quad (11)$$

Clearly, the opposite is not true – not every relation can be represented by a Skolem function.

As shown above, a Skolem relation eliminates an existential quantifier in a valid $\forall\exists$-formula. In fact, validity of a $\forall\exists$-formula is equivalent to an existence of an appropriate Skolem function (or relation). We now adapt this to the case of simulation checking.

**Theorem 1.** *Let $S(\vec{s}, \vec{x}, \vec{s'})$ and $T(\vec{t}, \vec{y}, \vec{t'})$ be two programs such that $S \preceq T$, and $U \subseteq \vec{t}$. Let $\rho_\alpha$ be a simulation relation such that $S \preceq_{\rho_\alpha} \alpha^{\exists}_{U}(T)$. Then, there exists a relation $Sk(\vec{s}, U)$ such that (a) $\rho_\alpha \wedge Sk$ is a simulation relation between $S$ and $T$ and (b) $Sk$ is a Skolem relation for $U$ in* (6).[3]

Recall that by Lemma 3, simulation checking between transition systems $S$ and $T$ is reduced to deciding validity of formula (2). In the next section, we will focus on solving (2) by iterative quantifier elimination and present a generalized algorithm for it.

## 4 Deciding Validity of $\forall\exists$-formulas and Extracting Skolem Relations

In this section, we present a novel algorithm, AE-VAL, for deciding validity of $\forall\exists$-formulas. Without loss of generality, we restrict the input formula to the form $S(\vec{x}) \implies \exists \vec{y} \cdot T(\vec{x}, \vec{y})$, where $S$ has no universal quantifiers, and $T$ is quantifier-free.

### 4.1 Deciding Validity of $\forall\exists$-formulas

Our algorithm is based on a notion of Model-Based Projection (MBP), introduced in [15], that under-approximates existential quantification. Let $M$ be a model of a formula $T(\vec{x}, \vec{y})$. Then, $T_{\vec{y}}(\vec{x})$ is an MBP if the following two conditions hold: (a) $M \models T_{\vec{y}}(\vec{x})$ and (b) $T_{\vec{y}}(\vec{x}) \implies \exists \vec{y} \cdot T(\vec{x}, \vec{y})$. That is, the only $\vec{x}$ variables appear in $T$, $T_{\vec{y}}$, $M$ is a model of $T_{\vec{y}}$, and $T_{\vec{y}}$ is an implicant of $T$. Furthermore, when $\vec{y}$ and $T$ are fixed, MBP is finite. That is, 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 $(\exists \vec{y} \cdot T(\vec{x}, \vec{y})) = \bigvee_{i=1}^{n} T_{\vec{y}_i}(\vec{x})$ for some $n$. In our implementation, we are using an MBP function from [15] for LRA that is based on Loos-Weispfenning [16] quantifier elimination. Additionally, we assume that for each projection $T_{\vec{y}_i}$ the MBP procedure produces a *local Skolem* relation $\phi_i(\vec{x}, \vec{y})$ such that $\phi_i(\vec{x}, \vec{y}) \implies (T_{\vec{y}_i}(\vec{x}) \implies T(\vec{x}, \vec{y}))$. Local Skolems are a natural by-product of the MBP algorithm in [15]. We write

[3] Due to lack of space, the proof is moved to Appendix A.

**Algorithm 1:** AE-VAL $(S(\vec{x}), \exists \vec{y} \cdot T(\vec{x}, \vec{y}))$

**Input**: $S(\vec{x}), \exists \vec{y} \cdot T(\vec{x}, \vec{y})$
**Output**: $res \in \{\text{VALID, INVALID}\}$ of $S(\vec{x}) \implies \exists \vec{y} \cdot T(\vec{x}, \vec{y})$
**Data**: Incremental SmtSolver, Model $M$,
Model-based projection $T_{\vec{y}}(\vec{x})$, local Skolem relation $\phi_i(\vec{x}, \vec{y})$

```
1  SmtAssert(S(x));
2  while true do
3  |  res ← SmtSolve();
4  |  if (isUNSAT(res)) then return VALID;
5  |  SmtPush();
6  |  SmtAdd(T(x, y));
7  |  res ← SmtSolve();
8  |  if (isUNSAT(res)) then return INVALID;
9  |  M ← SmtGetModel(T(x, y)));
10 |  T_y, φ(x, y) ← GetMBP(y, M, T(x, y)));
11 |  SmtPop();
12 |  SmtAdd(¬T_y);
13 end
```

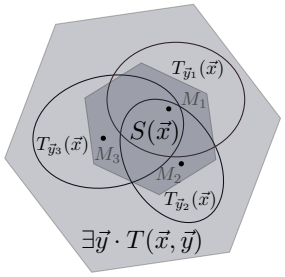



Fig. 3: Illustration to Example 3 with 3 iterations of AE-VAL ($S, \exists T$ - hexagons, MBPs - ovals, models - points)

$(T_i, \phi_i) \leftarrow \text{GetMBP}(\vec{y}, M, T(\vec{x}, \vec{y}))$ for an MBP algorithm that takes a formula $T$, a model $M$ of $T$ and a vector of varialbes $\vec{y}$, and returns a projection $T_i$ of $T$ that covers $M$ and the corresponding local Skolem $\phi_i$.

AE-VAL is shown in Alg. 1. Given two formulas $S(\vec{x})$ and $\exists \vec{y} \cdot T(\vec{x}, \vec{y})$ it decides the validity of $S(\vec{x}) \implies \exists \vec{y} \cdot T(\vec{x}, \vec{y})$. AE-VAL enumerates the models of $S \wedge T$. In each iteration, it first checks whether there $S$ is non-empty (line 3) and then looks for a model $M$ of $S \wedge T$ (line 9). If $M$ is found, AE-VAL constructs an MBP $T_{\vec{y}}$ of $T$ and $M$ (line 10) and blocks all models contained in $T_{\vec{y}}$ from $S$ (line 12). It iterates until either it discovers that there is a model of $S$ that can not be extended to a model of $T$ (line 8); or all models of $S(\vec{x})$ are blocked (line 4). In the first case, the formula is invalid. In the second, every model of $S$ has been extened 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 an MBP $T_{\vec{y}_1}$. Then, it picks a model $M_2$ and generalizes it to an MBP $T_{\vec{y}_2}$. Finally, it picks a model $M_3$ and generalizes it to $T_{\vec{y}_3}$. At this point, all models of $S$ are covered by $\vec{y}$-free implicants of $T$, and the algorithm terminates. We demonstrate this further in the following example.

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

$$S \equiv (a = b + 2) \tag{12}$$
$$T \equiv (a' = a + b) \wedge (a' = 1 \implies b' = c') \wedge (a' = 2 \implies b' = c' + 1) \wedge$$
$$(a' = 3 \implies b' = c' - 1)$$

We use $\Phi_i$ 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 $M_1 \equiv \{a = 0, b = -2, a' =$

$-2, b' = 0, c' = 0\}$. GETMBP$(\vec{y}, M_1, T)$ returns:

$$T_1 \equiv (a + b \neq 2) \wedge (a + b \neq 3) \qquad \phi_1 \equiv (a' = a + b) \wedge (b' = c') \tag{13}$$

In the second iteration, $\Phi_2 = \Phi_1 \wedge \neg T_1$, $M_2 \equiv \{a = 2, b = 1, a' = 3, b' = 0, c' = 1\}$, and GETMBP$(\vec{y}, M_2, T)$ returns:

$$T_2 \equiv (a + b \neq 1) \wedge (a + b \neq 2) \qquad \phi_2 \equiv (a' = b + a) \wedge (b' = c' - 1) \tag{14}$$

In the third iteration, $\Phi_3 = \Phi_2 \wedge \neg T_2$, $M_3 \equiv \{a = 2, b = 0, a' = 2, b' = 1, c' = 0\}$, and GETMBP$(\vec{y}, M_3, T)$ returns:

$$T_3 \equiv (a + b \neq 1) \wedge (a + b \neq 3) \qquad \phi_3 \equiv (a' = a + b) \wedge (b' = c' + 1) \tag{15}$$

Since $\Phi_4 = \Phi_3 \wedge \neg T_3$ is UNSAT, EA-VAL returns VALID and terminates. □

### 4.2 Extracting Skolem Relation

In the previous section, we have shown an algorithm AE-VAL that decides validity of $S(\vec{x}) \implies \exists \vec{y} \cdot T(\vec{x}, \vec{y})$. As a by-product, it constructs a set of MBPs $\{T_{\vec{y}_i}(\vec{x})\}$ for $T$ and the corresponding local Skolem relations $\{\phi_{\vec{y}_i}(\vec{x}, \vec{y})\}$. In this section, we show how this information can be turned into a Skolem relation $Sk_{\vec{y}}(\vec{x}, \vec{y})$.

Intuitively, $Sk_{\vec{y}}(\vec{x}, \vec{y})$ maps each model of $S$ to a corresponding model of $T$. However, the local Skolem relation $\phi_{\vec{y}_i}(\vec{x}, \vec{y})$ provides only a partial map (i.e., only for the subset $S(\vec{x}) \wedge T_{\vec{y}_i}(\vec{x})$ of $S$). Moreover, the local Skolem relations 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 local Skolem relation.

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 the partition. 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:

$$\begin{cases} S(\vec{x}) \wedge T_{\vec{y}_1}(\vec{x}) \implies I_1(\vec{x}) \\ S(\vec{x}) \wedge T_{\vec{y}_2}(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \implies I_2(\vec{x}) \\ \ldots \\ S(\vec{x}) \wedge T_{\vec{y}_n}(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \wedge \neg T_{\vec{y}_2}(\vec{x}) \wedge \ldots \wedge \neg T_{\vec{y}_{n-1}}(\vec{x}) \implies I_n(\vec{x}) \\ S(\vec{x}) \wedge I_1(\vec{x}) \wedge \neg T_{\vec{y}_1}(\vec{x}) \implies \bot \\ \ldots \\ S(\vec{x}) \wedge I_n(\vec{x}) \wedge \neg T_{\vec{y}_n}(\vec{x}) \implies \bot \end{cases} \tag{16}$$

Note that in (16), $S$ 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 Skolem relation $Sk_{vy}(\vec{x}, \vec{y})$ as follows:

$$Sk_{\vec{y}}(\vec{x}, \vec{y}) \equiv \begin{cases} \phi_{\vec{y}_1}(\vec{x}, \vec{y}) & \text{if } I_1(\vec{x}) \\ \cdots & \\ \phi_{\vec{y}_n}(\vec{x}, \vec{y}) & \text{else if } I_n(\vec{x}) \end{cases} \tag{17}$$

The following two theorems show the soundness and completeness of our Skolem relation $Sk_{\vec{y}}(\vec{x}, \vec{y})$. Soundness means that for chosen model $\vec{x}$, $Sk_{\vec{y}}(\vec{x}, \vec{y})$ satisfies the Def. 6. Completeness means that $Sk_{\vec{y}}(\vec{x}, \vec{y})$ is defined for all models of $\vec{x}$.

**Theorem 2 (Soundness of Skolem Relation).** *If the set $\{I_i(\vec{x})\}$ is a solution to* (16)*, and $Sk_{\vec{y}}(\vec{x}, \vec{y})$ is as in* (17) *then: $S(\vec{x}) \wedge Sk_{\vec{y}}(\vec{x}, \vec{y}) \implies T(\vec{x}, \vec{y})$.*

*Proof.* Simplifying (17), we need to prove that for every $1 \leq i \leq n$, $S(\vec{x}) \wedge I_i(\vec{x}) \wedge \phi_{\vec{y}_i}(\vec{x}, \vec{y}) \implies T(\vec{x}, \vec{y})$. It is enough to prove that for every $i$, $S(\vec{x}) \wedge I_i(\vec{x}) \implies T_{\vec{y}_i}(\vec{x})$, which is guaranteed by the last $n$ constraints of (16). □

**Theorem 3 (Completeness of Skolem Relation).** *If the set $\{I_i(\vec{x})\}$ is a solution to* (16) *then $S(\vec{x}) \implies \bigvee_i^n I_i(\vec{x})$.*

*Proof.* Follows immediately from the first $n$ constraints of (16). □

*Example 4.* A partitioning $I_1$, $I_2$, $I_3$ that determines a Skolem relation for Example 3 is: $I_1 \equiv (b \neq 1) \wedge (b \neq 0)$, $I_2 \equiv b \geq 1$, and $I_3 \equiv b = 0$. □

*Constructing a Minimal Skolem relation.* Any solution to (16) 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 (16) until no smaller solution can be found. We leave the problem of finding the minimum partitioning for future work.

## 5 Simulation-Abstraction-Refinement Loop

In this section, we present our algorithm SimAbs that iteratively constructs and strengthens a simulation relation. While so far we have assumed that a program is represented by a single transition relation, in practice, our algorithms operate on the Cut Point Graph (CPG) [12] representation. A CPG (or a Large Block Encoding (LBE) [4]) is a generalization of a Control Flow Graph (CFG) in which nodes (called *cutpoints*) correspond to loop heads and edges to longest loop free program fragments. In this section, we use CPGs informally and refer the reader to [12] for the formal definition.

The main loop of SimAbs is shown in Alg. 2. The input is two programs $S$ and $T$. The output is an abstraction $\alpha T$ of $T$ and a simulation relation $\rho_\alpha$ such

**Algorithm 2:** SimAbs$(S, T)$

**Input**: programs $S$ and $T$,
abstraction quality metric $\mathcal{Q} : \alpha \to \{\top, \bot\}$
**Output**: an abstraction $\alpha T$,
and a simulation relation $\rho_\alpha$, s.t. $S \preceq_{\rho_\alpha} \alpha T$

1 $\rho \leftarrow$ Synthesize$(S, T, \varnothing)$;
2 **while** $(\top)$ **do**
3 $\quad \alpha_{pre}T \leftarrow \alpha T$;
4 $\quad \alpha T, \rho_\alpha \leftarrow$ FindAbs$(S, T, \rho_\alpha)$;
5 $\quad$ **if** $(\alpha T \neq \mathbb{U})$ **then return** $\mathbb{U}, \varnothing$;
6 $\quad \alpha T, \rho_\alpha \leftarrow$ Extend$(S, \alpha T, \rho_\alpha)$;
7 $\quad$ **if** $(\mathcal{Q}(\alpha T) \vee (\alpha_{pre}T = \alpha T))$ **then return** $\alpha T, \rho_\alpha$;

**Algorithm 3:** FindAbs$(S, T, \rho)$

**Input**: programs $S$ and $T$, and a relation $\rho$
**Output**: an abstraction $\alpha T$,
and a simulation relation $\rho_\alpha$, s.t. $S \preceq_\alpha \alpha T$

1 **for each** $(u, v) \in E$ **do**
2 $\quad$ **if** $(\tau_S(u, v) \not\preceq_\rho \tau_T(u, v))$ **then**
3 $\quad\quad \alpha T \leftarrow$ Weaken$(T)$;
4 $\quad\quad$ **if** $(\alpha T \neq \mathbb{U})$ **then**
5 $\quad\quad\quad \rho_\alpha \leftarrow$ Synthesize$(S, \alpha T, \varnothing)$;
6 $\quad\quad\quad$ **return** FindAbs$(S, \alpha T, \rho_\alpha)$
7 $\quad\quad$ **else return** $\mathbb{U}, \varnothing$;
8 **return** $T, \rho$;

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

**Input**: $S$, abstraction $\alpha T$, simulation relation $\rho_\alpha$
**Output**: abstraction $\alpha^{ext}T$, simulation relation $\rho_\alpha^{ext}$

1 $\rho_\alpha^{ext} \leftarrow \rho_\alpha$; $\alpha^{ext}T \leftarrow \alpha T$;
2 $WL \leftarrow E$ ;
3 **while** $(WL \neq \emptyset)$ **do**
4 $\quad (u, v) \leftarrow$ GetWtoSmallestEdge$(WL)$;
5 $\quad WL \leftarrow WL \setminus \{(u, v)\}$;
6 $\quad$ **if** $(\tau_S(u, v) \preceq_{\rho_\alpha^{ext}} \tau_{\alpha^{ext}(T)}(u, v))$ **then**
7 $\quad\quad Sk \leftarrow$ ExtractSkolem$(u, v, \rho_\alpha^{ext})$;
8 $\quad\quad \alpha^{ext}T \leftarrow$ Strengthen$(\alpha T, Sk)$;
9 $\quad\quad \rho_\alpha^{ext} \leftarrow$ Synthesize$(S, \alpha T, Sk)$;
10 $\quad\quad WL \leftarrow WL \cup \{(v, x) \in E \mid x \in \text{CP}\} \cup \{(y, v) \in E \mid y \in \text{CP}\}$;
11 $\quad$ **else return** $\alpha T, \rho_\alpha$;
12 **return** $\alpha^{ext}T, \rho_\alpha^{ext}$

that $S \preceq_{\rho_\alpha} T$, if such an 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 begins by guessing an initial relation $\rho$ using Synthesize. The initial guess can be an arbitrary relation between the CPGs of $S$ and $T$. In our implementation, for every cutpoint, we take $\rho$ to be the identity relation between the live variables of $S$ and $T$ at that cutpoint, that have identical names. Then, SimAbs uses FindAbs to check whether there exists an abstraction of $\rho$ (including $\rho$ itself) that is a simulation relation between $S$ and a corresponding abstraction $\alpha T$ of $T$. If this succeeds, the method Extend is used to refine the abstraction $\alpha T$ and synthesize a new simulation relation $\rho_\alpha$. This process continues until the abstraction is satisfied by some quality metric $\mathcal{Q}$ (line 7) (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 an abstraction cannot be constructed (we use a shortcut $\mathbb{U}$ for this), SimAbs terminates with a negative result (line 5).

FindAbs (shown in Alg. 3) is given $S$, $T$ and $\rho$. It traverses the set of CPG edges $E$, common to $S$ and $T$ (line 1) in the Weak Topological Ordering (WTO) [5] (in which inner loops are traversed before outer loops). For each edge $(u, v) \in E$, FindAbs checks (on line 2) whether $\rho$ 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, the $\rho$ is a simulation relation between $S$ and $T$, and it is returned to the user. Otherwise, FindAbs chooses an abstraction $\alpha T$ of $T$ using the method Weaken, synthesizes a new $\rho_\alpha$ and repeats the check for $S$, $\alpha T$, and $\rho_\alpha$. Weaken introduces non-determinism to the interpretation of control-flow edges of $T$. In our implementation, the most successful implementation of Weaken is the existential abstraction of all state-variables that are live at the source and destination 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) is given $S$, $\alpha T$, and $\rho_\alpha$ and constructs a refinement $\rho_\alpha^{ext}$ of simulation relation $\rho_\alpha$ and corresponding refinement $\alpha^{ext}T$ of $\alpha T$. Extend maintains a work-list $WL$ of control-flow edges to be processed. Initially, $WL$ is populated with all the CPG edges $E$ (line 2). In each iteration, Extend strengthens $\alpha T$ and synthesizes a new $\rho_\alpha$ by strengthening the old $\rho_\alpha$ by a Skolem relation $Sk$. $Sk$ is produced from the proof of the simulation between fragments $\tau_S(u, v)$ and $\tau_{\alpha T}(u, v)$ of the smallest element $(u, v)$ of $E$ according to the WTO (line 6). Finally, Extend updates $WL$ (line 10) with the edges that share the next-state variables that have appeared in $Sk$ and iterates until $WL$ is empty. If in some iteration, a strengthening is impossible, Extend returns the last successful values for $\rho_\alpha^{ext}$ and $\alpha^{ext}T$.

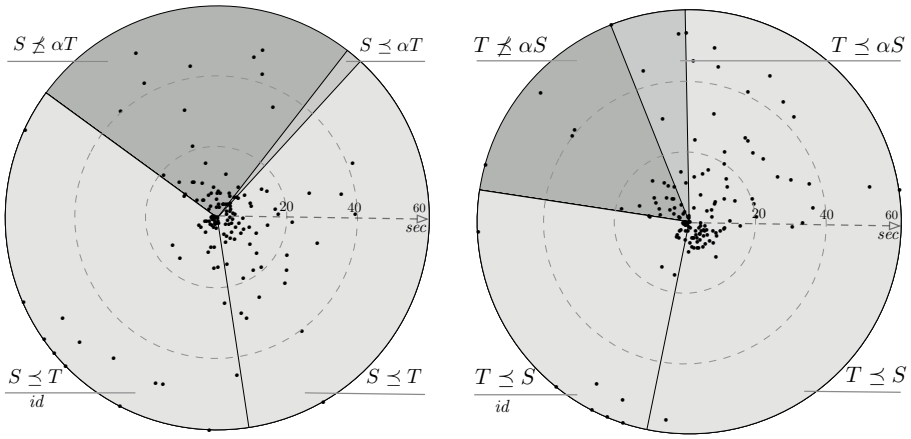
Recall programs `P` and `Q` from Fig. 2a and Fig. 2b. In Examples 1 and 2, we proved that `P` $\preceq$ `Q`. In the following , we show how SymAbs automates this.

*Example 5.* First, SymAbs is given `P` and `Q`; Synthesize guesses $\mathcal{C}$ and $\mathcal{N}$ as in (4). Then FindAbs disproves that $\mathcal{C}$ and $\mathcal{N}$ is a simulation relation, but chooses an implicit abstraction of `Q` (equivalent to `AQ` on Fig. 2c), and constructs $\mathcal{C}_\alpha$ and $\mathcal{N}_\alpha$, as in (7), for which the simulation-relation-formula (8) for `P` and `AQ` is valid. Finally, Extend extracts Skolem relation and uses it to create $\mathcal{C}_\alpha^{ext}$ and $\mathcal{N}_\alpha^{ext}$, as in (10) and confirms that it is a simulation relation for `P` and `Q`. □

## 6 Evaluation

We have implemented SimAbs in the UFO framework, and evaluated it on the Software Verification Competition (SVCOMP'14) benchmarks and `instcombine` and `simpifycfg` optimizations of LLVM. The `instcombine` performs local arithmetic optimizations (e.g., replacing `a = 1 - 1` by `a = 0`). The `simpifycfg` performs dead code elimination and basic block merging (e.g., replacing `if (true) {a++;} else {a--;}` by `a++`). The combination of these optimizations provides more aggressive optimizations.

For each source benchmark ($S$) (300 - 5000 lines of source code), we created a target optimization ($T$) by two applications of `(instcombine + simpifycfg)`. We applied SimAbs to discover two simulations: $S \preceq T$ and $T \preceq S$. Out of all benchmarks, we chose 157 for which SimAbs terminates within 5 minutes. We




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

Fig. 4: Pie chart and running times in spherical coordinate system.

present the results in two diagrams in Fig. 4. Full results are available at `http://www.inf.usi.ch/phd/fedyukovich/simabs.pdf`[4].

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. 4a represents a proportion of four main classes of SimAbs results: whether (a-b) $T$ simulates $S$ (in (a), via identity relation), (c) $T$ does not simulates $S$, but some abstraction $\alpha T$ does, (d) $T$ does not simulate $S$ and we did not find an abstraction $\alpha T$ that does. Each dot represents a runtime of SimAbs on a single benchmark. It is placed in one of the segments (a)–(d) 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 20s is placed in quadrant (a) in a distance 20 from the center. Closer to the center means faster. Runs that took longer than 60s are placed on the boundary. Fig. 4b is structured similarly, but with inverse order of $S$ and $T$.

The experiment confirms our intuition that the original program more often simulates the optimal one than vice versa. According to Fig. 4, in 115 cases $S \preceq \alpha T$, and in 122 cases $T \preceq \alpha S$; in 40 cases $S \not\preceq \alpha T$, and in 27 cases $T \not\preceq \alpha S$.

The experiment confirms that the use of our novel simulation-abstraction-refinement loop by SimAbs is necessary in majority of cases. There is a relatively small subset of benchmarks (59 of $S \preceq T$ and 39 of $T \preceq S$), in which the identity relation, guessed at the first iteration of SimAbs is already a simulation relation (but still, in many cases it took more than a minute to establish that), while in the rest of the cases, SimAbs goes into the simulation-abstraction-refinement loop and successfully terminates (faster than a minute).

[4] For convenience, we have included the table of results in an appendix.

## 7 Related Work

The algebraic notion of simulation relation between programs dates back to Milner [18]. The approach indirectly refers to simulation relation between $S$ and the abstraction of $T$, by introducing *weak simulation*, and to simulation relation between $S$ and $T$ as *strong simulation*. While being purely theoretic, this work does not consider a practical application of using weak simulations in order to construct strong simulation.

Translation Validation for optimizing compiler [21] checks simulation relation between a program and its GCC-optimization. As a secondary result, that work proposes a simple heuristic to construct simulation relation, restricted to specific optimizations. Checking is done on the level of control-flow graphs and takes into account all program variables. In contrast, our SimAbs algorithm is able to find simulation relation independently on an optimizer.

Classical approach to check that a relation is a simulation relation is by *game-theoretic* approach, in which the state space of the source and the target is traversed by the evader and a pursuer solvers. For instance, [13] applies it to prove simulation relation between infinite graphs. In our setting, this result can used to extend SimAbs to deal with programs with different CPGs.

The need of eliminating quantifiers by a method AE-VAL makes our approach similar to template-based synthesis [24, 3, 2, 17]. The goal of the approach is to synthesize an arbitrary program that fulfills a given specification represented by a template. While instantiating existential quantifiers, synthesis is filling placeholders in the predefined template formula. While discovering a Skolem relation on the top of valid simulation-relation-checking formula, we also perform a synthesis, but do not require any template for it.

Apart of discovering simulation between programs, there exist another ways to prove their equivalence. For example, rather practical solution to check equivalence between a Verilog circuit and C program was established in [7]. It is based on translation of both programs into quantifier-free propositional formula, satisfiable iff the circuit and the program disagree.

## References


1. A. Albarghouthi, A. Gurfinkel, and M. Chechik. UFO: A Framework for Abstraction- and Interpolation-Based Software Verification. In *CAV*, 2012.
2. R. Alur, R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *FMCAD*, pages 1–17, 2013.
3. T. A. Beyene, C. Popeea, and A. Rybalchenko. Solving existentially quantified horn clauses. In *CAV*, pages 869–882, 2013.
4. D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, and R. Sebastiani. Software Model Checking via Large-Block Encoding. In *FMCAD*, pages 25–32, 2009.
5. F. A. Bourdoncle. Efficient Chaotic Iteration Strategies with Widenings. In *Proc. of FMPA'93*, LNCS, pages 128–141, 1993.
6. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

7. E. M. Clarke, D. Kroening, and K. Yorav. Behavioral consistency of c and verilog programs using bounded model checking. In *DAC*, pages 368–371, 2003.
8. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, 2008.
9. G. Fedyukovich, A. Gurfinkel, and N. Sharygina. Incremental verification of compiler optimizations. In *NFM*, volume 8430 of *LNCS*. Springer, 2014.
10. G. Fedyukovich, O. Sery, and N. Sharygina. eVolCheck: Incremental Upgrade Checker for C. In *Proc. of TACAS'13*, volume 7795 of *LNCS*, pages 292–307, 2013.
11. B. Godlin and O. Strichman. Regression verification. In *DAC*, 2009.
12. A. Gurfinkel, S. Chaki, and S. Sapra. Efficient Predicate Abstraction of Program Summaries. In *NASA Formal Methods*, pages 131–145, 2011.
13. M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462, 1995.
14. M. Kawaguchi, S. K. Lahiri, and H. Rebelo. Conditional equivalence. Technical Report MSR-TR-2010-119, Microsoft Research, 2010.
15. A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-Based Model Checking for Recursive Programs. In *CAV*, pages 17–34, 2014.
16. R. Loos and V. Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.
17. R. Madhavan and V. Kuncak. Symbolic resource bound inference for functional programs. In *Computer Aided Verification (CAV)*, 2014.
18. R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.
19. K. S. Namjoshi. Lifting Temporal Proofs through Abstractions. In *VMCAI*, volume 2575 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2003.
20. K. S. Namjoshi and L. D. Zuck. Witnessing program transformations. In *SAS*, 2013.
21. G. C. Necula. Translation validation for an optimizing compiler. In *PLDI*, 2000.
22. S. Person, M. B. Dwyer, S. G. Elbaum, and C. S. Pasareanu. Differential symbolic execution. In *Foundations of SW Engineering (FSE '08)*, pages 226–237, 2008.
23. A.-D. Phan, N. Bjørner, and D. Monniaux. Anatomy of alternating quantifier satisfiability (work in progress). In *SMT*, pages 120–130, 2012.
24. S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *POPL*, pages 313–326, 2010.

## A The proof of Theorem 1.

**Theorem 1.** *Let $S(\vec{s}, \vec{x}, \vec{s'})$ and $T(\vec{t}, \vec{y}, \vec{t'})$ be two programs such that $S \preceq T$, and $U \subseteq \vec{t}$. Let $\rho_\alpha$ be a simulation relation such that $S \preceq_{\rho_\alpha} \alpha_U^{\exists}(T)$. Then, there exists a relation $Sk(\vec{s}, U)$ such that (a) $\rho_\alpha \wedge Sk$ is a simulation relation between $S$ and $T$ and (b) $Sk$ is a Skolem relation for $U$ in* (6).

*Proof.* Let $\vec{t} \equiv \vec{t_1} \cup U$ and $\vec{t'} \equiv \vec{t'_1} \cup U'$. Since $S \preceq T$, there exist relation $\rho$ such that $S \preceq_\rho T$ and $\rho \implies \rho_\alpha$. Let $Sk$ be a relation over $\vec{s}$, $\vec{s'}$, $U$ and $U'$, such that

$$\rho(\vec{s}, \vec{t}) \equiv \rho_\alpha(\vec{s}, \vec{t_1}) \wedge Sk(\vec{s}, U) \qquad \rho(\vec{s'}, \vec{t'}) \equiv \rho_\alpha(\vec{s'}, \vec{t'_1}) \wedge Sk(\vec{s'}, U') \tag{18}$$

From (2) and (18), we get

$$\rho_\alpha(\vec{s}, \vec{t_1}) \wedge Sk(\vec{s}, U) \wedge S(\vec{s}, \vec{s'}) \implies \exists \vec{t'_1}, U', \vec{y} \cdot T(\vec{t_1}, U, \vec{y}, \vec{t'_1}, U') \wedge \rho_\alpha(\vec{s'}, \vec{t'_1}) \wedge Sk(\vec{s'}, U') \tag{19}$$

In (19), $Sk$ is witnessing a Skolem relation for $U$. By Def. 6, we get:

$$\exists U \cdot \rho_\alpha(\vec{s}, \vec{t_1}) \wedge S(\vec{s}, \vec{s'}) \implies \exists \vec{t'_1}, U', \vec{y} \cdot T(\vec{t_1}, U, \vec{y}, \vec{t'_1}, U') \wedge \rho_\alpha(\vec{s'}, \vec{t'_1}) \wedge Sk(\vec{s'}, U') \tag{20}$$

Since (20) is valid then a weaker formula (21) is also valid.

$$\exists U \cdot \rho_\alpha(\vec{s}, \vec{t_1}) \wedge S(\vec{s}, \vec{s'}) \implies \exists \vec{t'_1}, U, U', \vec{y} \cdot T(\vec{t_1}, U, \vec{y}, \vec{t'_1}, U') \wedge \rho_\alpha(\vec{s'}, \vec{t'_1}) \tag{21}$$

Notably, (21) is a simulation-checking-formula (6) for $S$ and $\alpha_U^{\exists}(T)$. It means, the chosen $Sk$ is also a Skolem relation for (6). □

## B Concrete data

Tables 1, 2, 3 gather statistics for all 3 cases ($S \preceq T$, $S \preceq \alpha T$, and $S \not\preceq \alpha T$) respectively.

| name | #Vars | #Nondet Vars | | | Time | cutpoints |
|---|---|---|---|---|---|---|
| | | beg | abs | end | | |
| locks/test_locks_14_unsafe.o3 | 5 | 0 | 0 | 0 | 4.34 | 0/51 |
| locks/test_locks_15_unsafe.o3 | 5 | 0 | 0 | 0 | 4.32 | 0/51 |
| locks/test_locks_15_unsafe.o0 | 4 | 0 | 0 | 0 | 0.18 | 0/5 |
| locks/test_locks_14_unsafe.o0 | 4 | 0 | 0 | 0 | 0.18 | 0/5 |
| ntdrivers/diskperf_unsafe.1.cil.o3 | 5 | 0 | 0 | 0 | 1.26 | 0/5 |
| ntdrivers/diskperf_unsafe.1.cil.o0 | 36 | 0 | 0 | 0 | 152.23 | 0/11 |
| ntdrivers/kbfiltr_unsafe.1.cil.o3 | 7 | 0 | 0 | 0 | 31.8 | 0/7 |
| product-lines/minepump_spec3_product20_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.84 | 0/3 |
| product-lines/minepump_spec3_product18_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.61 | 0/3 |
| product-lines/minepump_spec3_product29_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.58 | 0/3 |
| product-lines/minepump_spec3_product05_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.58 | 0/3 |
| product-lines/minepump_spec3_product13_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.61 | 0/3 |
| product-lines/minepump_spec3_product24_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.88 | 0/3 |
| product-lines/minepump_spec3_product26_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.56 | 0/3 |
| product-lines/minepump_spec3_product30_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.57 | 0/3 |
| product-lines/minepump_spec3_product06_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.6 | 0/3 |
| product-lines/minepump_spec3_product25_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.6 | 0/3 |
| product-lines/minepump_spec3_product10_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.59 | 0/3 |
| product-lines/minepump_spec3_product17_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.6 | 0/3 |
| product-lines/minepump_spec3_product32_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.92 | 0/3 |
| product-lines/minepump_spec3_product08_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.87 | 0/3 |
| product-lines/minepump_spec3_product18_unsafe.cil.o0 | 4 | 0 | 0 | 0 | 0.59 | 0/3 |
| product-lines/minepump_spec3_product11_unsafe.cil.o3 | 4 | 0 | 0 | 0 | 0.85 | 0/3 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-usb-storage-usb-storage.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 17 | 0 | 0 | 0 | 4.22 | 0/8 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-vp7045.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 64 | 0 | 0 | 0 | 4.06 | 0/56 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-scsi-libfc-libfc.ko-ldv_main5_sequence_infinite_withcheck_stateful.cil.out.o0 | 20 | 0 | 0 | 0 | 20.73 | 0/21 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-mfd-tps6105x.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 17 | 0 | 0 | 0 | 3.26 | 0/12 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-staging-keucr-keucr.ko-ldv_main1_sequence_infinite_withcheck_stateful.cil.out.o0 | 12 | 0 | 0 | 0 | 5.48 | 0/5 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-isdn-capi-kernelcapi.ko-ldv_main3_sequence_infinite_withcheck_stateful.cil.out.o0 | 15 | 0 | 0 | 0 | 5.91 | 0/5 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-vp7045.ko-ldv_main1_sequence_infinite_withcheck_stateful.cil.out.o3 | 231 | 0 | 0 | 0 | 111.13 | 0/65 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-mfd-tps6105x.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o3 | 6 | 0 | 0 | 0 | 1.34 | 0/8 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-isdn-capi-kernelcapi.ko-ldv_main3_sequence_infinite_withcheck_stateful.cil.out.o3 | 16 | 0 | 0 | 0 | 5.46 | 0/5 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-vp7045.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o3 | 70 | 0 | 0 | 0 | 19.85 | 0/32 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-vp7045.ko-ldv_main1_sequence_infinite_withcheck_stateful.cil.out.o0 | 162 | 0 | 0 | 0 | 14.82 | 0/117 |
| systemc/transmitter.02_unsafe.cil.o3 | 28 | 0 | 0 | 0 | 12.49 | 0/7 |
| systemc/token_ring.02_safe.cil.o3 | 37 | 0 | 0 | 0 | 23.86 | 0/7 |
| systemc/pc_sfifo_1_safe.cil.o3 | 56 | 0 | 0 | 0 | 26.79 | 0/11 |
| systemc/pc_sfifo_2_safe.cil.o3 | 105 | 0 | 0 | 0 | 34.46 | 0/13 |
| systemc/transmitter.01_unsafe.cil.o3 | 22 | 0 | 0 | 0 | 1.93 | 0/12 |
| systemc/token_ring.01_unsafe.cil.o0 | 56 | 0 | 0 | 0 | 80.07 | 0/7 |
| systemc/transmitter.01_unsafe.cil.o0 | 39 | 0 | 0 | 0 | 22.3 | 0/7 |
| systemc/hist_cell_safe.cil.o0 | 35 | 0 | 0 | 0 | 105.95 | 0/7 |
| systemc/pc_sfifo_1_unsafe.cil.o3 | 56 | 0 | 0 | 0 | 16.19 | 0/11 |
| systemc/token_ring.02_unsafe.cil.o3 | 37 | 0 | 0 | 0 | 25.85 | 0/7 |
| systemc/pc_sfifo_2_unsafe.cil.o3 | 105 | 0 | 0 | 0 | 32.4 | 0/14 |
| systemc/transmitter.03_unsafe.cil.o3 | 38 | 0 | 0 | 0 | 129.72 | 0/7 |
| systemc/kundul_unsafe.cil.o3 | 57 | 0 | 0 | 0 | 7.62 | 0/11 |
| systemc/token_ring.01_safe.cil.o3 | 35 | 0 | 0 | 0 | 3.74 | 0/12 |
| systemc/token_ring.01_unsafe.cil.o3 | 27 | 0 | 0 | 0 | 1.8 | 0/7 |
| systemc/token_ring.01_safe.cil.o0 | 56 | 0 | 0 | 0 | 88.99 | 0/7 |

Table 1: Full cycle of SimAbs, $S \preceq T$

| name | #Vars | #Nondet Vars | | | Time | cutpoints |
|---|---|---|---|---|---|---|
| | | beg | abs | end | | |
| ntdrivers/floppy_safe.i.cil.o3 | 7 | 1 | 1 | 0 | 1.26 | 0/5 |
| ntdrivers/kbfiltr_unsafe.1.cil.o0 | 10 | 1 | 1 | 0 | 57.22 | 0/7 |
| ntdrivers/floppy_unsafe.1.cil.o3 | 3 | 1 | 1 | 0 | 0.14 | 0/5 |
| product-lines/minepump_spec5_product47_safe.cil.o3 | 8 | 0 | 4 | 0 | 14.33 | 1/3 |
| product-lines/minepump_spec1_product49_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 5.81 | 1/3 |
| product-lines/minepump_spec3_product36_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 29.81 | 1/3 |
| product-lines/minepump_spec3_product48_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 32.29 | 1/3 |
| product-lines/minepump_spec4_product52_safe.cil.o3 | 6 | 0 | 3 | 0 | 4.64 | 1/3 |
| product-lines/minepump_spec5_product53_safe.cil.o0 | 6 | 0 | 3 | 0 | 6.89 | 1/3 |
| product-lines/minepump_spec2_product58_safe.cil.o0 | 8 | 0 | 4 | 0 | 10.09 | 1/3 |
| product-lines/minepump_spec1_product45_safe.cil.o0 | 6 | 0 | 3 | 0 | 4.63 | 1/3 |
| product-lines/minepump_spec4_product38_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 4.12 | 1/3 |
| product-lines/minepump_spec1_product39_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 18.13 | 1/3 |
| product-lines/minepump_spec4_product38_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 6.66 | 1/3 |
| product-lines/minepump_spec4_product40_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 36.17 | 1/3 |
| product-lines/minepump_spec1_product58_safe.cil.o0 | 6 | 0 | 3 | 0 | 6.89 | 1/3 |
| product-lines/minepump_spec5_product46_safe.cil.o3 | 6 | 0 | 3 | 0 | 5.12 | 1/3 |
| product-lines/minepump_spec1_product50_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 3.97 | 1/3 |
| product-lines/minepump_spec4_product35_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 3.69 | 1/3 |
| product-lines/minepump_spec2_product43_unsafe.cil.o0 | 10 | 0 | 5 | 0 | 25.8 | 1/3 |
| product-lines/minepump_spec2_product59_safe.cil.o0 | 10 | 0 | 5 | 0 | 43.92 | 1/3 |
| product-lines/minepump_spec5_product64_safe.cil.o0 | 8 | 0 | 4 | 0 | 62.07 | 1/3 |
| product-lines/minepump_spec4_product56_safe.cil.o0 | 8 | 0 | 4 | 0 | 49.44 | 1/3 |
| product-lines/minepump_spec2_product38_safe.cil.o0 | 8 | 0 | 4 | 0 | 5.55 | 1/3 |
| product-lines/minepump_spec1_product44_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 16.69 | 1/3 |
| product-lines/minepump_spec1_product63_safe.cil.o0 | 8 | 0 | 4 | 0 | 34.62 | 1/3 |
| product-lines/minepump_spec1_product54_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 6.8 | 1/3 |
| product-lines/minepump_spec2_product61_safe.cil.o0 | 10 | 0 | 5 | 0 | 38.64 | 1/3 |
| product-lines/minepump_spec1_product48_safe.cil.o3 | 8 | 0 | 4 | 0 | 17.85 | 1/3 |
| product-lines/minepump_spec5_product52_safe.cil.o3 | 6 | 0 | 3 | 0 | 5.39 | 1/3 |
| product-lines/minepump_spec5_product52_safe.cil.o0 | 8 | 0 | 4 | 0 | 28.23 | 1/3 |
| product-lines/minepump_spec1_product36_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 12.11 | 1/3 |
| product-lines/minepump_spec2_product39_safe.cil.o3 | 10 | 0 | 5 | 0 | 10.9 | 1/3 |
| product-lines/minepump_spec5_product40_safe.cil.o0 | 8 | 0 | 4 | 0 | 26.17 | 1/3 |
| product-lines/minepump_spec5_product35_safe.cil.o3 | 6 | 0 | 3 | 0 | 2.72 | 1/3 |
| product-lines/minepump_spec3_product34_safe.cil.o0 | 6 | 0 | 3 | 0 | 6.18 | 1/3 |
| product-lines/minepump_spec4_product61_safe.cil.o0 | 6 | 0 | 3 | 0 | 10.57 | 1/3 |
| product-lines/minepump_spec4_product54_safe.cil.o3 | 6 | 0 | 3 | 0 | 3.82 | 1/3 |
| product-lines/minepump_spec2_product50_safe.cil.o3 | 8 | 0 | 4 | 0 | 6.27 | 1/3 |
| product-lines/minepump_spec1_product42_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 3.64 | 1/3 |
| product-lines/minepump_spec2_product48_safe.cil.o3 | 10 | 0 | 5 | 0 | 24.22 | 1/3 |
| product-lines/minepump_spec2_product45_safe.cil.o0 | 8 | 0 | 4 | 0 | 5.4 | 1/3 |
| product-lines/minepump_spec4_product61_safe.cil.o3 | 6 | 0 | 3 | 0 | 5.08 | 1/3 |
| product-lines/minepump_spec1_product57_safe.cil.o0 | 6 | 0 | 3 | 0 | 6.72 | 1/3 |
| product-lines/minepump_spec5_product58_safe.cil.o0 | 6 | 0 | 3 | 0 | 8.19 | 1/3 |
| product-lines/minepump_spec5_product35_safe.cil.o0 | 8 | 0 | 4 | 0 | 12.07 | 1/3 |
| product-lines/minepump_spec2_product45_safe.cil.o3 | 8 | 0 | 4 | 0 | 6.72 | 1/3 |
| product-lines/minepump_spec1_product33_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 3.25 | 1/3 |
| product-lines/minepump_spec3_product07_unsafe.cil.o0 | 4 | 2 | 2 | 0 | 1.16 | 0/3 |
| product-lines/minepump_spec2_product53_safe.cil.o0 | 8 | 0 | 4 | 0 | 8.53 | 1/3 |
| product-lines/minepump_spec4_product55_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 27.13 | 1/3 |
| product-lines/minepump_spec3_product45_safe.cil.o3 | 6 | 0 | 3 | 0 | 5.54 | 1/3 |
| product-lines/minepump_spec1_product38_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 2.77 | 1/3 |
| product-lines/minepump_spec1_product58_safe.cil.o3 | 6 | 0 | 3 | 0 | 4.22 | 1/3 |
| product-lines/minepump_spec3_product39_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 14.71 | 1/3 |
| product-lines/minepump_spec4_product53_safe.cil.o3 | 6 | 0 | 3 | 0 | 3.93 | 1/3 |
| product-lines/minepump_spec4_product46_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 7.39 | 1/3 |
| product-lines/minepump_spec1_product60_safe.cil.o0 | 8 | 0 | 4 | 0 | 35.22 | 1/3 |
| product-lines/minepump_spec1_product42_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 3.84 | 1/3 |
| product-lines/minepump_spec1_product43_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 17.79 | 1/3 |
| product-lines/minepump_spec3_product34_safe.cil.o3 | 6 | 0 | 3 | 0 | 5.65 | 1/3 |
| product-lines/minepump_spec4_product43_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 16.24 | 1/3 |
| product-lines/minepump_spec1_product50_unsafe.cil.o0 | 6 | 0 | 3 | 0 | 6.42 | 1/3 |
| product-lines/minepump_spec3_product40_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 46.76 | 1/3 |
| product-lines/minepump_spec4_product60_safe.cil.o0 | 8 | 0 | 4 | 0 | 30.59 | 1/3 |
| product-lines/minepump_spec5_product56_safe.cil.o3 | 8 | 0 | 4 | 0 | 30.91 | 1/3 |
| product-lines/minepump_spec5_product38_safe.cil.o3 | 6 | 0 | 3 | 0 | 2.88 | 1/3 |
| product-lines/minepump_spec1_product44_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 17.13 | 1/3 |
| product-lines/minepump_spec3_product47_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 43.21 | 1/3 |
| product-lines/minepump_spec4_product40_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 23.21 | 1/3 |
| product-lines/minepump_spec5_product62_safe.cil.o3 | 6 | 0 | 3 | 0 | 7.35 | 1/3 |
| product-lines/minepump_spec5_product64_safe.cil.o3 | 8 | 0 | 4 | 0 | 41.83 | 1/3 |
| product-lines/minepump_spec3_product38_safe.cil.o0 | 6 | 0 | 3 | 0 | 4.19 | 1/3 |
| product-lines/minepump_spec3_product45_safe.cil.o0 | 6 | 0 | 3 | 0 | 8.95 | 1/3 |
| product-lines/minepump_spec2_product63_safe.cil.o0 | 10 | 0 | 5 | 0 | 47.37 | 1/3 |
| product-lines/minepump_spec3_product46_safe.cil.o0 | 6 | 0 | 3 | 0 | 8.88 | 1/3 |
| product-lines/minepump_spec1_product37_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 2.88 | 1/3 |
| product-lines/minepump_spec2_product57_safe.cil.o3 | 8 | 0 | 4 | 0 | 7.01 | 1/3 |
| product-lines/minepump_spec1_product36_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 14.3 | 1/3 |
| product-lines/minepump_spec2_product45_safe.cil.o3 | 8 | 0 | 4 | 0 | 4.14 | 1/3 |
| product-lines/minepump_spec1_product35_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 11.73 | 1/3 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-usb-storage-usb-storage.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o3 | 30 | 1 | 1 | 0 | 8.23 | 0/8 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-usb-image-microtek.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o3 | 25 | 1 | 21 | 14 | 1.14 | 2/9 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-usb-image-microtek.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o0 | 18 | 0 | 13 | 6 | 1.72 | 2/9 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-az6027.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o3 | 19 | 0 | 15 | 12 | 15.23 | 1/5 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-pctv452e.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o3 | 85 | 0 | 66 | 62 | 49.18 | 1/15 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-pctv452e.ko-ldv_main0_sequence_infinite_witchcheck_stateful.cil.out.o0 | 39 | 0 | 21 | 19 | 33.73 | 1/15 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-staging-keucr-keucr.ko-ldv_main1_sequence_infinite_witchcheck_stateful.cil.out.o3 | 25 | 1 | 1 | 0 | 6.03 | 0/5 |
| systemc/pc_sfifo_2_safe.cil.o0 | 70 | 6 | 6 | 3 | 59.9 | 0/14 |
| systemc/pc_sfifo_2_unsafe.cil.o0 | 70 | 6 | 6 | 3 | 59.14 | 0/15 |
| systemc/kundul_unsafe.cil.o0 | 40 | 10 | 10 | 4 | 13.14 | 0/7 |

Table 2: Full cycle of SimAbs, $S \preceq \alpha T$

| name | #Vars | #Nondet Vars beg | abs | end | Time | cutpoints |
|---|---|---|---|---|---|---|
| ntdrivers/diskperf_safe.i.cil.o3 | 5 | 0 | 0 | 0 | 1.41 | 1/5 |
| product-lines/minepump_spec3_product50_safe.cil.o0 | 6 | 0 | 3 | 0 | 9.52 | 2/3 |
| product-lines/minepump_spec2_product61_safe.cil.o3 | 8 | 0 | 4 | 0 | 5.42 | 2/3 |
| product-lines/minepump_spec1_product54_unsafe.cil.o3 | 6 | 0 | 3 | 0 | 4.38 | 2/3 |
| product-lines/minepump_spec3_product52_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 55.3 | 2/3 |
| product-lines/minepump_spec3_product51_unsafe.cil.o0 | 8 | 0 | 4 | 0 | 39.18 | 2/3 |
| product-lines/minepump_spec4_product50_safe.cil.o0 | 6 | 0 | 3 | 0 | 3.27 | 1/3 |
| product-lines/minepump_spec3_product54_safe.cil.o0 | 6 | 0 | 3 | 0 | 11.96 | 2/3 |
| product-lines/minepump_spec3_product57_safe.cil.o0 | 6 | 0 | 3 | 0 | 10.95 | 2/3 |
| product-lines/minepump_spec3_product49_safe.cil.o3 | 6 | 0 | 3 | 0 | 6.41 | 2/3 |
| product-lines/minepump_spec3_product50_safe.cil.o3 | 6 | 0 | 3 | 0 | 6.48 | 2/3 |
| product-lines/minepump_spec3_product60_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 39.74 | 2/3 |
| product-lines/minepump_spec4_product57_safe.cil.o3 | 6 | 0 | 3 | 0 | 3.12 | 1/3 |
| product-lines/minepump_spec1_product63_safe.cil.o3 | 8 | 0 | 4 | 0 | 12.43 | 2/3 |
| product-lines/minepump_spec3_product62_safe.cil.o0 | 6 | 0 | 3 | 0 | 13.54 | 2/3 |
| product-lines/minepump_spec2_product54_safe.cil.o3 | 8 | 0 | 4 | 0 | 5.42 | 2/3 |
| product-lines/minepump_spec1_product55_unsafe.cil.o3 | 8 | 0 | 4 | 0 | 11.84 | 2/3 |
| product-lines/minepump_spec3_product58_safe.cil.o3 | 6 | 0 | 3 | 0 | 7.88 | 2/3 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-mtd-chips-cfi_cmdset_0001.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o3 | 236 | 0 | 146 | 145 | 21.11 | 1/121 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-input-mouse-synaptics_usb.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o3 | 22 | 1 | 18 | 10 | 8.51 | 3/10 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-az6027.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 43 | 0 | 10 | 7 | 2.97 | 2/27 |
| ldv-linux-3.4/32_7_cilled_unsafe_const_ok_linux-32_1-drivers-input-mouse-synaptics_usb.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 18 | 0 | 13 | 4 | 9.34 | 3/10 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-az6007.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o0 | 87 | 0 | 10 | 10 | 15.5 | 1/84 |
| ldv-linux-3.4/32_1_cilled_safe_ok_nondet_linux-3.4-32_1-drivers-media-dvb-dvb-usb-dvb-usb-az6007.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.o3 | 47 | 0 | 10 | 10 | 64.15 | 1/13 |
| systemc/pc_sfifo_1_safe.cil.o0 | 43 | 3 | 20 | 1 | 4.9 | 6/12 |
| systemc/pc_sfifo_1_unsafe.cil.o0 | 43 | 3 | 20 | 2 | 3.14 | 7/13 |

Table 3: Full cycle of SimAbs, $S \not\preceq \alpha T$