Università della Svizzera italiana

Faculty of Informatics

Automated Discovery of Simulation Between Programs

Grigory Fedyukovich, Arie Gurfinkel, and Natasha Sharygina

LPAR, Fiji, 25 Nov 2015

What is this talk about

- Searching for total simulation relations
  - *for each behavior of one program (called source)*

    there should exist a behavior of another program (called target)

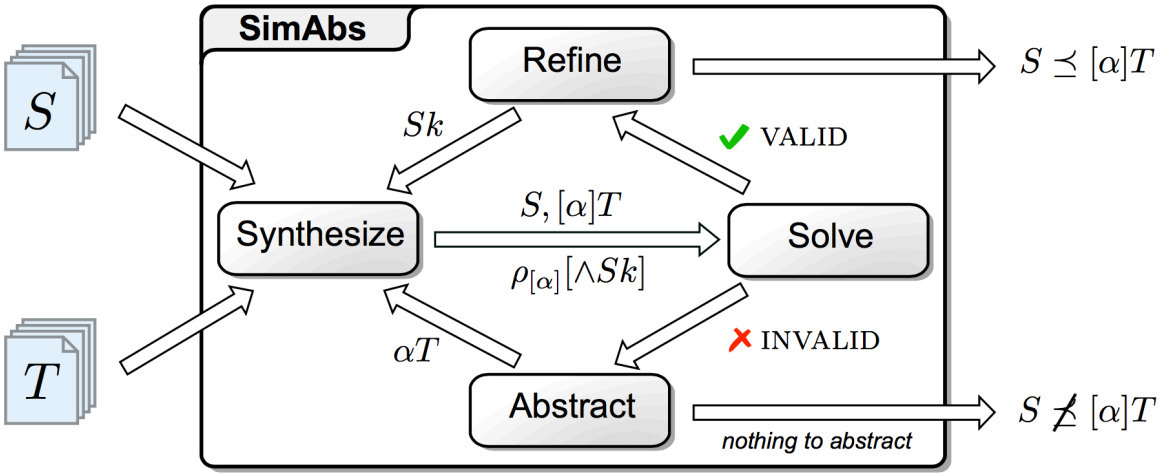
    matched by some simulation relation

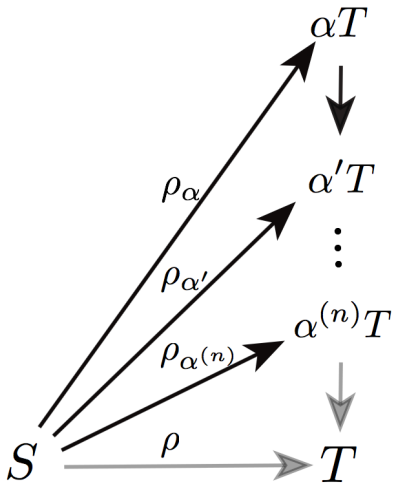
- Synthesizing abstract simulation relations
  - *if the target does not simulate the source*

    search for an abstraction of the target that does

Our solutions

big picture





- Not a CEGAR / CEGIS loop
  - first, find an abstraction
  - second, refine as much as possible
- Solving and Refining:
  - sequences of $\forall\exists$-formulas
  - witnessing Skolem functions

Our solutions

in more detail

- Symbolic simulation checking in SMT
  - Solving formulas of this kind: $\forall \vec{x}.S(\vec{x}) \implies \exists \vec{y}.T(\vec{x},\vec{y})$
- The abstraction-refinement-based algorithm SimAbs
  - Abstractions and simulations are constructed automatically
- Implicit existential abstraction
  - Making some variables from T existentially quantified
- Refinement based on Skolem relations
  - Extracted from valid $\forall\exists$-formulas
- Horn clause-based Skolemizer AE-VAL
  - Based on iterative construction of MBPs
- LLVM-based implementation and evaluation
  - Checking correctness of compiler optimizations

SimAbs in action

source

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

target

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

SimAbs in action

source

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

target

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

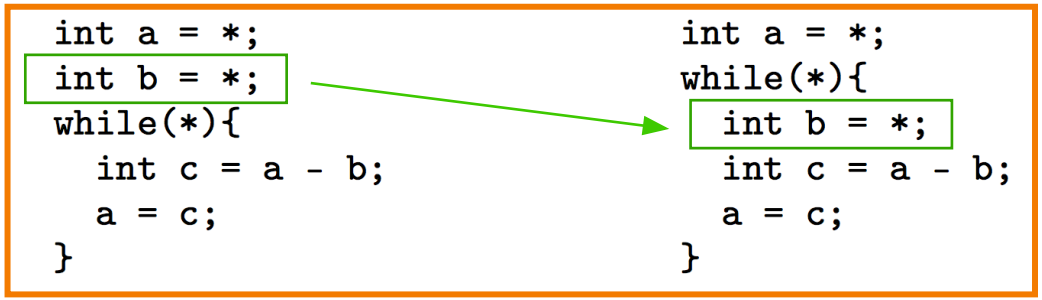
not a simulation relation:

$$(a_S = a_T \wedge b_S = b_T)$$

SimAbs in action


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

existential abstraction of the target



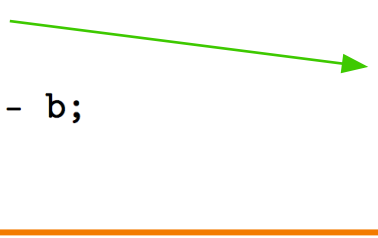
SimAbs in action

source

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

existential abstraction of the target

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



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

simulation relation

$(a_S = a_T)$

What makes the formula valid?

✗ INVALID

$$\forall a_S, b_S, a'_S, b'_S, a_T, b_T.$$
$$(a_S = a_T \wedge b_S = b_T) \wedge (a'_S = a_S + b_S \wedge b'_S = b_S) \implies$$
$$\exists a'_T, b_T, c_T.(c_T = a_T - b_T \wedge a'_T = c_T) \wedge (a'_S = a'_T \wedge b'_S = b'_T)$$

✓ VALID

$$\forall a_S, b_S, a'_S, b'_S, a_T.$$
$$(a_S = a_T) \wedge (a'_S = a_S + b_S \wedge b'_S = b_S) \implies$$
$$\exists b_T, a'_T, b'_T, c_T.(c_T = a_T - b_T \wedge a'_T = c_T) \wedge (a'_S = a'_T)$$

b_T is now existentially quantified and thus can be skolemized: $b_T = -b_S$

What makes the formula valid?

✘ INVALID

$$\forall a_S, b_S, a'_S, b'_S, a_T, b_T.$$
$$(a_S = a_T \wedge b_S = b_T) \wedge (a'_S = a_S + b_S \wedge b'_S = b_S) \implies$$
$$\exists a'_T, b'_T, c_T.(c_T = a_T - b_T \wedge a'_T = c_T) \wedge (a'_S = a'_T \wedge b'_S = b'_T)$$

✔ VALID

$$\forall a_S, b_S, a'_S, b'_S, a_T.$$
$$(a_S = a_T) \wedge (a'_S = a_S + b_S \wedge b'_S = b_S) \implies$$
$$\exists b_T, a'_T, b'_T, c_T.(c_T = a_T - b_T \wedge a'_T = c_T) \wedge (a'_S = a'_T)$$

?

$$\forall a_S, b_S, a'_S, b'_S, a_T.$$
$$(a_S = a_T \wedge \boxed{b_S = -b_T}) \wedge (a'_S = a_S + b_S \wedge b'_S = b_S) \implies$$
$$\exists a'_T, b'_T, c_T.(c_T = a_T - b_T \wedge a'_T = c_T) \wedge (a'_S = a'_T \wedge \boxed{b'_S = -b'_T})$$

SimAbs in action

source

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

target

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

simulation relation:

$$(a_S = a_T \wedge b_S = -b_T)$$

AE-VAL solver and skolemizer

- Decision procedure for $\forall \vec{x}.S(\vec{x}) \implies \exists \vec{y}.T(\vec{x},\vec{y})$
  - Each model of S needs to be extended to a model of T
- Skolem extracting capabilities
  - To describe relations between the matched models
  - Skolem relations in the ite-format
- Can be done naively
  - Iterative enumeration of all models of S
- A more elegant way needs a generalization
  - With the help of Model-based Projections
  - With the help of a Horn solver

Deciding Validity of ∀∃-formulas

- Expensive to find $\varphi(\vec{x}) = \exists \vec{y}.T(\vec{x},\vec{y})$
  - Instead, it is cheaper to find an under-approximation of $\varphi(\vec{x})$

[Komuravelli et al. 2014]

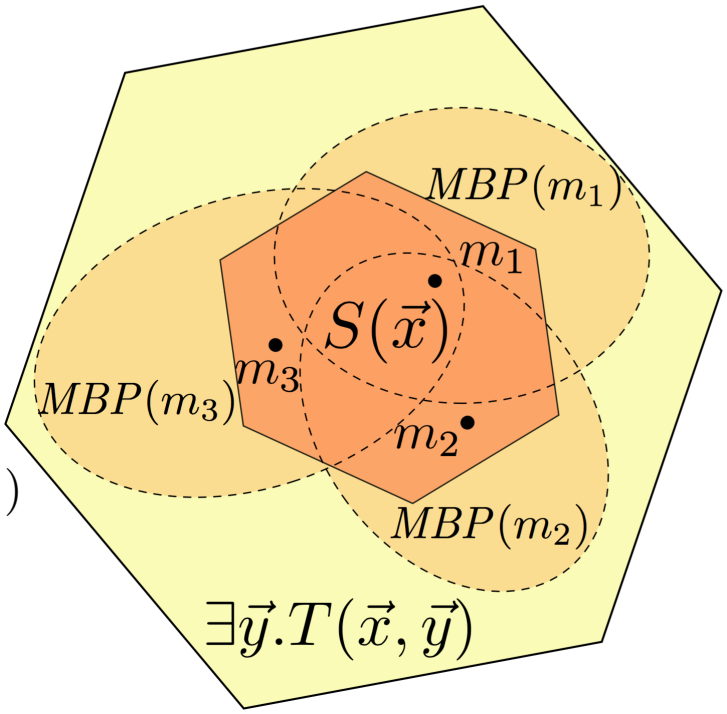
- Pick a model $m \models S(\vec{x}) \wedge T(\vec{x},\vec{y})$ and construct $MBP(m)$ s.t.:
  - (1) $m \models MBP(m)$
  - (2) $MBP(m) \implies \exists \vec{y}.T(\vec{x},\vec{y})$

- Allows iteratively find a *finite* coverage:

$$MBP(m_1) \vee MBP(m_2) \vee MBP(m_3) \implies \exists \vec{y}.T(\vec{x},\vec{y})$$

$$S(\vec{x}) \implies MBP(m_1) \vee MBP(m_2) \vee MBP(m_3)$$

- Loos-Weispfenning quantifier elimination
  - Using virtual substitution for LRA



Horn clause-based Skolemization

- *Local* Skolem relation for each MBP:

$$sk_i(\vec{x},\vec{y}) \implies \big(MBP(m_i) \iff T(\vec{x},\vec{y})\big)$$

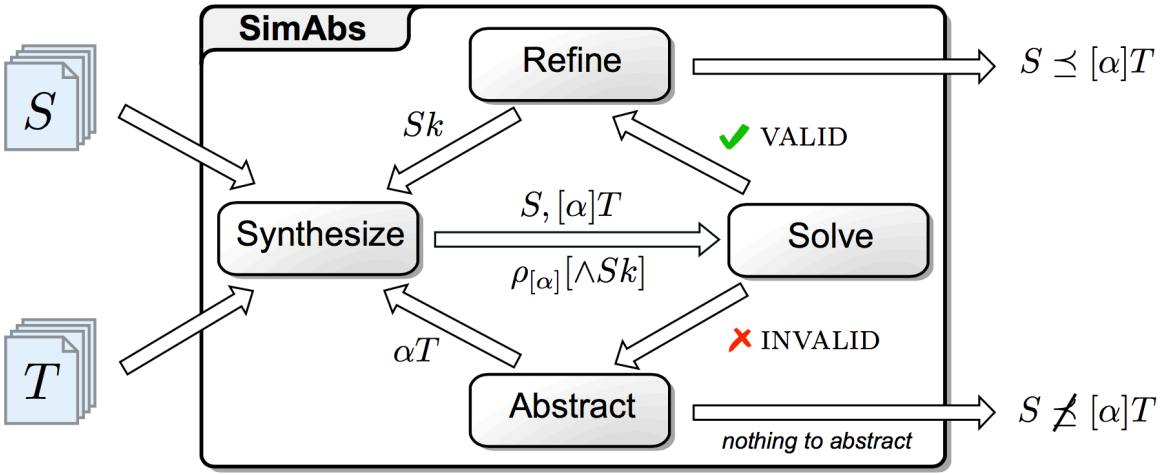
- Composing *local* Skolems:

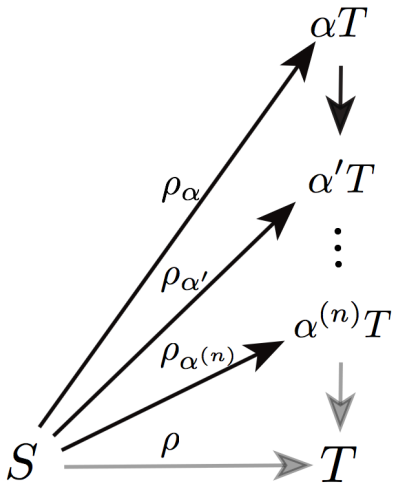
$$Sk_{\vec{y}}(\vec{x},\vec{y}) \equiv \begin{cases} sk(m_1) & \text{if } I_1(\vec{x}) \\ sk(m_2) & \text{else if } I_2(\vec{x}) \\ sk(m_3) & \text{else } I_3(\vec{x}) \end{cases}$$

- Generating guards
  - *By creating and solving a system of Horn Clauses:*

$$\begin{cases} S(\vec{x}) \wedge MBP(m_1) \implies I_1(\vec{x}) \\ S(\vec{x}) \wedge MBP(m_2) \wedge \neg MBP(m_1) \implies I_2(\vec{x}) \\ S(\vec{x}) \wedge MBP(m_3) \wedge \neg MBP(m_1) \wedge \neg MBP(m_2) \implies I_3(\vec{x}) \\ S(\vec{x}) \wedge I_1(\vec{x}) \wedge \neg MBP(m_1) \implies \bot \\ S(\vec{x}) \wedge I_2(\vec{x}) \wedge \neg MBP(m_2) \implies \bot \\ S(\vec{x}) \wedge I_3(\vec{x}) \wedge \neg MBP(m_3) \implies \bot \end{cases}$$

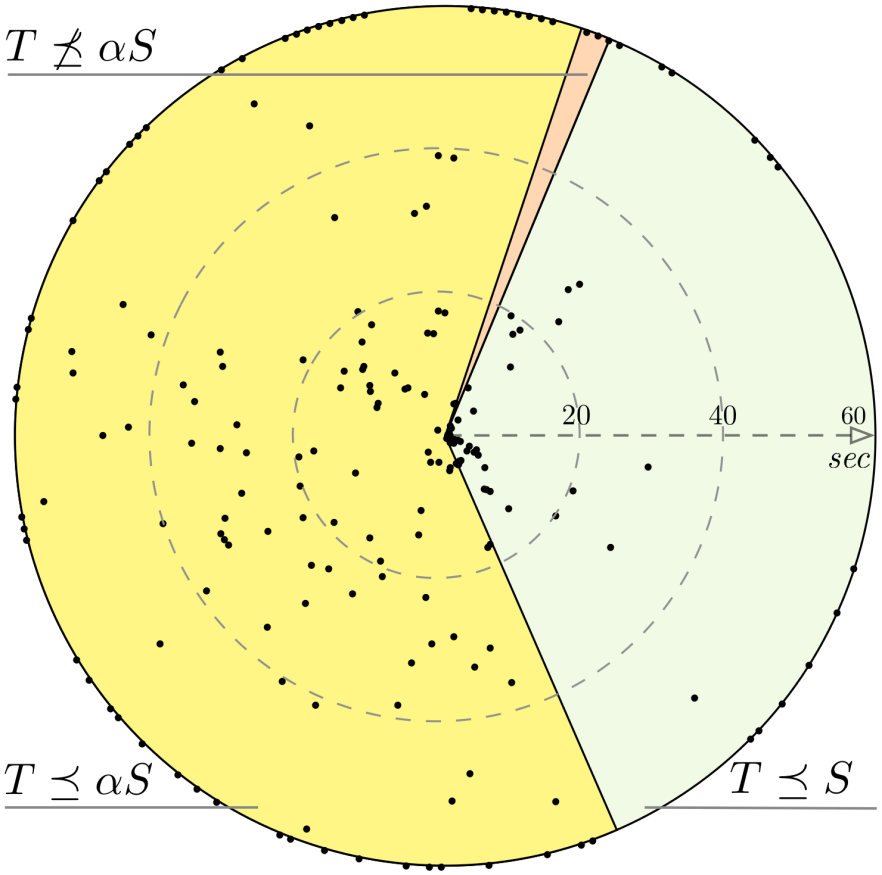
Implementation and Evaluation





- Tools
  - LLVM compiler
  - UFO model checker (Z3/PDR engine) for preprocessing
  - AE-VAL / MBP is based on Z3 (currently supports LRA)
- Benchmarks
  - SV-COMP (0.3 – 5K LoC)
  - 228 pairs of programs
  - LLVM-optimization passes
  - `constprop -globalopt -instcombine -simplifycfg -mem2reg –adce`

Evaluation



- 228 pairs of programs (timeout: 10 min)
- `-constprop -globalopt -instcombine -simplifycfg -mem2reg –adce`
- simulation by identity: 65 pairs
- abstract/concrete simulation with Skolem: 160 pairs
- no simulation: 3 pairs

Related work

- *[Milner 1971]*
  - Algebraic notion of simulation preorder
- *[Dill et al. 1991]*
  - The first symbolic approach to construct simulation with BDDs
- *[Henzinger et al. 1995]*
  - Game-theoretic approach to check simulation
- *[Necula 2000]*
  - Translation validation
- *[Namjoshi et al. 2013]*
  - Constructing simulations by augmenting program transformers
- *[Lahiri et al. 2013]*
  - Invariant generation of the programs combined together
- *[Felsing et al. 2014]*
  - Synthesis of relational specifications using Horn Solving

We have proposed the first SMT-based approach to simulation synthesis

To wrap up

- Contributions
  - Synthesize simulation relations and abstractions
  - Deciding validity of $\forall\exists$-formulas and Skolemization
- Implementation
  - AE-VAL / MBP is based on Z3 (currently supports LRA)
  - SimAbs is based on UFO model checker
- Evaluation
  - Benchmark based on SVCOMP
  - Verifying correctness of LLVM optimizations
  - Killing mutants for mutation testing
- Future
  - Enhancing AE-VAL with bit-vectors, LIA and other theories
  - Skolem minimization and factoring techniques
  - Property Directed Equivalence

Università della Svizzera italiana

Faculty of Informatics

Thank you!