Semantics-Parametric Program Equivalence (work-in-progress)

Ștefan Ciobâcă

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
stefan.ciobaca@info.uaic.ro

April 2018

(joint work with Dorel Lucanu)

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISCDI, project number PN-III-P2-2.1-BG-2016-0394, within PNCDI III.

Outline

Program Verifiers

1. Programming languages should have formal semantics;
2. Verification tools should be sound w.r.t. said semantics;
3. Typical (research-level) workflow today:
   - 3.1 Develop formal semantics of language;
   - 3.2 Develop verification method;
   - 3.3 Prove verification method sound.
4. Trouble: redo word with every new feature in language, bug in semantics, new language version, entirely new language.

Solution: Semantics-Parametric Verifiers[1]

1. Design a semantics S of some language;
2. Test it for bugs;
3. A verifier V should take as input a program P and the semantics S:
   - 3.1 $V(P, S)$ should be yes, no, unknown, timeout, depending on what property V checks of P;
   - 3.2 prove V sound;
4. If bug found in S, run $V(P, S')$ (no need to redo the soundness proof of V);
5. If feature added to S, run $V(P, S_2)$ (no need to redo soundness proof of V).

[1] Idea due to Grigore Roșu

Today's Talk: Semantics-Parametric Language Equivalence

Design a verifier V such that:

- V takes as input two programs: P and Q and the semantics of the languages L and R of P and Q;
- $V(P, Q, L, R)$ checks whether the programs P and Q are equivalent, when P is interpreted w.r.t. the semantics L and Q w.r.t. to R;
- Nice side-effect: can check equivalence of programs in different languages.

Outline

Rewriting Modulo Theories

Constrained term rewriting rule:

$$l \rightarrow r \text{ if } b$$

- l and r = terms with uninterpreted and interpreted symbols;
- b = boolean constraint.

Example: $f(n \times 2) \rightarrow f(n)$ if $odd(n)$.
(Symbolic) rewriting can be implemented with help of SMT solvers.
Operational semantics of languages can be encoded as constrained rewrite rules.

A Small IMPerative Language

$$
\begin{array}{lcl}
\mathit{AExp} & ::= & \mathit{Int} \mid \mathit{Id} \mid \mathit{AExp}\ \texttt{binop}\ \mathit{AExp} \\
\mathit{BExp} & ::= & \mathit{Bool} \mid \mathit{BExp}\ \texttt{binop}\ \mathit{BExp} \mid \texttt{uop}\ \mathit{BExp} \mid \mathit{AExp}\ \texttt{relop}\ \mathit{AExp} \\
\mathit{Stmt} & ::= & \mathit{skip} \mid \mathit{Stmt};\mathit{Stmt} \mid \\
 & & \mathit{Id}{:=}\mathit{AExp} \mid \\
 & & \mathit{while}\ \mathit{BExp}\ \mathit{do}\ \mathit{Stmt} \mid \\
 & & \mathit{if}\ \mathit{BExp}\ \mathit{then}\ \mathit{Stmt}\ \mathit{else}\ \mathit{Stmt} \\
\mathit{Code} & ::= & \mathit{AExp} \mid \mathit{BExp} \mid \mathit{Stmt} \\
\mathit{Cfg} & ::= & \mathit{Stack}\{\mathit{Code}\} \times \mathit{Map}\{\mathit{Id}\}\{\mathit{Int}\}
\end{array}
$$

Operational Semantics of IMP

$$\begin{array}{lll} \mathit{Code} & ::= & \mathit{AExp} \mid \mathit{BExp} \mid \mathit{Stmt} \\ \mathit{Cfg} & ::= & \mathit{Stack}\{\mathit{Code}\} \times \mathit{Map}\{\mathit{Id}\}\{\mathit{Int}\} \end{array}$$

Configurations: $\langle c_1 \rightsquigarrow c_2 \rightsquigarrow \ldots \rightsquigarrow c_n \rightsquigarrow \epsilon, env \rangle$.

- $\langle x := i \rightsquigarrow s, env \rangle \rightarrow \langle s, update(env, x, i) \rangle$;
- $\langle x := e \rightsquigarrow s, env \rangle \rightarrow \langle e \rightsquigarrow x := \square \rightsquigarrow s, env \rangle$ if $\neg isval(e)$;
- $\langle e_1 + e_2 \rightsquigarrow s, env \rangle \rightarrow \langle e_1 \rightsquigarrow \square + e_2 \rightsquigarrow s, env \rangle$ if $\neg isval(e_1)$;
- $\langle i_1 \rightsquigarrow \square + e_2 \rightsquigarrow s, env \rangle \rightarrow \langle i_1 + e_2 \rightsquigarrow s, env \rangle$;
- $\langle i_1 + e_2 \rightsquigarrow s, env \rangle \rightarrow \langle e_2 \rightsquigarrow i_1 + \square \rightsquigarrow s, env \rangle$ if $\neg isval(e_2)$;
- $\langle i_2 \rightsquigarrow i_1 + \square \rightsquigarrow s, env \rangle \rightarrow \langle i_1 + i_2 \rightsquigarrow s, env \rangle$;
- $\langle i_1 + i_2 \rightsquigarrow s, env \rangle \rightarrow \langle i_1 +_{Int} i_2 \rightsquigarrow s, env \rangle$;
- $\langle i \rightsquigarrow x := \square \rightsquigarrow s, env \rangle \rightarrow \langle x := i \rightsquigarrow s, env \rangle$.

Operational Semantics of IMP

$$\begin{array}{lll} \mathit{Code} & ::= & \mathit{AExp} \mid \mathit{BExp} \mid \mathit{Stmt} \\ \mathit{Cfg} & ::= & \mathit{Stack}\{\mathit{Code}\} \times \mathit{Map}\{\mathit{Id}\}\{\mathit{Int}\} \end{array}$$

Configurations: $\langle c_1 \rightsquigarrow c_2 \rightsquigarrow \ldots \rightsquigarrow c_n \rightsquigarrow \epsilon, \mathit{env} \rangle$.
Example:

1. $\langle \mathtt{x} := x + 2 \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
2. $\langle x + 2 \rightsquigarrow \mathtt{x} := \square \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
3. $\langle x \rightsquigarrow \square + 2 \rightsquigarrow \mathtt{x} := \square \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
4. $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow \mathtt{x} := \square \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
5. $\langle 12 + 2 \rightsquigarrow \mathtt{x} := \square \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
6. $\langle 14 \rightsquigarrow \mathtt{x} := \square \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
7. $\langle \mathtt{x} := 14 \rightsquigarrow \epsilon, x \mapsto 12 \rangle \rightarrow$
8. $\langle \epsilon, x \mapsto 14 \rangle \not\rightarrow$.

Control structures

1. $\langle \text{if true then } s_1 \text{ else } s_2 \rightsquigarrow s, env\rangle \rightarrow \langle s_1 \rightsquigarrow s, env\rangle$;
2. $\langle \text{if false then } s_1 \text{ else } s_2 \rightsquigarrow s, env\rangle \rightarrow \langle s_2 \rightsquigarrow s, env\rangle$;
3. $\langle \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow s, env\rangle \rightarrow \langle e \rightsquigarrow \text{if } \square \text{ then } s_1 \text{ else } s_2 \rightsquigarrow s, env\rangle$ *if* $\neg isval(e)$;
4. $\langle \text{while } e \text{ do } s_1 \rightsquigarrow s, env\rangle \rightarrow$
   $\langle \text{if } e \text{ then } s_1; \text{while } e \text{ do } s_1 \text{ else } skip \rightsquigarrow s, env\rangle$.

Reachability Properties

Constrainted term: t if ϕ.
Semantics: $[\![t \text{ if } \phi]\!] =$ all instances of t that satisfy ϕ.
Reachability properties:

$$t \text{ if } \phi \Rightarrow^{+} t' \text{ if } \phi'.$$

(IJCAR 2018) Stefan Ciobaca, Dorel Lucanu. *A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems*.

Outline

Running Example

```
c := n;
n := 1;
while (c != 1)
  n := n + 1;
  if (c % 2 != 0)
    then c := 3 * c + 1
    else c := c / 2
```

μf.λn.λa.

```
  if n != 1
   then
    if n % 2 != 0
      then f (3 * n + 1) (a + 1)
      else f (n / 2) (a + 1)
   else
      a
```

Equivalence Proofs

Assume L is the IMPerative language and R is a FUNctional language, described as LCTRSs.

Let $\mathit{Cfg} ::= (\mathit{CfgL}, \mathit{CfgR})$.

Let $E = \{(\langle \_, \mathit{env} \rangle, x)$ if $\mathit{lookup}(\mathit{env}, \mathtt{n}) = x\}$ be the terminal configurations that are considered equivalent (base case).

Let $R = \{(P, Q)$ if $\phi\}$ be a set of pairs of configurations (constrained by ϕ) that we want to show equivalent.

Proof System for Program Equivalence

To prove that R contains equivalent configurations:

$$
\begin{array}{l}
\forall (P, Q) \mathit{if}\ \phi \in R : \\
\quad \exists P', Q', \phi' \text{ s.t.:} \\
\qquad P \text{ if } \phi \Rightarrow_L^+ P' \text{ if } \phi' \wedge \\
\qquad Q \text{ if } \phi \Rightarrow_R^+ Q' \text{ if } \phi' \wedge \\
\qquad (P', Q') \text{ if } \phi' \in [\![R \cup E]\!]
\end{array}
$$

Demo Time

Outline

Conclusion

1. Implementation of equivalence-checking based on LCTRSs working on toy examples.
2. Completeness? Automation? Expressiveness?

μf.λn.

```
if n != 1
 then
  if n % 2 != 0
    then 1 + f (3*n+1)
    else 1 + f (n/2)
  else
    0
```

μf.λn.λa.

```
if n != 1
 then
  if n % 2 != 0
    then f (3*n+1) (a+1)
    else f (n/2) (a+1)
  else
    a
```

$$R = \left\{ \begin{array}{l} (\underbrace{1+1+\ldots+1}_{m \text{ times}} + f\ n, f\ n\ m) \text{ if true,} \\ (\underbrace{1+1+\ldots+1}_{k \text{ times}} + l, m) \text{ if } k +_{Int} l = m \end{array} \right\}$$

https://fmse.info.uaic.ro/event/from-2018/