Reducing Total Correctness to Partial Correctness by a Transformation of the Language Semantics

Sebastian Buruiană[1] Ștefan Ciobâcă[2]

[1]Alexandru Ioan Cuza University
Bitedefender

[2]Alexandru Ioan Cuza University

WPTE 2018

Overview

Program Verifiers

- Programming languages should have formal semantics;
- Verifiers should be sound w.r.t. said semantics;
- Typical workflow today:
  1. Develop formal semantics of language;
  2. Develop verification method;
  3. Prove that verification method is sound.
- Problem: work has to be redone with every change in the language (new features, new versions etc).

Solution: Semantics-Parametric Verifiers

- A verifier V should take as input a program P and the semantics S;
- $V(P, S)$ should be yes, no, unknown, timeout (depending on what property of P is checked by V);
- Prove V sound;
- If semantics changes to S', run $V(P, S')$ (no reed to redo soundness proof of V).

Today's Talk : Semantics-Parametric reduction of Total Correctness to Partial Correctness

- Assume we have a verifier V^1 s.t. $V(P, S)$ checks whether the program P is partially correct when interpreted w.r.t. the semantics S;
- Apply some transformations to P and S and obtain $\theta(P)$ and $\theta(S)$;
- $V(\theta(P), \theta(S))$ guarantees Total Correctness of program P when interpreted w.r.t. the semantics S.

[1] Andrei Ştefănescu et al. "All-Path Reachability Logic". In: *RTA-TLCA*. 2014, pp. 425–440. DOI: http://dx.doi.org/10.1007/978-3-319-08918-8_29.

Example : IMP language

Syntax of IMP

$$
\begin{array}{ll}
\mathit{Id} ::= & \mathtt{x} \mid \mathtt{y} \mid \mathtt{z} \mid \ldots \\
\mathit{Int} ::= & 0, 1, -1, \ldots \\
\mathit{Bool} ::= & \mathit{True} \mid \mathit{False} \\
\mathit{AE} ::= & \mathit{Int} \mid \mathit{Id} \mid \mathit{AE} + \mathit{AE} \mid \ldots \\
\mathit{BE} ::= & \mathit{Bool} \mid \mathit{AE} = \mathit{AE} \mid \mathit{AE} < \mathit{AE} \mid \mathit{not}\ \mathit{BE} \mid \ldots \\
\mathit{Stmt} ::= & \mathit{skip} \\
& \mid \mathit{Stmt};\ \mathit{Stmt} \\
& \mid \mathit{Id} := \mathit{AE} \\
& \mid \text{while } \mathit{BE} \text{ do } \mathit{Stmt} \\
& \mid \text{if } \mathit{BE} \text{ then } \mathit{Stmt} \text{ else } \mathit{Stmt}
\end{array}
$$

Example : IMP language

Configurations in IMP

$$\begin{array}{ll} Code ::= & AE \mid BE \mid Stmt \\ Cfg ::= & List\,\{Code\} \times Map\,\{Id\}\,\{Int\} \end{array}$$

$$\langle c_1 \rightsquigarrow c_2 \rightsquigarrow \ldots \rightsquigarrow c_n \rightsquigarrow \epsilon \mid env \rangle$$

Language semantics

$$\begin{array}{lll} \langle (v := i) \rightsquigarrow l \mid env \rangle & \Rightarrow & \langle l \mid update(v, i, env) \rangle \\ \langle (\text{if } b \text{ then } s_1 \text{ else } s_2) \rightsquigarrow l \mid env \rangle & \Rightarrow & \langle s_1 \rightsquigarrow l \mid env \rangle \text{ if } b = True \end{array}$$

$$\begin{array}{ll} & \langle (\text{while } b\ s) \rightsquigarrow l \mid env \rangle \\ \Rightarrow & \langle (\text{if } b \text{ then } (s; \text{while } b\ s) \text{ else } skip) \rightsquigarrow l \mid env \rangle \end{array}$$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \to$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \to$
- $\langle x \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \to$
- $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \to$
- $\langle 12 + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \to$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 12 + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 14 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 12 + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle 14 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$
- $\langle x := 14 \rightsquigarrow \epsilon \mid x \mapsto 12 \rangle \rightarrow$

Example : IMP language

Program execution

- $\langle x := x + 2 \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle x + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle x \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle 12 \rightsquigarrow \square + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle 12 + 2 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle 14 \rightsquigarrow x := \square \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle x := 14 \rightsquigarrow \epsilon \mid x \mapsto 12\rangle \to$
- $\langle \epsilon \mid x \mapsto 14\rangle$

Example : IMP language

Partial Correctness

An *all-path reachability rule* is a pair $\varphi \Rightarrow^{\forall} \varphi'$. We say that $\varphi \Rightarrow^{\forall} \varphi'$ is *satisfied* by S, denoted by $S \models \varphi \Rightarrow^{\forall} \varphi'$, iff for all complete paths τ starting with γ and for all valuations ρ such that $(\gamma, \rho) \models \varphi$, there exists some $\gamma' \in \tau$ such that $(\gamma', \rho) \models \varphi'$.

SUM Program in IMP

s := 0
while not (m = 0) do s := s + m; m := m - 1

Partial Correctness Sequent

$$S \vdash \langle SUM \mid env_1 \rangle \wedge lookup(\mathtt{m}, env_1) = n \wedge n \geq 0 \Rightarrow^{\forall}$$
$$\exists env_2.(\langle skip \mid env_2 \rangle \wedge lookup(\mathtt{s} env_2) = n(n+1)/2),$$

Reducing Total Correctness to Partial Correctness

Total Correctness

We say that an all-path reachability rule $\varphi \Rightarrow^{\forall} \varphi'$ is *totally satisfied* by S, denoted by $S \models_t \varphi \Rightarrow^{\forall} \varphi'$, iff for all complete or diverging executions τ starting with γ and for all valuations ρ such that $(\gamma, \rho) \models \varphi$, there exists some $\gamma' \in \tau$ such that $(\gamma', \rho) \models \varphi'$

Reducing Total Correctness to Partial Correctness

Total Correctness

We say that an all-path reachability rule $\varphi \Rightarrow^{\forall} \varphi'$ is *totally satisfied* by S, denoted by $S \models_t \varphi \Rightarrow^{\forall} \varphi'$, iff for all complete or diverging executions τ starting with γ and for all valuations ρ such that $(\gamma, \rho) \models \varphi$, there exists some $\gamma' \in \tau$ such that $(\gamma', \rho) \models \varphi'$

Semantics transformation

$$(\langle (v := i) \rightsquigarrow l \mid env \rangle, n) \quad \Rightarrow \quad (\langle l \mid update(v, i, env) \rangle, n - 1)$$

This sequent guarantees total correctness

$$\theta(S) \vdash (\langle SUM \mid env_1 \rangle, 200|n| + 200) \wedge lookup(\mathtt{m}, env_1) = n \wedge n \geq 0 \Rightarrow^{\forall}$$
$$\exists g, env_2.((\langle skip \mid env_2 \rangle, g) \wedge lookup(\mathtt{s}, env_2) = n(n+1)/2),$$

Total Correctness Theorem

Theorem

If there exists some term $s \in Term_{\Sigma,Nat}(Var)$ of sort Nat such that

$$\theta(S) \models \theta(\varphi, s) \Rightarrow^{\forall} \exists M.\theta(\varphi', M),$$

where $M \in Var_{Nat}$, then $S \models_t \varphi \Rightarrow^{\forall} \varphi'$.

Corollary

If there exists $s \in Term_{\Sigma,Nat}(Var)$ of sort Nat such that $\theta(S) \models \theta(\varphi, s) \Rightarrow^{\forall} \exists M.\theta(\varphi', M)$, where $M \in Var_{Nat}$, then:

- $S \models \varphi \Rightarrow^{\forall} \varphi'$;
- If φ' terminates in S, then φ also terminates in S.

Conclusion and Future Work

- Language semantics transformation that can be used to prove total correctness of programs;
- Working proof-of-concept implementation.

`http://github.com/ciobaca/rmt`

- More modular alternative to program variants?
- Can our method be combined with existing state of the art automated termination provers?

Thank you

Questions?

References

Ciobâcă, Ștefan and Dorel Lucanu. "A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems". In: *IJCAR 2018*. (to appear).

Ștefănescu, Andrei et al. "All-Path Reachability Logic". In: *RTA-TLCA*. 2014, pp. 425–440. DOI: http://dx.doi.org/10.1007/978-3-319-08918-8_29.