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

Sebastian Buruiană\textsuperscript{1} Ștefan Ciobâcă\textsuperscript{2}

\textsuperscript{1}Alexandru Ioan Cuza University
Bitedefender

\textsuperscript{2}Alexandru Ioan Cuza University

WPTE 2018
Overview

1. Context and Introduction
2. Formalism for Language Semantics
3. Total Correctness
4. Conclusion and Future Work
5. Questions
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).
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 need to redo soundness proof of $V$).
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 Ștefănescu 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

\[
Id ::= x | y | z | \ldots \\
Int ::= 0, 1, -1, \ldots \\
Bool ::= True | False \\
AE ::= Int | Id | AE + AE | \ldots \\
BE ::= Bool | AE = AE | AE < AE | \text{not } BE | \ldots \\
Stmt ::= \text{skip} \\
\quad | Stmt; Stmt \\
\quad | Id ::= AE \\
\quad | \text{while } BE \text{ do } Stmt \\
\quad | \text{if } BE \text{ then } Stmt \text{ else } Stmt
\]
Example: IMP language

Configurations in IMP

\[
\text{Code ::= } \text{AE} \mid \text{BE} \mid \text{Stmt} \\
\text{Cfg ::= } \text{List }\{\text{Code}\} \times \text{Map }\{\text{Id}\} \{\text{Int}\}
\]

\[\langle c_1 \rightsquigarrow c_2 \rightsquigarrow \ldots \rightsquigarrow c_n \rightsquigarrow \epsilon \mid \text{env}\rangle\]

Language semantics

\[\langle (v := i) \rightsquigarrow l \mid \text{env}\rangle \Rightarrow \langle l \mid \text{update}(v, i, \text{env}) \rangle\]
\[\langle (\text{if } b \text{ then } s_1 \text{ else } s_2) \rightsquigarrow l \mid \text{env}\rangle \Rightarrow \langle s_1 \rightsquigarrow l \mid \text{env}\rangle \text{ if } b = \text{True}\]

\[\langle (\text{while } b \ s) \rightsquigarrow l \mid \text{env}\rangle \Rightarrow \langle (\text{if } b \text{ then } (s; \text{while } b \ s) \text{ else } \text{skip}) \rightsquigarrow l \mid \text{env}\rangle\]
Example: IMP language

Program execution

\[ \langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \]
Example: IMP language

Program execution

\[ \langle x := x + 2 \leadsto \epsilon | x \mapsto 12 \rangle \rightarrow \]
\[ \langle x + 2 \leadsto x := \emptyset \leadsto \epsilon | x \mapsto 12 \rangle \rightarrow \]
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

\[ \langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \]
\[ \langle x + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \]
\[ \langle x \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \]
\[ \langle 12 \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \]
Example: IMP language

Program execution:

1. $\langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
2. $\langle x + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
3. $\langle x \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
4. $\langle 12 \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
5. $\langle 12 + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
Example: IMP language

Program execution

\[
\langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \\
\langle x + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \\
\langle x \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \\
\langle 12 \leadsto \square + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \\
\langle 12 + 2 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow \\
\langle 14 \leadsto x := \square \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow 
\]
Example: IMP language

Program execution

1. $\langle x := x + 2 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
2. $\langle x + 2 \leadsto x := \Box \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
3. $\langle x \leadsto \Box + 2 \leadsto x := \Box \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
4. $\langle 12 \leadsto \Box + 2 \leadsto x := \Box \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
5. $\langle 12 + 2 \leadsto x := \Box \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
6. $\langle 14 \leadsto x := \Box \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
7. $\langle x := 14 \leadsto \epsilon \mid x \mapsto 12 \rangle \rightarrow$
**Example: IMP language**

**Program execution**

1. \( \langle x := x + 2 \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
2. \( \langle x + 2 \sim x := \square \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
3. \( \langle x \sim \square + 2 \sim x := \square \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
4. \( \langle 12 \sim \square + 2 \sim x := \square \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
5. \( \langle 12 + 2 \sim x := \square \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
6. \( \langle 14 \sim x := \square \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
7. \( \langle x := 14 \sim \epsilon \mid x \mapsto 12 \rangle \rightarrow \)
8. \( \langle \epsilon \mid x \mapsto 14 \rangle \rightarrow \)
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 $\tau$ starting with $\gamma$ and for all valuations $\rho$ such that $\langle \gamma, \rho \rangle \models \varphi$, there exists some $\gamma' \in \tau$ such that $\langle \gamma', \rho \rangle \models \varphi'$.

SUM Program in IMP

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

Partial Correctness Sequent

$$S \vdash \langle SUM \mid env_1 \rangle \land lookup(m, env_1) = n \land n \geq 0 \Rightarrow \forall \exists env_2. (\langle skip \mid env_2 \rangle \land lookup(senv_2) = n(n + 1)/2),$$
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 $\tau$ starting with $\gamma$ and for all valuations $\rho$ 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 \textit{totally satisfied} by $S$, denoted by $S \models_t \varphi \Rightarrow \forall \varphi'$, iff for all complete or diverging executions $\tau$ starting with $\gamma$ and for all valuations $\rho$ such that $(\gamma, \rho) \models \varphi$, there exists some $\gamma' \in \tau$ such that $(\gamma', \rho) \models \varphi'$

Semantics transformation

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

This sequent guarantees total correctness

$$\theta(S) \vdash (\langle SUM \mid env_1 \rangle, 200|n| + 200) \land lookup(m, env_1) = n \land n \geq 0 \Rightarrow \forall \exists g, env_2.((\langle skip \mid env_2 \rangle, g) \land lookup(s, env_2) = n(n + 1)/2),$$
Total Correctness Theorem

**Theorem**
If there exists some term \( s \in \text{Term}_{\Sigma, \text{Nat}}(\text{Var}) \) of sort \( \text{Nat} \) such that
\[
\theta(S) \models \theta(\varphi, s) \Rightarrow \forall \exists M. \theta(\varphi', M),
\]
where \( M \in \text{Var}_{\text{Nat}} \), then \( S \models_t \varphi \Rightarrow \forall \varphi' \).

**Corollary**
If there exists \( s \in \text{Term}_{\Sigma, \text{Nat}}(\text{Var}) \) of sort \( \text{Nat} \) such that
\[
\theta(S) \models \theta(\varphi, s) \Rightarrow \forall \exists M. \theta(\varphi', M), \text{ where } M \in \text{Var}_{\text{Nat}}, \text{ then:}
\]
- \( S \models \varphi \Rightarrow \forall \varphi' \);
- If \( \varphi' \) terminates in \( S \), then \( \varphi \) 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
