Axiomatic Semantics for Compiler Verification

Steven Schäfer Sigurd Schneider Gert Smolka

Programming Systems Lab
Saarland University
Germany

January 19, 2016

Overview

- Formally verify a compiler from a non-deterministic language of guarded commands (GC)...

$$\begin{aligned} \textbf{if } x \geq 0 &\Rightarrow s := 1 \\ \| \; x \leq 0 &\Rightarrow s := -1 \end{aligned}$$

- to a determinisitc low-level language of imperative continuations (IC)...

$$\textbf{if } x \leq 0 \textbf{ then } s := -1 \textbf{ else } s := 1$$

- using axiomatic semantics.

Compiler Verification

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\frac{\langle P \rangle\, s\, \langle Q \rangle}{\langle P \rangle\, \mathcal{C}\, s\, \langle Q \rangle}$$

compiler correctness:

$$\forall s\, P\, Q.\ \langle P \rangle\, s\, \langle Q \rangle \implies \langle P \rangle\, \mathcal{C}\, s\, \langle Q \rangle$$

s : high-level program (GC)

P, Q : pre- & Postcondition

$\langle P \rangle\, s\, \langle Q \rangle$: (total) correctness judgement

$\mathcal{C}$: compiler (GC $\rightarrow$ IC)

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Guarded Commands (GC)
   - Axiomatic Semantics
2. Imperative Continuations (IC)
   - Operational Semantics
   - Axiomatic Semantics
   - Soundness & Completeness
3. Compiling GC to IC

Guarded Commands [Dijkstra75]

Syntax

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$
\begin{array}{rcll}
\sigma, \tau & : & \Sigma & \text{states} \\
a & : & \Sigma \to \Sigma & \text{actions} \\
b & : & \Sigma \to \mathbb{B} & \text{guards} \\
s, t & ::= & \textbf{skip} \mid a \mid s; t \mid \textbf{if}\ G \mid \textbf{do}\ G & \text{programs} \\
G & ::= & b_1 \Rightarrow s_1 \parallel \ldots \parallel b_n \Rightarrow s_n & (\text{n} > 0)
\end{array}
$$

Example (Greatest Common Divisor)

$$
\begin{aligned}
\textbf{do}\ x > y &\Rightarrow x := x - y \\
\parallel y > x &\Rightarrow y := y - x
\end{aligned}
$$

Guarded Commands

Semantics

Correctness Judgements

$\langle \sigma \rangle\, s\, \langle Q \rangle \;\approx$ **all** runs of s starting in σ terminate with a state in Q.

$$\langle P \rangle\, s\, \langle Q \rangle \;:=\; \forall \sigma.\; P\,\sigma \implies \langle \sigma \rangle\, s\, \langle Q \rangle$$
$$\mathrm{wp}\, s\, Q \;:=\; \lambda \sigma.\; \langle \sigma \rangle\, s\, \langle Q \rangle$$

Defined inductively analogous to **big-step semantics**, e.g.

$$\frac{\sigma s \Downarrow \tau \qquad \tau t \Downarrow \theta}{\sigma(s;t) \Downarrow \theta} \implies \frac{\langle \sigma \rangle\, s\, \langle P \rangle \qquad \langle P \rangle\, t\, \langle Q \rangle}{\langle \sigma \rangle\, s; t\, \langle Q \rangle}$$

Guarded Commands

Semantics (cont)

- Conditionals

$$\frac{\widehat{G}\sigma \qquad \forall (b \Rightarrow s) \in G.\ \ b\sigma \implies \langle \sigma \rangle\, s\, \langle Q \rangle}{\langle \sigma \rangle\ \textbf{if}\ \ G\, \langle Q \rangle}$$

- Loops

$$\frac{\langle \sigma \rangle\ \textbf{if}\ \ G\, \langle P \rangle \qquad \langle P \rangle\ \textbf{do}\ \ G\, \langle Q \rangle}{\langle \sigma \rangle\ \textbf{do}\ \ G\, \langle Q \rangle} \qquad \frac{\neg \widehat{G}\sigma \qquad Q\sigma}{\langle \sigma \rangle\ \textbf{do}\ \ G\, \langle Q \rangle}$$

$$\langle P \rangle\, s\, \langle Q \rangle := \forall \sigma.\ P\sigma \to \langle \sigma \rangle\, s\, \langle Q \rangle$$

$$\widehat{G} := \lambda \sigma.\ \exists (b \Rightarrow s) \in G.\ b\sigma$$

Imperative Continuations [Schneider et. al. 14]

Syntax

$$
\begin{array}{rcll}
a & : & \Sigma \to \Sigma & \text{actions} \\
b & : & \Sigma \to \mathbb{B} & \text{guards} \\
f, g, \mathsf{ret} & : & \mathcal{L} & \text{labels} \\
s, t & ::= & a; s \mid \textbf{if } b \textbf{ then } s \textbf{ else } t \mid \textbf{def } f = s \textbf{ in } t \mid f & \text{programs}
\end{array}
$$

Example (Greatest Common Divisor)

$$
\begin{array}{l}
\textbf{def } f = \textbf{ if } x > y \textbf{ then } x := x - y;\ f \textbf{ else} \\
\qquad\qquad\quad \textbf{if } y > x \textbf{ then } y := y - x;\ f \textbf{ else} \\
\qquad\qquad\quad \textbf{ret} \\
\textbf{in } f
\end{array}
$$

Imperative Continuations

Small-step operational semantics

- Actions

$$(a; s, \sigma) \rhd (s, a\sigma)$$

- Conditionals

$$\frac{b\sigma}{(\textbf{if } b \textbf{ then } s \textbf{ else } t, \sigma) \rhd (s, \sigma)} \qquad \frac{\neg b\sigma}{(\textbf{if } b \textbf{ then } s \textbf{ else } t, \sigma) \rhd (t, \sigma)}$$

- Local definitions

$$(\textbf{def } f = s \textbf{ in } t, \sigma) \rhd (t^{f}_{\textbf{def } f=s \textbf{ in } s}, \sigma)$$

Imperative Continuations

Axiomatic Semantics

- One post-condition per label $\mathcal{Q} : \mathcal{L} \to (\Sigma \to \mathbf{P})$
- Local definitons

$$\frac{\mathcal{Q} f \sigma}{\langle \sigma \rangle\, f\, \langle \mathcal{Q} \rangle}$$

$$\frac{\langle \sigma \rangle\, t\, \langle \mathcal{Q}[f \mapsto P] \rangle \qquad \langle P \rangle\, \mathbf{def}\ f = s\ \mathbf{in}\ s\, \langle \mathcal{Q} \rangle}{\langle \sigma \rangle\, \mathbf{def}\ f = s\ \mathbf{in}\ t\, \langle \mathcal{Q} \rangle}$$

$$\langle P \rangle\, s\, \langle Q \rangle := \forall \sigma.\ P\sigma \implies \langle \sigma \rangle\, s\, \langle Q \rangle$$

$$\mathcal{Q}[f \mapsto P] := \lambda g.\ \text{if } g = f \text{ then } P \text{ else } \mathcal{Q}\, g$$

Structural Properties of Axiomatic Semantics

- Monotonicity:

$$\mathcal{P} \subseteq \mathcal{Q} \implies \mathsf{wp}\, s\, \mathcal{P} \subseteq \mathsf{wp}\, s\, \mathcal{Q}$$

- Distributivity:

$$\mathsf{wp}\, s\, (\mathcal{P} \cap \mathcal{Q}) \equiv \mathsf{wp}\, s\, \mathcal{P} \cap \mathsf{wp}\, s\, \mathcal{Q}$$

- Continuity:
  For $\mathcal{Q}_1 \subseteq \mathcal{Q}_2 \subseteq \mathcal{Q}_3 \subseteq \ldots$

$$\mathsf{wp}\, s \left(\bigcup_{n \in \mathbb{N}} \mathcal{Q}_n \right) \equiv \bigcup_{n \in \mathbb{N}} \mathsf{wp}\, s\, \mathcal{Q}_n$$

where $\mathsf{wp}\, s\, \mathcal{Q} = \lambda \sigma.\ \langle \sigma \rangle\, s\, \langle \mathcal{Q} \rangle$

Recursive Definition of WP

Theorem

The following equivalences hold for IC.

$$
\begin{aligned}
\mathsf{wp}\, f\, \mathcal{Q} &\equiv \mathcal{Q}\, f \\
\mathsf{wp}\, (a; s)\, \mathcal{Q} &\equiv \mathsf{wp}\, s\, \mathcal{Q} \circ a \\
\mathsf{wp}\, (\textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ s\ \textbf{\textit{else}}\ t)\, \mathcal{Q} &\equiv \lambda\sigma.\ \text{if}\ b\sigma\ \text{then}\ \mathsf{wp}\, s\, \mathcal{Q}\, \sigma\ \text{else}\ \mathsf{wp}\, t\, \mathcal{Q}\, \sigma \\
\mathsf{wp}\, (\textbf{\textit{def}}\ f = s\ \textbf{\textit{in}}\ t)\, \mathcal{Q} &\equiv \mathsf{wp}\, t\, \mathcal{Q}[f \mapsto \mathsf{fix}\, F] \\
\text{where}\ F\, P &:= \mathsf{wp}\, s\, \mathcal{Q}[f \mapsto P]
\end{aligned}
$$

Where $\mathsf{fix}\, F$ *is the least fixed-point of* F *and* $\mathsf{wp}\, s\, \mathcal{Q} = \lambda\sigma.\ \langle\sigma\rangle\, s\, \langle\mathcal{Q}\rangle$.

Soundness & Completeness

Lemma (Substitution Lemma)

$$\mathsf{wp}\, s_t^f\, \mathcal{Q} \;\equiv\; \mathsf{wp}\, s\, \mathcal{Q}[f \mapsto \mathsf{wp}\, t\, \mathcal{Q}]$$

Lemma (Invariance under Reduction)

$$(s,\sigma) \rhd (t,\tau) \implies (\langle\sigma\rangle\, s\, \langle\mathcal{Q}\rangle \iff \langle\tau\rangle\, t\, \langle\mathcal{Q}\rangle)$$

Theorem (Soundness & Completeness)

$$\langle\sigma\rangle\, s\, \langle\mathcal{Q}\rangle \iff \exists f\, \tau.\;\; (s,\sigma) \rhd^* (f,\tau) \;\wedge\; \mathcal{Q}\, f\, \tau$$

Compiling GC to IC - Overview

Theorem (Compiler correctness)

$$\langle \sigma \rangle \, s \, \langle Q \rangle \implies \langle \sigma \rangle \, \mathcal{C} \, s \, \langle \mathbf{ret} \mapsto Q \rangle$$

- The compiler is a function $\mathcal{C} : \mathrm{GC} \to \mathrm{IC}$
- On termination in IC, call **ret**

Compiling GC to IC - Details

- Sequence guarded command sets.

$$\textbf{if } b_1 \Rightarrow s_1 \parallel b_2 \Rightarrow s_2 \parallel b_3 \Rightarrow s_3$$
$$\rightsquigarrow \textbf{if } b_2 \textbf{ then } s_2 \textbf{ else if } b_3 \textbf{ then } s_3 \textbf{ else } s_1$$

- Translate loops to jumps.

$$\textbf{do } b_1 \Rightarrow s_1 \parallel b_2 \Rightarrow s_2$$
$$\rightsquigarrow \textbf{def } f = \textbf{if } b_1 \textbf{ then } s_1; f \textbf{ else}$$
$$\textbf{if } b_2 \textbf{ then } s_2; f \textbf{ else ret}$$
$$\textbf{in } f$$

- Linearize programs.

$$(\textbf{do } b \Rightarrow (a_1; a_2); a_3); u$$
$$\rightsquigarrow \textbf{def } f = (\textbf{if } b \textbf{ then } a_1; a_2; a_3; f \textbf{ else } u) \textbf{ in } f$$

Sequencing Guarded Commands

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\begin{aligned} \mathcal{F} &: (\mathsf{GC} \to \mathsf{IC}) \to \mathsf{IC} \to \{\mathsf{GC}\} \to \mathsf{IC} \\ \mathcal{F}\, f\, v\, \varnothing &:= v \\ \mathcal{F}\, f\, v\, (b \Rightarrow s \parallel G) &:= \textbf{if } b \textbf{ then } f\, s \textbf{ else } \mathcal{F}\, f\, v\, G \end{aligned}$$

- *Flatten* a guarded command set, given
  - translation function f
  - default case v.
- Correctness judgements in IC

$$\frac{\widehat{G}\sigma \qquad \forall (b \Rightarrow s) \in G.\, b\sigma \implies \langle \sigma \rangle\, f\, s\, \langle Q \rangle}{\langle \sigma \rangle\, \mathcal{F}\, f\, v\, G\, \langle Q \rangle} \qquad \frac{\neg \widehat{G}\sigma \qquad \langle \sigma \rangle\, v\, \langle Q \rangle}{\langle \sigma \rangle\, \mathcal{F}\, f\, v\, G\, \langle Q \rangle}$$

Linearizing GC programs

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$
\begin{aligned}
\mathcal{T} &: \mathsf{IC} \to \mathsf{GC} \to \mathsf{IC} \\
\mathcal{T}\, u\, (s; t) &:= \mathcal{T}\, (\mathcal{T}\, u\, t)\, s \\
\mathcal{T}\, u\, (\textbf{if}\ \mathbf{b} \Rightarrow \mathbf{s} \parallel G) &:= \mathcal{F}\, (\mathcal{T}\, u)\, (\mathcal{T}\, u\, \mathbf{s})\, G \\
\mathcal{T}\, u\, (\textbf{do}\ G) &:= \textbf{def}\ \mathbf{f} = \mathcal{F}\, (\mathcal{T}\, \mathbf{f})\, u\, G\ \textbf{in}\ \mathbf{f} \qquad (\mathbf{f}\ \text{fresh}) \\
&\vdots
\end{aligned}
$$

- $\mathcal{T}\, u\, s \approx$ execute s, then u.
- $\textbf{if} \varnothing$ is undefined behavior.

Compiler from GC to IC

$$\mathcal{C}\, s \;:=\; \mathcal{T}\, \mathsf{ret}\, s$$

Lemma

$\mathcal{T}$ sequences a GC and an IC program into an IC program.

$$\frac{\langle \sigma \rangle\, s\, \langle P \rangle \qquad \langle P \rangle\, u\, \langle Q \rangle}{\langle \sigma \rangle\, \mathcal{T}\, u\, s\, \langle Q \rangle}$$

Theorem (Compiler correctness)

$$\langle \sigma \rangle\, s\, \langle Q \rangle \implies \langle \sigma \rangle\, \mathcal{C}\, s\, \langle \mathsf{ret} \mapsto Q \rangle$$

Avoiding exponential blowup

- Abstract the continuation when compiling conditionals...

$$\mathcal{T}\, u\, (\textbf{if}\ b \Rightarrow s \parallel G) =$$
$$\quad \textbf{let}\ f = u\ \textbf{in}\ \mathcal{F}\, (\mathcal{T}\, f)\, (\mathcal{T}\, f\, s)\, G \qquad\qquad (f \text{ fresh})$$

- ...if we have to.

$$\textbf{let}\ f = s\ \textbf{in}\ t \quad := \quad \begin{cases} \textbf{def}\ g = s\ \textbf{in}\ t_g^f, \ (g \text{ fresh}) & \text{if } |s| > 1 \\ t_s^f & \text{otherwise} \end{cases}$$

- Correctness: $\mathsf{wp}\, (\textbf{let}\ f = s\ \textbf{in}\ t)\, \mathcal{Q} \equiv \mathsf{wp}\, t\, \mathcal{Q}[f \mapsto \mathsf{wp}\, s\, \mathcal{Q}]$
- **Does not complicate the proof of compiler correctness.**

Paper & Coq Development

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Program equivalence $s \cong t \;:=\; \forall \mathcal{Q}.\ \mathsf{wp}\, s\, \mathcal{Q} \equiv \mathsf{wp}\, t\, \mathcal{Q}$
  - ... is a congruence (easy proof with wp!)
  - ... is contextual equivalence (under mild assumptions)
- Continuity + relation to Dijkstra's definition of wp.

- Coq development uses de Bruijn representation.
- Substitution lemmas are generalized over parallel substitutions.
- $\sim$ 300 lines of spec, $\sim$ 600 lines of proofs[1]

[1] Using Ssreflect [Gonthier,Mahboubi,Tassi08] and Autosubst [Schäfer,Tebbi,Smolka15]

Related Work

- Refinements for Compiler Verification
  - **"On the Translation of Procedures to Finite Machines."** M. Müller-Olm, A. Wolf. TOPLAS (2000)
  - **"Compiler verification meets cross-language linking via data abstraction."** P. Wang, S. Cuellar, and A. Chlipala. OOPSLA (2014)
- Axiomatic Semantics
  - **"Verification of sequential imperative programs in Isabelle-HOL."** N. Schirmer. Diss. Technical University Munich (2006)
  - **"Program verification through characteristic formulae."** A. Charguéraud. ACM Sigplan Notices 45.9 (2010)
- Compiler Verification
  - **"Formal verification of a realistic compiler."** X. Leroy. Communications of the ACM 52.7 (2009)
  - **"CakeML: a verified implementation of ML."** R. Kumar, M. O. Myreen, M. Norrish, and S. Owens. ACM Sigplan Notices 49.1 (2014)

Summary

- Use **axiomatic semantics** to show **compiler correctness!**

- Future Work
  - How to handle non-termination? (also preserve liberal specifications?)
  - How to handle effects (I/O) gracefully?
  - Generally: How to handle richer languages?

Fin

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Paper & Coq development at:
http://www.ps.uni-saarland.de/extras/ascv/

Fixpoints in Type Theory

Definition

$$\text{fix} \;:\; ((X \to \mathbf{P}) \to X \to \mathbf{P}) \to X \to \mathbf{P}$$
$$\text{fix}\, F\, x \;:=\; \forall P.\; F\, P \subseteq P \implies P\, x$$

Lemma (Knaster-Tarski)

For monotone F, the predicate $\text{fix}\, F$ *is the least fixed point of F.*

$$\text{fix}\, F \equiv F(\text{fix}\, F)$$