Axiomatic Semantics

Adam Chlipala
MIT

[based on Armando Solar-Lezama's slides]

6.820
October 16, 2013
Related Reading

Winskel, Chapter 6,
“The axiomatic semantics of IMP”
Motivation

Consider the following program:

```c
z = 0;
i = x;
while (i) {
    z = z + y;
i = i - 1;
}
```

Claim: for any initial values of $x$ and $y$,
- The loop will terminate.
- When it does, $z = x \times y$.

How can we prove this?
Motivation

The tools we have seen so far are insufficient.

- **Operational semantics**
  - Easy to argue that a given input will produce a given output.
  - Also easy to argue that all constructs in the language will preserve some property (like when we proved type soundness).
  - Much harder to prove general properties of the behavior of a program on all inputs.

- **Type-based reasoning**
  - Types allow us to design custom checkers to verify specific properties.
  - Very good at reasoning about properties of the data pointed at by particular variables.
Axiomatic Semantics
(AKA program logics)

A system for proving properties about programs

Key idea:
- We can define the semantics of a construct by describing its effect on *assertions* about the program state.

Two components
- A language for stating assertions (“the assertion logic”)
  - Can be First-Order Logic (FOL), a specialized logic such as separation logic, or Higher-Order Logic (HOL), which can encode the others.
- Many specialized languages developed over the years:
  - Z, Larch, JML, Spec#
- Deductive rules (“the program logic”) for establishing the truth of such assertions
A little history

Early years: Unbridled optimism
- Heavily endorsed by the likes of Hoare and Dijkstra
- If you can prove programs correct, bugs will be a thing of the past.
- You won’t even have to test your programs!

The middle ages
- 1979 paper by DeMillo, Lipton and Perlis
  - “Proofs in math only work because there is a social process in place to get people to argue them and internalize them.”
  - “Program proofs are too boring for social process to form around them.”
  - “Programs change too fast and proofs are too brittle.”

The renaissance
- New generation of automated reasoning tools
- A handful of success stories
- Better appreciation of costs, benefits and limitations?
The basics

Hoare triple
- If the precondition holds before stmt and stmt terminates, postcondition will hold afterward.

This is a partial correctness assertion.
- We sometimes use the notation

\[ [A] \text{ stmt } [B] \]

to denote a total correctness assertion

which means you also have to prove termination. [more next class]
What do assertions mean?

We first need to introduce a language.

For today we will be using Winskel’s IMP:

\[
e := n \mid x \mid e_1 + e_2 \mid e_1 - e_2
\]

\[
c := x := e \mid c_1; c_2 \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c
\]

Big Step Semantics have two kinds of judgments: expressions result in values; commands change the state.

\[
\langle e, \sigma \rangle \rightarrow n \quad \langle c, \sigma \rangle \rightarrow \sigma'
\]
Semantics of IMP

Commands mutate the state:

\[
\langle e, \sigma \rangle \rightarrow e' \\
\langle X := e, \sigma \rangle \rightarrow \sigma[X \rightarrow e']
\]

\[
\langle c_1, \sigma \rangle \rightarrow \sigma'' \quad \langle c_2, \sigma'' \rangle \rightarrow \sigma' \\
\langle c_1; c_2, \sigma \rangle \rightarrow \sigma'
\]

\[
\langle e_1, \sigma \rangle \rightarrow \text{false} \quad \langle c_f, \sigma \rangle \rightarrow \sigma' \\
\langle \text{if } e_1 \text{ then } c_t \text{ else } c_f, \sigma \rangle \rightarrow \sigma'
\]

\[
\langle e_1, \sigma \rangle \rightarrow \text{true} \quad \langle c_t, \sigma \rangle \rightarrow \sigma' \\
\langle \text{if } e_1 \text{ then } c_t \text{ else } c_f, \sigma \rangle \rightarrow \sigma'
\]

What about loops?
Semantics of IMP

The definition for loops must be recursive.

\[
\begin{align*}
\langle e_1, \sigma \rangle & \rightarrow false \\
\langle \text{while } e_1 \text{ then } c, \sigma \rangle & \rightarrow \sigma
\end{align*}
\]

\[
\begin{align*}
\langle e_1, \sigma \rangle & \rightarrow true & \langle c; \text{while } e_1 \text{ then } c, \sigma \rangle & \rightarrow \sigma' \\
\langle \text{while } e_1 \text{ then } c, \sigma \rangle & \rightarrow \sigma'
\end{align*}
\]

\[
\begin{align*}
\langle e_1, \sigma \rangle & \rightarrow true & \langle c, \sigma \rangle & \rightarrow \sigma'' & \langle \text{while } e_1 \text{ then } c, \sigma'' \rangle & \rightarrow \sigma' \\
\langle \text{while } e_1 \text{ then } c, \sigma \rangle & \rightarrow \sigma'
\end{align*}
\]
What do assertions mean?

The language of assertions:
- \( A := \text{true} \mid \text{false} \mid e_1 = e_2 \mid e_1 \geq e_2 \mid A_1 \land A_2 \mid \neg A \mid \forall x. A \)

Notation \( \sigma \models A \) means that the assertion holds on state \( \sigma \).
- This is defined inductively over the structure of \( A \).
- Ex.
  \begin{align*}
  \sigma \models A \text{ and } B & \iff \sigma \models A \text{ and } \sigma \models B
  \end{align*}

Partial Correctness can then be defined in terms of OS:
\( \{A\} \implies \{B\} \iff \forall \sigma \forall \sigma' (\sigma \models A \land \langle c, \sigma \rangle \rightarrow \sigma') \Rightarrow \sigma' \models B \)
Or we can just use Coq formulas!

A special syntactically defined language of assertions is the classic choice for axiomatic semantics, but we can also just use Coq's language of formulas, since we already have some practice with it!

Its logical decision problems are undecidable, but it's still quite pleasant to work with.

Important point: a given program logic can often support many different assertion logics, mix-and-match style.
Defining axiomatic semantics

Establishing the truth of a Hoare triple in terms of the operational semantics is impractical.

The real power of AS is the ability to establish the validity of a Hoare triple by using deduction rules.

- $\vdash \{ A \} c \{ B \}$ means we can deduce the triple from a set of basic axioms.
Derivation Rules

Derivation rules for each language construct

\[ \vdash \{A[x \to e]\} x := e \{A\} \]
\[ \vdash \{A \land b\} c_1 \{B\} \quad \vdash \{A \land \neg b\} c_2 \{B\} \]
\[ \vdash \{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\} \]
\[ \vdash \{A \land b\} c \{A\} \]
\[ \vdash \{A\} \text{while } b \text{ do } c \{A \land \neg b\} \]
\[ \vdash \{A\} c_1 \{C\} \quad \vdash \{C\} c_2 \{B\} \]
\[ \vdash \{A\} c_1 ; c_2 \{B\} \]

Can be combined together with the rule of consequence

\[ \vdash A' \Rightarrow A \vdash \{A\} c \{B\} \vdash B \Rightarrow B' \]
\[ \vdash \{A'\} c \{B'\} \]
Soundness and Completeness

What does it mean for our deduction rules to be sound?
- You will never be able to prove anything that is not true.
- Truth is defined in terms of our original definition of \( \{A\} \subset \{B\} \).

\[
\forall \sigma \forall \sigma'(\sigma \vdash A \land \langle c, \sigma \rangle \rightarrow \sigma') \Rightarrow \sigma' \vdash B
\]
- We can prove this, but it’s tricky.
  - (Actually, most trickiness comes from handling of variables, which we'll avoid in the Coq code coming up next!)

What does it mean for them to be complete?
- If a statement is true, we should be able to prove it via deduction.
- See Winskel Chapter 7 if you're curious about the details.

So are they complete?
- yes and no
- They are complete relative to the logic.
- ...but there are no complete and consistent logics for elementary arithmetic (Gödel).
Coq code demo,
starting from L12-AxiomaticSemantics-Template.v
A Small IMP Extension

Consider adding assertion checking to IMP:
\[
e := n \mid x \mid e_1 + e_2 \mid e_1 - e_2
\]
\[
c := x := e \mid c_1; c_2 \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c
\]
\[
\mid \text{assert}(A) \mid \text{skip}
\]

New rule: \((\sigma, \text{assert}(A))\) evaluates to \(\sigma\), \text{iff} \(A(\sigma)\).

Does our statement of Hoare triple soundness continue to allow us to prove the program correctness properties that we care about?

No! It only tells us about \textit{terminating} executions, but now programs can \textbf{fail to terminate} because they \textbf{crash} with \textbf{assertion failures}!
A Type Soundness-Style Proof

**Soundness.** If \((\sigma, c) \rightarrow^* (\sigma', c')\)

And \(\{A\} c \{B\}\)

And \(A(\sigma)\)

Then \((c' = \text{skip} \land B(\sigma'))\)

Or \(\exists \sigma'', c'' . (\sigma', c') \rightarrow (\sigma'', c'').\)

**Progress.** If \(\{A\} c \{B\}\)

And \(A(\sigma)\)

Then \((c = \text{skip} \land B(\sigma))\)

Or \(\exists \sigma', c'. (\sigma, c) \rightarrow (\sigma', c').\)

**Preservation.** If \((\sigma, c) \rightarrow (\sigma', c')\)

And \(\{A\} c \{B\}\)

Then \(\exists A'. \{A'\} c' \{B\} \land A'(\sigma').\)

See this lecture's Coq code for formal proofs.