Big-Step and Small-Step Semantics for Abstract Imp

Gert Smolka, Saarland University

October 23, 2017

We specify the semantics of a simple imperative language Imp with two complementary techniques known as big-step and small-step semantics and show that the two semantics agree. Both semantics are realized with inductive predicates.

1 States, Actions, Tests

We have in mind a simple imperative language that can write and read registers realized with an abstract memory. All registers hold values of the same type (e.g., integers). We abstract away from concrete values and assume the following types and functions:

$$s : \mathsf{State} \;:\; \mathbf{T} \qquad\qquad$$
$$a : \mathsf{Action} \;:\; \mathbf{T} \qquad \alpha : \mathsf{Action} \to \mathsf{State} \to \mathsf{State}$$
$$b : \mathsf{Test} \;:\; \mathbf{T} \qquad \beta : \mathsf{Test} \to \mathsf{State} \to \mathsf{B}$$

Concrete examples for actions are assignments like $x := 2$ and $x := x + y$. Concrete examples for tests are comparisons like $x > 0$ and $x \le y + z$. Note that the semantics of actions and tests is given by the functions α and β.

For examples it will be useful to assume an action and a test

$$\mathsf{skip} : \mathsf{Action} \qquad\qquad \mathsf{tt} : \mathsf{Test}$$

such that $\alpha\,\mathsf{skip}\,s = s$ and $\beta\,\mathsf{tt}\,s = \mathsf{true}$ for all states s.

2 Commands

The **commands** of our language Imp are obtained with actions, sequentialisations, conditionals, and loops:

$$c : \mathsf{Com} \ ::= \ a \mid c_1; c_2 \mid \mathsf{if}\ b\ c \mid \mathsf{while}\ b\ c$$

Commands are executed on states. The execution of a command on a state may terminate or not terminate. If execution of a command on a state terminates, it yields a state. We speak of the initial and the final state of an execution. Informally, we may describe the semantics of commands as follows:

- Action a: The state is transformed with αa.
- Sequentialisation $c_1; c_2$: First execute c_1, then execute c_2.
- Conditional $\mathsf{if}\ b\ c$: If b yields true, execute c, otherwise do nothing.
- Loop $\mathsf{while}\ b\ c$: execute c as long as b yields true.

Nontermination comes into play through loops: The loop $(\mathsf{while\ tt\ skip})$ does not terminate.

3 Big-Step Semantics

The big-step semantics of Imp describes a recursive interpreter function that given a command and a state computes the resulting state. Since type theory admits only total functions and the interpretation of a command may not terminate, we are forced to formalize the interpreter function as an inductive predicate. We employ a predicate

$$\vdash : \mathsf{State} \to \mathsf{Com} \to \mathsf{State} \to \mathbf{P}$$

which we define with the following rules using the notation $s, c \vdash s'$:

$$\frac{}{s, a \vdash \alpha a s} \qquad \frac{s, c_1 \vdash s' \qquad s', c_2 \vdash s''}{s, c_1; c_2 \vdash s''}$$

$$\frac{\beta b s = \mathsf{true} \qquad s, c \vdash s'}{s, \mathsf{if}\ b\ c \vdash s'} \qquad \frac{\beta b s = \mathsf{false}}{s, \mathsf{if}\ b\ c \vdash s}$$

$$\frac{\beta b s = \mathsf{true} \qquad s, c \vdash s' \qquad s', \mathsf{while}\ b\ c \vdash s''}{s, \mathsf{while}\ b\ c \vdash s''} \qquad \frac{\beta b s = \mathsf{false}}{s, \mathsf{while}\ b\ c \vdash s}$$

Note that the recursions coming with the rule for sequentialisation and the first rule for loops are binary. Without the first rule for loops the rules yield an always terminating interpreter since each recursion step employs a smaller command.

Fact 1 (Functionality) If $s, c \vdash s'$ and $s, c \vdash s''$, then $s' = s''$.

Proof By induction on $s, c \vdash s'$. ∎

Exercise 2 (Nontermination) Informally, for every command c, execution of the loop $\mathsf{while}\ \mathsf{tt}\ c$ does not terminate. Prove $\neg\exists s'.\ s, \mathsf{while}\ \mathsf{tt}\ c \vdash s'$ for all states s and all commands c.

4 Small-Step Semantics

We now specify a second semantics for Imp that models single reductions steps and provides for an iterative interpreter. We do this with an inductive predicate

$$\succ : \mathsf{State} \times \mathsf{L}(\mathsf{Com}) \to \mathsf{State} \times \mathsf{L}(\mathsf{Com}) \to \mathbf{P}$$

for which we use the notation $s, C \succ s', C'$. The list of commands represents a stack of commands needed so that the binary recursion coming with sequentialisations and loops can be accounted for iteratively. We will show that the equivalence

$$s, [c] \succ^* s', \mathsf{nil} \quad \leftrightarrow \quad s, c \vdash s' \tag{1}$$

where $\succ^*$ denotes the reflexive transitive closure of $\succ$. The equivalence says that the small-step semantics $s, C \succ s', C'$ agrees with the big-step semantics $s, c \vdash s'$.

We define the inductive predicate realising the small-step semantics with the following rules:

$$\frac{}{s, a :: C \succ \alpha a s, C} \qquad \frac{}{s, c_1; c_2 :: C \succ s, c_1 :: c_2 :: C}$$

$$\frac{\beta b s = \mathsf{true}}{s, \mathsf{if}\ b\ c :: C \succ c :: C, s} \qquad \frac{\beta b s = \mathsf{false}}{s, \mathsf{if}\ b\ c :: C \succ s, C}$$

$$\frac{\beta b s = \mathsf{true}}{s, \mathsf{while}\ b\ c :: C \succ s, c :: \mathsf{while}\ b\ c :: C} \qquad \frac{\beta b s = \mathsf{false}}{s, \mathsf{while}\ b\ c :: C \succ s, C}$$

We also define the reflexive transitive closure $\succ^*$ of $\succ$ as an inductive predicate:

$$\frac{}{s, C \succ^* s, C} \qquad \frac{s, C \succ s', C' \qquad s', C' \succ^* s'', C''}{s, C \succ^* s'', C''}$$

Fact 3

1. If $s, C \succ s', C'$ and $s, C \succ s'', C''$, then $s' = s''$ and $C' = C''$.

2. If $s, C \succ s', C'$, then $s, C \succ^* s', C'$,
3. If $s, C \succ^* s', C'$ and $s', C' \succ^* s'', C''$, then $s, C \succ^* s'', C''$.
4. If $s, C \succ s', C'$, then $s, C + D \succ s', C' + D$.
5. If $s, C \succ^* s', C'$, then $s, C + D \succ^* s', C' + D$.

5 Agreement

We now show that the small-step semantics agrees with the big-step semantics of Imp as specified by the equivalence (1). The two directions require different proofs. The direction from big-step to small-step follows by induction on the big-step predicate if the claim is generalised.

Lemma 4 If $s, c \vdash s'$, then $s, c :: C \succ^* s', C$.

Proof By induction on $s, c \vdash s'$ using Fact 3 (2,3). ∎

The other direction requires an auxiliary predicate $s, C \vdash s'$ defined inductively as follows:

$$\frac{}{s, \mathsf{nil} \vdash s} \qquad \frac{s, c \vdash s' \qquad s', A \vdash s''}{s, c :: A \vdash s''}$$

Lemma 5 (Inversion)

1. If $s, \mathsf{nil} \succ^* s'$, then $s = s'$.
2. If $s, c :: C \vdash s'$, then $s, c \vdash s_1$ and $s_1, C \vdash s'$ for some s_1.
3. If $s, [c] \vdash s'$, then $s, c \vdash s'$.

Lemma 6 (Absorption) If $s, C \succ s', C'$ and $s', C' \vdash s''$, then $s, C \vdash s''$.

Proof Case analysis on $s, C \succ s', C'$ using Lemma 5. ∎

Lemma 7 If $s, C \succ^* s', \mathsf{nil}$, then $s, C \vdash s'$.

Proof Induction on $s, C \succ^* s', \mathsf{nil}$ using Lemma 6. ∎

Theorem 8 $s, c \vdash s'$ if and only if $s, [c] \succ^* s', \mathsf{nil}$.

Proof Follows with lemmas 4, 7, and 5. ∎

Exercise 9 For Lemma 6, verify the case for while with satisfied test by hand.

6 Discussion

The proofs of Lemmas 4 and 6 require the verification of many cases involving many variables. This is tedious and error-prone if done by hand. With Coq, these verifications can be done semi-automatically using the eauto tactic supplied with the relevant inductive predicates. The automation of the inversions needed for the proof of Lemma 6 requires a custom tactic realising Lemma 5 in generalised form using Coq's low-level inversion tactic. See the accompanying Coq development for more information. There the proof script for Lemma 6 is a one-liner. Doing this proof in detail by hand will require dozens of lines.