

Comp 411 Notes - 19 Jan 2005

SEMANTICS FOR A SIMPLE PROGRAMMING LANGUAGE

Reproduced by Mathias Ricken
(mgricken@cs.rice.edu)

1 Important Points from Previous Lecture

- Do not forget homework assigned in lectures. Scribes should put them in the notes.
- We use type systems and semantics to prove guarantees.

2 A Tiny Programming Language: Grammar

The grammar for our tiny programming language is:

$$e \in E ::= T \mid F \mid \text{if } e \text{ then } e \text{ else } e \\ 0 \mid e + \mid e - \mid \text{isZero } e$$

Questions:

- How do we define semantics for this programming language?
- How do we define a type system so we do not confuse booleans and natural numbers?

3 Semantics

There are many different ways to define semantics, but most fall into two major categories:

- operational semantics
- denotational semantics

Depending on who you ask, some will say operational semantics build on denotational semantics or vice versa. We will only touch on denotational semantics and mainly deal with operational semantics.

Among others, there are two important sub-categories of operational semantics:

- big-step semantics
- small-step semantics

These notes will present a series of attempts to define semantics for our programming language. Below is our first attempt. Note that some of the rules are incorrect and will be changed.

$$\begin{array}{ccc}
R_T \frac{}{T \hookrightarrow T} & R_T \frac{}{F \hookrightarrow F} & R_0 \frac{}{0 \hookrightarrow 0} \\
R_+ \frac{}{e+ \hookrightarrow e+} & & R_- \frac{}{e- \hookrightarrow e-} \\
R_{isZeroT} \frac{e \hookrightarrow 0}{\text{isZero } e \hookrightarrow T} & & R_{isZeroF} \frac{e \not\hookrightarrow 0}{\text{isZero } e \hookrightarrow F} \\
R_{ifT} \frac{e_1 \hookrightarrow T, e_2 \hookrightarrow e'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow e'} & & R_{ifF} \frac{e_1 \hookrightarrow F, e_3 \hookrightarrow e'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow e'}
\end{array}$$

A general characteristic of big-step semantics is that it is a function from programs to values. After applying the semantics to a program, we expect to get a value, never another program.

When we look at our semantics above with this in mind, we find a problem with our R_+ and R_- rules: They are axioms and return e unchanged. We, however, want $0 + - \hookrightarrow 0$, for example.

We also do not want to constrain the subexpressions, the es , in our semantics; type systems our BNFs should do that. The only thing we can do with subexpressions is to evaluate them.

We therefore change the R_+ and R_- rules to:

$$R_+ \frac{e \hookrightarrow e'}{e+ \hookrightarrow e'+} \quad R_- \frac{e \hookrightarrow e'+}{e- \hookrightarrow e'}$$

Since e will evaluate to a value in big-step semantics, we will never get a program as e' .

The new R_- rule uses a very common type of pattern matching: If e evaluates to “one more than e' ”, then “one less than e ” evaluates to e' .

We define the set of values as:

$$v \in V ::= T \mid F \mid 0 \mid v+$$

We could now say that $e \hookrightarrow v \in V$, but $V \subset E$. It is better to keep stating that $e \hookrightarrow e \in E$, and then *prove* that $\forall e \in E, e \hookrightarrow v \in V$.

4 Properties of Big-Step Semantics

If a semantic is a big-step semantic, then it will have the following properties:

1. $e \hookrightarrow e_1 \wedge e \hookrightarrow e_2 \Rightarrow e_1 \equiv e_2$ (semantics as function)
2. $e \hookrightarrow e' \Rightarrow e' \in V$ (evaluate to a value)
3. $\forall v \in V, v \hookrightarrow v$ (idempotence of values)

We may have programs like “ $T+$ ”, but we ignore these for now since our language is untyped. A program like “ $0-$ ” is interesting too: We defined our set of values as booleans and natural numbers. We will have to decide what to do here.

Evaluating “ $0-$ ” should cause a runtime error. In a type system, we can make this more explicit and prove that our type system prevents runtime errors. Similarly, if in `if e_1 then e_2 else e_3` , the e_1 evaluates to a natural number, this should be a runtime error. We find that our semantics currently do not cover every case. For some programs, there currently just are no derivations.

To prove that $0 + - \hookrightarrow 0$, we derive the following proof object:

$$\frac{\frac{\frac{}{0 \hookrightarrow 0} R_0}{(0)+ \hookrightarrow 0+} R_+}{(0+)- \hookrightarrow 0} R_-$$

The program “ $0 - +$ ” would generate a runtime error.

With our current semantics, the program “`isZero (0-)`” evaluates to F . This is not what we desire, since “ $0-$ ” by itself generates a runtime error. The program “`isZero T`” also evaluates to F , even though our intuition tells us that this should be a runtime or later a type error.

The use of the antecedent $e \not\hookrightarrow 0$ is problematic. In general, we should avoid negative conditions on evaluation. We can rephrase the $R_{isZeroF}$ rule as:

$$R_{isZeroF} \frac{e \hookrightarrow e' \neq 0}{\text{isZero } e \hookrightarrow F}$$

The program “`isZero (0-)`” now is not covered by this rule anymore. However, “`isZero T`” still evaluates to F . To prevent this, we can modify $R_{isZeroF}$ to:

$$R_{isZeroF} \frac{e \hookrightarrow e' +}{\text{isZero } e \hookrightarrow F}$$

Intuitively, the rule R_+ only works on natural numbers, so booleans are excluded. This rule still does not apply to the program “`isZero 0`” since no e' exists so that $0 \hookrightarrow e'$.

The following summarizes our semantics so far:

$$\begin{array}{ccc}
R_T \frac{}{T \hookrightarrow T} & R_T \frac{}{F \hookrightarrow F} & R_0 \frac{}{0 \hookrightarrow 0} \\
R_+ \frac{e \hookrightarrow e'}{e+ \hookrightarrow e'+} & & R_- \frac{e \hookrightarrow e'+}{e- \hookrightarrow e'} \\
R_{isZeroT} \frac{e \hookrightarrow 0}{isZero\ e \hookrightarrow T} & & R_{isZeroF} \frac{e \hookrightarrow e'+}{isZero\ e \hookrightarrow F} \\
R_{ifT} \frac{e_1 \hookrightarrow T, e_2 \hookrightarrow e'}{if\ e_1\ then\ e_2\ else\ e_3 \hookrightarrow e'} & & R_{ifF} \frac{e_1 \hookrightarrow F, e_3 \hookrightarrow e'}{if\ e_1\ then\ e_2\ else\ e_3 \hookrightarrow e'}
\end{array}$$

5 Making the Semantics a Total Function

These semantics are not a total function: some programs do not evaluate to anything. The partiality here stems from inexhaustive pattern matching.

To make this exhaustive, we can modify our semantics to evaluate to something even if runtime errors are encountered. The result of such an erroneous evaluation could be a special value like “fault”, or we can just decide what normal value $v \in V$ should be returned.

An example of semantics modified this way is below. The rules R_{ifF} and $R_{isZeroF}$ were modified, and a R_{0-} rule was added. Using these semantics, the program “ $e-$ ” evaluates to e if e does not evaluate to a natural number ≥ 0 , “if e_1 then e_2 else e_3 ”, where e_1 is not boolean, evaluates to e_3 , and “isZero e ”, where e is not a natural number, evaluates to F .

$$\begin{array}{ccc}
R_T \frac{}{T \hookrightarrow T} & R_T \frac{}{F \hookrightarrow F} & R_0 \frac{}{0 \hookrightarrow 0} \\
R_+ \frac{e \hookrightarrow e'}{e+ \hookrightarrow e'+} & R_- \frac{e \hookrightarrow e'+}{e- \hookrightarrow e'} & R_{0-} \frac{e \hookrightarrow e' \neq e''+}{e- \hookrightarrow e} \\
R_{isZeroT} \frac{e \hookrightarrow 0}{isZero\ e \hookrightarrow T} & & R_{isZeroF} \frac{e \hookrightarrow e' \neq 0}{isZero\ e \hookrightarrow F} \\
R_{ifT} \frac{e_1 \hookrightarrow T, e_2 \hookrightarrow e'}{if\ e_1\ then\ e_2\ else\ e_3 \hookrightarrow e'} & & R_{ifF} \frac{e_1 \hookrightarrow e'' \neq T, e_3 \hookrightarrow e'}{if\ e_1\ then\ e_2\ else\ e_3 \hookrightarrow e'}
\end{array}$$

Using these semantics, we could prove that every program evaluates to a value, i.e. $\forall e \in E, \exists v \in V, e \hookrightarrow v$. We will not use these semantics further, though.

6 Stratifying the Set of Values

To deal with typing problems and reduce the number of runtime errors that can occur in programs, we can stratify the set of values, i.e. split the set V into two sets, one set B for booleans and one set N for natural numbers.

$$\begin{aligned} b \in B & ::= T \mid F \mid \text{if } b \text{ then } b \text{ else } b \mid \text{isZero } n \\ n \in N & ::= 0 \mid n + \mid n - \mid \text{if } b \text{ then } n \text{ else } n \end{aligned}$$

We split **if** into two separate parts to ensure that the result of evaluating **if** e_1 **then** e_2 **else** e_3 is either a $b \in B$ or a $n \in N$ regardless of what the result of evaluating e_1 is.

This prevents writing safe programs like “**if** T **then** 0 **else** F ” since e_2 and e_3 now must either both be elements of B or both be elements of N . Remember that the set of provably safe programs for some semantics might only be a subset of the set of all safe programs for these semantics:

$$\begin{aligned} B \subset E, N \subset E \\ P = B \cup E, P \subset S \subset E \end{aligned}$$

where P is the set of “provably safe” programs, S is the set of “safe” programs, and E is the set of all programs.

If we modify our semantics to use this grammar, we can also prove that our big-step semantics have the following property:

4. $(\forall b \in B, \exists b' \in B, b \hookrightarrow b') \wedge$
 $(\forall n \in N, \exists n' \in N, n \hookrightarrow n')$ (everything evaluates)

provided we add a special rule to deal with a program like “0-”:

$$R_{0-} \frac{e \hookrightarrow 0}{e- \hookrightarrow 0}.$$

Property 4 has to be proved using simultaneous induction, i.e. both parts have to be proved together.

7 Homework

Due with next problem set: Prove the properties 1 - 4 for the semantics we have developed.

$$\begin{array}{ccc}
R_T \overline{T \hookrightarrow T} & R_T \overline{F \hookrightarrow F} & R_0 \overline{0 \hookrightarrow 0} \\
R_+ \frac{e \hookrightarrow e'}{e_+ \hookrightarrow e'_+} & R_- \frac{e \hookrightarrow e'_+}{e_- \hookrightarrow e'} & R_{0-} \frac{e \hookrightarrow 0}{e_- \hookrightarrow 0} \\
R_{isZeroT} \frac{e \hookrightarrow 0}{\text{isZero } e \hookrightarrow T} & & R_{isZeroF} \frac{e \hookrightarrow e'_+}{\text{isZero } e \hookrightarrow F} \\
R_{ifT} \frac{e_1 \hookrightarrow T, e_2 \hookrightarrow e'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow e'} & & R_{ifF} \frac{e_1 \hookrightarrow F, e_3 \hookrightarrow e'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow e'}
\end{array}$$