Operational Semantics for Bool* Language

Thomas H. Austin
San José State University
thomas.austin@sjsu.edu

February 15, 2021

Abstract

In this paper, we will provide a review of the big-step operational semantics for the Bool* language we discussed in class.

Bool* is a very minimal language that allows us to experiment with operational semantics.

First, we define the valid expressions in our language. These expressions dictate the possibilities of what expressions we may have in our source programs. (Note that other language might also have *statements*. Statements might not evaluate to a value; expressions always will.)

Figure 1 shows the list of expressions and values for the Bool* language. Expressions can be the value `true`, the value `false`, or the conditional expression `if` e `then` e `else` e. Note that conditional expressions have a recursive structure, with 3 sub-expressions.

After evaluating a program, we should be able to produce a value in this language. (In other languages, we might hit a bad situation and need to crash instead; that won't happen in this language.) The valid values for Bool* are `true` and `false`.

With our expressions and values defined, we can now specify the semantics for our language. To do so, we will use the following big-step evaluation relation:

$$e \Downarrow v$$

The above line should be read as "the expression e evaluates to the value v".

Figure 2 shows the big-step evaluation rules for the Bool* language. Of course, there are additional possible rules.

The [B-Value] rule applies when the expression (to the left of "$\Downarrow$") is also a value, as defined in Figure 1. There are no premises for this rule (above the line), meaning that it is an *axiom*. This rule states that a value evaluates to itself, so that `true` evaluates to `true` and `false` evaluates to `false`.

Figure 1: The Bool* language

e ::=		*Expressions*
	`true`	true value
	`false`	false value
	`if` e `then` e `else` e	conditional expressions
v ::=		*Values*
	`true`	true value
	`false`	false value

Figure 2: Big-step semantics for Bool*

Evaluation Rules: $\boxed{e \Downarrow v}$

$$\text{[B-Value]} \quad \frac{}{v \Downarrow v}$$

$$\text{[B-IfTrue]} \quad \frac{e_1 \Downarrow \texttt{true} \quad e_2 \Downarrow v}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Downarrow v}$$

$$\text{[B-IfFalse]} \quad \frac{e_1 \Downarrow \texttt{false} \quad e_3 \Downarrow v}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Downarrow v}$$

Two different rules are needed for handling conditional expressions. Which rule applies depends on the premises.

For the [B-IfTrue] rule, the premise states that e_1 evaluates to `true` and e_2 evaluates to some value v. If the premise holds, then the result of evaluating the expression is the value v. The structure of [B-IfFalse] is similar.

Exercise

Let's extend the language with new features. Our new language will be called BoolNum*. The valid expressions and values are shown below:

e ::=		*Expressions*
	`true`	true value
	`false`	false value
	n	natural numbers
	`if` e `then` e `else` e	conditional expressions
	`succ` e	successor
	`pred` e	predecessor
v ::=		*Values*
	`true`	true value
	`false`	false value
	n	natural numbers

In addition to numbers, our expanded language has the ability to increment a number (`succ` e) and decrement a number (`pred` e).

Using LaTeX, write the evaluation rules for these additional constructs. If necessary, revise any existing rules.

Then, add these new constructs to the Haskell interpreter that we have been working with. Note that there are a few reasonable choices on how these features should work. Whatever you choose is fine, as long as your semantics match your interpreter.