

# Operational Semantics for BoolInt\* Language

Yuri Tatishchev  
San José State University  
iurii.tatishchev@sjsu.edu

February 8, 2026

## Abstract

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

BoolInt\* 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 BoolInt\* 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 BoolInt\* are **true**, **false**, and  $i$ .

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 BoolInt\* language. Of course, there are additional possible rules.

Figure 1: The BoolInt\* language

$e ::=$	<b>true</b> <b>false</b> $i$ <b>if <math>e</math> then <math>e</math> else <math>e</math></b> <b>succ <math>e</math></b> <b>pred <math>e</math></b>	<i>Expressions</i> true value false value integers conditional expressions successor predecessor
$v ::=$	<b>true</b> <b>false</b> $i$	<i>Values</i> true value false value integers

Figure 2: Big-step semantics for BoolInt\*

**Evaluation Rules:**

$$\boxed{e \Downarrow v}$$

[B-VALUE]

$$\frac{}{v \Downarrow v}$$

[B-IFTRUE]

$$\frac{\begin{array}{c} e_1 \Downarrow \mathbf{true} \\ e_2 \Downarrow v \end{array}}{\mathbf{if } e_1 \mathbf{ then } e_2 \mathbf{ else } e_3 \Downarrow v}$$

[B-IFFALSE]

$$\frac{\begin{array}{c} e_1 \Downarrow \mathbf{false} \\ e_3 \Downarrow v \end{array}}{\mathbf{if } e_1 \mathbf{ then } e_2 \mathbf{ else } e_3 \Downarrow v}$$

[B-SUCC]

$$\frac{e \Downarrow i}{\mathbf{succ } e \Downarrow i + 1}$$

[B-PRED]

$$\frac{e \Downarrow i}{\mathbf{pred } e \Downarrow i - 1}$$

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**.

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.

We also have rules for handling the integer operations. The [B-SUCC] rule states that if the operand  $e$  evaluates to an integer  $i$ , then the expression **succ**  $e$  evaluates to  $i + 1$ . Similarly, [B-PRED] evaluates **pred**  $e$  to  $i - 1$  if  $e$  evaluates to  $i$ .