# Operational Semantics for BoolInt* Language

Kushagra Bainsla
San José State University
kushagra.bainsla@sjsu.edu

February 8, 2026

**Abstract**

In this paper, we will provide a review of the big-step operational semantics for the BoolInt* language, an extension of Bool* with integer values and arithmetic operations (succ and pred).

BoolInt* is a very minimal language that allows us to experiment with operational semantics, now extended to include integer values and arithmetic operations.

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`, an integer $i$, or the conditional expression `if` $e$ `then` $e$ `else` $e$, or the arithmetic expressions `succ` $e$ and `pred` $e$. Note that conditional and arithmetic expressions have a recursive structure, with 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 integers $i$.

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 1: The BoolInt* language

| | | |
|---|---|---|
| $e ::=$ | | *Expressions* |
| | `true` | true value |
| | `false` | false value |
| | $i$ | integers |
| | `if` $e$ `then` $e$ `else` $e$ | conditional expressions |
| | `succ` $e$ | successor |
| | `pred` $e$ | predecessor |
| $v ::=$ | | *Values* |
| | `true` | true value |
| | `false` | false value |
| | $i$ | integers |

Figure 2: Big-step semantics for BoolInt*

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

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

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

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

[B-Succ] $$\frac{e \Downarrow i}{\texttt{succ } e \Downarrow i+1}$$

[B-Pred] $$\frac{e \Downarrow i}{\texttt{pred } e \Downarrow i-1}$$

Figure 2 shows the big-step evaluation rules for the BoolInt* 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`, `false`, and any integer $i$ evaluate to themselves.

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.

The [B-Succ] rule states that if $e$ evaluates to an integer $i$, then `succ` $e$ evaluates to $i + 1$. The [B-Pred] rule states that if $e$ evaluates to an integer $i$, then `pred` $e$ evaluates to $i - 1$.