

Structural Operational Semantics

Martin Bråthen

December 4, 2019

1 SOS

I have no rules for declaration of new functions and variable, and not for function calls either. For variable declarations, I will use the [assign] rule. And I will pretend the return from function calls are just the expression that they return.

m_i	Store
e_i	Expression
n_i	Integer
b_i	Boolean
v_i	Boolean or Integer
s_i	Statement

Table 1: Notations

1.1 Arithmetic

$\frac{\langle m, e_0 \rangle \longrightarrow n_0, \langle m, e_1 \rangle \longrightarrow n_1}{\langle m, e_0 + e_1 \rangle \longrightarrow \langle m, n_0 + n_1 \rangle}$	[add]
$\frac{\langle m, e_0 \rangle \longrightarrow n_0, \langle m, e_1 \rangle \longrightarrow n_1}{\langle m, e_0 * e_1 \rangle \longrightarrow \langle m, n_0 * n_1 \rangle}$	[mul]
$\frac{\langle m, e_0 \rangle \longrightarrow n_0, \langle m, e_1 \rangle \longrightarrow n_1}{\langle m, e_0 / e_1 \rangle \longrightarrow \langle m, n_0 / n_1 \rangle}$	[div]
$\frac{\langle m, e_0 \rangle \longrightarrow n_0, \langle m, e_1 \rangle \longrightarrow n_1}{\langle m, e_0 \% e_1 \rangle \longrightarrow \langle m, n_0 \% n_1 \rangle}$	[mod]
$\frac{\langle m, e \rangle \longrightarrow n}{\langle m, -e \rangle \longrightarrow \langle m, -1 * n \rangle}$	[neg]

1.2 Boolean

$$\frac{\langle m, e_0 \rangle \longrightarrow false, \langle m, e_1 \rangle \longrightarrow false}{\langle m, e_0 \text{ AND } e_1 \rangle \longrightarrow \langle m, false \rangle} \quad [\text{and1}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow true, \langle m, e_1 \rangle \longrightarrow false}{\langle m, e_0 \text{ AND } e_1 \rangle \longrightarrow \langle m, false \rangle} \quad [\text{and2}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow false, \langle m, e_1 \rangle \longrightarrow true}{\langle m, e_0 \text{ AND } e_1 \rangle \longrightarrow \langle m, false \rangle} \quad [\text{and3}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow true, \langle m, e_1 \rangle \longrightarrow true}{\langle m, e_0 \text{ AND } e_1 \rangle \longrightarrow \langle m, true \rangle} \quad [\text{and4}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow b}{\langle m, NOT\ e \rangle \longrightarrow \langle m, NOT\ b \rangle} \quad [\text{not}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow false, \langle m, e_1 \rangle \longrightarrow false}{\langle m, e_0 \text{ OR } e_1 \rangle \longrightarrow \langle m, false \rangle} \quad [\text{or1}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow true, \langle m, e_1 \rangle \longrightarrow false}{\langle m, e_0 \text{ OR } e_1 \rangle \longrightarrow \langle m, true \rangle} \quad [\text{or2}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow false, \langle m, e_1 \rangle \longrightarrow true}{\langle m, e_0 \text{ OR } e_1 \rangle \longrightarrow \langle m, true \rangle} \quad [\text{or3}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow true, \langle m, e_1 \rangle \longrightarrow true}{\langle m, e_0 \text{ OR } e_1 \rangle \longrightarrow \langle m, true \rangle} \quad [\text{or4}]$$

1.3 Comparison

$$\frac{\langle m, e_0 \rangle \longrightarrow v_0, \langle m, e_1 \rangle \longrightarrow v_1}{\langle m, e_0 == e_1 \rangle \longrightarrow \langle m, v_0 == v_1 \rangle} \quad [\text{eq}]$$

$$\frac{\langle m, e_0 \rangle \longrightarrow v_0, \langle m, e_1 \rangle \longrightarrow v_1}{\langle m, e_0 != e_1 \rangle \longrightarrow \langle m, v_0 != v_1 \rangle} \quad [\text{neq}]$$

1.4 Statements

$$\frac{\langle m, e \rangle \longrightarrow v}{\langle m, \text{assign}(x, e) \rangle \longrightarrow \langle m[x \mapsto v], \text{skip} \rangle} \quad [\text{assign}]$$

$$\frac{\langle m, e \rangle \longrightarrow true}{\langle m, \text{if}(e, s_{\text{then}}, s_{\text{else}}) \rangle \longrightarrow \langle m, s_{\text{then}} \rangle} \quad [\text{if1}]$$

$$\frac{\langle m, e \rangle \longrightarrow false}{\langle m, \text{if}(e, s_{\text{then}}, s_{\text{else}}) \rangle \longrightarrow \langle m, s_{\text{else}} \rangle} \quad [\text{if2}]$$

$$\frac{}{\langle m, \text{seq}(\text{skip}, s) \rangle \longrightarrow \langle m, s \rangle} \quad [\text{seq1}]$$

$$\begin{array}{c}
\frac{\langle m, s_0 \rangle \longrightarrow \langle m', s'_0 \rangle}{\langle m, \text{seq}(s_0, s_1) \rangle \longrightarrow \langle m', \text{seq}(s'_0, s_1) \rangle} \quad [\text{seq2}] \\
\\
\frac{\langle m, e \rangle \longrightarrow b}{\langle m, \text{while}(e, s) \rangle \longrightarrow \langle m, \text{if}(e, \text{seq}(s, \text{while}(e, s)), \text{skip}) \rangle} \quad [\text{while}] \\
\\
\frac{\langle m, e \rangle \longrightarrow v}{\langle m, \text{return}(e) \rangle \longrightarrow \langle m', \text{return}(v) \rangle} \quad [\text{return}]
\end{array}$$

1.5 Variable reference

$$\overline{\langle m, x \rangle \longrightarrow \langle m, v \rangle} \quad [\text{var}]$$

2 Interpretation of example

2.1 Code

```

fn f2(x: i32, y: i32) -> i32 {
    return x*y
}
fn f1() -> i32 {
    let a : i32 = f2(5,3);
    let b : i32 = 0;
    while b != 10 {
        b = b + 1;
    }
    if true && true {
        a = a + 3;
    } else {
        a = a + 5;
    }
    return a + b;
}

```

2.2 Interpretation

We start interpretation of `f1()`, which can be thought of as the main function.

- `let a : i32 = f2(5,3);`

Pretend we have: `a = 5*3`; and use rule `[assign]`; Right hand expression has to evaluate into either a boolean or interger, rule `[mul]` can be applied since both operands are integer values, and the result is the product `5*3 = 15` which is an integer value. The store is then updated so that variable `a` gets the value 15.

- `let b : i32 = 0;`

This is very similar to the interpretation of the line above. `b` gets the value 0.

- `while b != 10 {
 b = b + 1;
}`

Rule [while] states that the expression `b != 10` has to evaluate to a boolean value. Rule [neq] requires that both operands evaluate to a value, 10 is already an integer value, and `b` is evaluated into 0 using rule [var]. Rule [neq] then gives `0 != 10` which evaluates into `true`. We then have to evaluate `if(true, seq(s, while(e, s)), skip)` with rule [if1] which determines that `seq(s, while(e, s))` is to be evaluated, which is done by rule [seq2]. It is then determined that `s` should be evaluated followed by `while(e, s)`. `s` is in this case `seq(b = b + 1, skip)` which evaluates into `b = b + 1` which in turn is evaluated following rule [assign], rule [add] and rule [var]. Rule [add] will evaluate `0 + 1` into 1. `b` is therefore incremented by 1. `seq(skip, while(e, s))` will then be evaluated by rule [seq1] into `while(e, s)`, which will restart the whole process. The loop will exit when `b` is equal to 10.

- `if true && true {
 a = a + 3;
} else {
 a = a + 5;
}`

Rule [if1] requires that the expression `true && true` is evaluated by rule [and4] into `true`, which it is. This means that `a = a + 3;` would be evaluated, and `a` would get the value of 18.

- `return a + b;`

Rule [return] states that the expression `a + b` should be evaluated into a value, in this case by rule [add] and rule [var], which results in 28, which is then returned from the function.

3 Requirements

- Function definitions
- Commands (let, assignment, if then (else), while)
- Expressions (including function calls)
- Primitive types (boolean, i32) and their literals

- Explicit types everywhere
- Explicit return(s)
- Your interpreter should be able to correctly execute programs according to your SOS.
- Your interpreter should panic (with an appropriate error message) when encountering an evaluation error (e.g., `1 + false`)

All of the above requirements have been met. I implemented everything myself.