Implementing Objective B

Programming Language Implementation and Formalisation (Spring 2024)

The Cobra Cooperation has been receiving numerous error reports regarding their imperative dynamically typed programming language, Boa. The primary issue seems to stem from a lack of proper code structure and absence of a type system, resulting in an increased number of errors. To address these concerns, the Cobra Cooperation has initiated the development of a new language, Objective B, with the aim of resolving these issues and providing a more efficient and error-free programming experience. In this assignment, you will explore the design and features of Objective B and analyze the potential impact it may have on the programming community. Additionally, you will have the opportunity to propose potential improvements or additional features for the language. Let's get started!

SYNTAX

The goal of this assignment is to gain experience in implementing a programming language from its formalisation. You will be implementing a parser and interpreter for the Objective B programming language, for which you will find the concrete syntax is defined by the BNF in Figure 1.

$f, x \in$ **Name**	(Well-formed identifiers).
$i, j \in \mathbb{Z}$	(Integers)
$\mathbb{P} := \mathrm{p}^*$	(Programs)
$\mathrm{p} := \tau\ f\ (\tau_i\ x_i)\ \{\mathrm{s}\}$	(Procedures)
$\tau :=$ `integer`	(Primitive Types)
$\mid$ `boolean`	
$\mid$ `void`	
$\mathrm{v} :=$ `true` $\mid$ `false` $\mid i \mid$ `null`	(Simple values)
$\mathrm{s} := \{\mathrm{s}\}$	(Statements)
$\mid ((\tau)^?\ x =)^?\ \mathrm{e}$	
$\mid$ `if` e `then` s_1 `else` s_2	
$\mid$ `while` e s	
$\mid$ `return` e	
$\mid \mathrm{s}_1$ `;` s_2	
$\mid \varepsilon$	(The empty statement)
$\mathrm{e} :=$ `(` e `)`	(Expressions)
$\mid f$ `(` e_i `)`	
$\mid \mathrm{e}_1 \oplus \mathrm{e}_2$	
$\mid$ `!` e	
$\mid x$	
$\mid$ v	
$\oplus :=$ `+` $\mid$ `-` $\mid$ `*` $\mid$ `/` $\mid$ `==` $\mid$ `<` $\mid$ `&&` $\mid$ `||`	(Binary operators)

Figure 1. The syntax of Objective B. The star ($\odot^*$) denotes a sequence containing 0 or more occurences. and the questionmark ($\odot^?$) denotes choice (zero or one), and the subscript ($\odot_i$) denotes a comma separated sequence containing zero or more indexable entities ($\odot_0, \odot_1, \ldots, \odot_n$).

A program $\mathbb{P}$, consists of zero or more procedures, and a procedure is defined by specifying its return type and name, followed by a list of arguments and a statement. We will cover types and statements in more detail later, but first, here are a couple of example procedures

```
boolean isEven(integer n) {
  if   n == 0
  then return true
  else return isOdd(n - 1)
}

boolean isOdd(integer n) {
  if   n == 0
  then return false
  else return isEven (n - 1)
}

boolean isPrime(integer n){
  integer i = 2 ;
  while (i < n) {
    if (n / i) * i == n
    then return false
    else i = i + 1
  } ;
  return true
}
```

As you may have noticed, the main part of a procedure is its statement. To avoid syntactic ambiguity, statements can be grouped by curly braces. For instance, the statement

```
{ while e s1 } ; s2
```

will run the statement `s1` until `e` holds, and then it will run `s2` once. Whereas

```
while e { s1 ; s2 }
```

will run the statement `s1 ; s2` until `e` holds. Furthermore a statement can be empty, and this is just to accomodate any leading or trailing statement composiiton operators s_1 `;` s_2.

SEMANTICS

A Objective B program must contain a procedure called `main`, and the semantics of running a program, is to call `main` with its supplied arguments. It is up to you to decide what those parameters should be, but you must state your choice clearly in the report.

A procedure runs in the context of a source program π, and a state σ, containing the variable bindings currently in scope.

There are two evaluation judgements: An evaluation judgement for expressions $\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v} \rangle$ in which an expression evaluates to a value. And, a judgement for statements, $\langle \pi, \sigma_1, \mathrm{s} \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle$, in which a statement computes a value, and possibly changes the state. You can find the judgements for expressions and statements in Figures 2 and 3 respectively.

YOUR TASK

The board of Cobra Cooperation has requested that you write a prototype interpreter for Objective B. That is, your task is to initiate a `stack` project, it could be called `objective-b`, and design and implement a data structure and parser for the abstract syntax of Objective B. Then you are asked to write an interpreter for Objective B. The choice is yours, and you can take inspiration from `Boa` language. You should hand in a report together with your code, describing the choices you made.

The board has a list of questions that you should answer in the report, and a list of optional questions below.

$\boxed{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v} \rangle}$

$$\text{Variable} : \frac{}{\langle \pi, \sigma[x \mapsto v], x \rangle \downarrow \langle v \rangle} \qquad \text{Value} : \frac{}{\langle \pi, \sigma, \mathrm{v} \rangle \downarrow \langle \mathrm{v} \rangle}$$

$$\text{Call} : \frac{\langle \pi, \sigma, \mathrm{e}_i \rangle \downarrow \langle \mathrm{v}_i \rangle \quad \langle \pi, [x_i \mapsto \mathrm{v}_i], \mathrm{s} \rangle \downarrow \langle \mathrm{v}, \sigma' \rangle}{\langle \pi[\tau f(\tau_i, x_i)\{\mathrm{s}\}], \sigma, f\ (\mathrm{e}_i)\ \rangle \downarrow \langle \mathrm{v} \rangle}$$

$$\text{Not-True} : \frac{\langle \pi, \sigma, \mathrm{e}_1 \rangle \downarrow \langle \texttt{true} \rangle}{\langle \pi, \sigma, \texttt{!}\,\mathrm{e}_1 \rangle \downarrow \langle \texttt{false} \rangle} \qquad \text{Not-False} : \frac{\langle \pi, \sigma, \mathrm{e}_1 \rangle \downarrow \langle \texttt{false} \rangle}{\langle \pi, \sigma, \texttt{!}\,\mathrm{e}_1 \rangle \downarrow \langle \texttt{true} \rangle}$$

$$\text{Operation} : \frac{\langle \pi, \sigma, \mathrm{e}_1 \rangle \downarrow \langle \mathrm{v}_1 \rangle \quad \langle \pi, \sigma, \mathrm{e}_2 \rangle \downarrow \langle \mathrm{v}_2 \rangle}{\langle \pi, \sigma, \mathrm{e}_1 \oplus \mathrm{e}_2 \rangle \downarrow \langle \mathrm{v} \rangle} (\mathrm{v}_1 \oplus \mathrm{v}_2 = \mathrm{v})$$

Figure 2

$\boxed{\langle \pi, \sigma_1, \mathrm{e} \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle}$

$$\text{Skip} : \frac{}{\langle \pi, \sigma, \varepsilon \rangle \downarrow \langle \texttt{null}, \sigma \rangle} \qquad \text{Expression} : \frac{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v} \rangle}{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v}, \sigma \rangle}$$

$$\text{AssignNew} : \frac{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v} \rangle}{\langle \pi, \sigma, \tau\ x = \mathrm{e} \rangle \downarrow \langle \mathrm{v}, \sigma[x \mapsto v] \rangle} \qquad \text{AssignOld} : \frac{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \mathrm{v} \rangle}{\langle \pi, \sigma[x \mapsto w], x = \mathrm{e} \rangle \downarrow \langle \mathrm{v}, \sigma[x \mapsto \mathrm{v}] \rangle}$$

$$\text{If-True} : \frac{\langle \pi, \sigma_1, \mathrm{e} \rangle \downarrow \langle \texttt{true} \rangle \quad \langle \pi, \sigma_1, \mathrm{s}_1 \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle}{\langle \pi, \sigma_1, \texttt{if}\ \mathrm{e}\ \texttt{then}\ \mathrm{s}_1\ \texttt{else}\ \mathrm{s}_2 \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle}$$

$$\text{If-False} : \frac{\langle \pi, \sigma_1, \mathrm{e} \rangle \downarrow \langle \texttt{false} \rangle \quad \langle \pi, \sigma_1, \mathrm{s}_2 \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle}{\langle \pi, \sigma_1, \texttt{if}\ \mathrm{e}\ \texttt{then}\ \mathrm{s}_1\ \texttt{else}\ \mathrm{s}_2 \rangle \downarrow \langle \mathrm{v}, \sigma_2 \rangle}$$

$$\text{While-True} : \frac{\langle \pi, \sigma_1, \mathrm{e} \rangle \downarrow \langle \texttt{true} \rangle \quad \langle \pi, \sigma_1, \mathrm{s} \rangle \downarrow \langle \mathrm{v}_1, \sigma_2 \rangle \quad \langle \pi, \sigma_2, \texttt{while}\ \mathrm{e}\ \mathrm{s} \rangle \downarrow \langle \mathrm{v}_2, \sigma_3 \rangle}{\langle \pi, \sigma_1, \texttt{while}\ \mathrm{e}\ \mathrm{s} \rangle \downarrow \langle \mathrm{v}_2, \sigma_3 \rangle}$$

$$\text{While-False} : \frac{\langle \pi, \sigma, \mathrm{e} \rangle \downarrow \langle \texttt{false} \rangle}{\langle \pi, \sigma, \texttt{while}\ \mathrm{e}\ \mathrm{s} \rangle \downarrow \langle \texttt{null}, \sigma \rangle} \qquad \text{Sequence} : \frac{\langle \pi, \sigma, \mathrm{s}_1 \rangle \downarrow \langle \mathrm{v}_1, \sigma_1 \rangle \quad \langle \pi, \sigma_1, \mathrm{s}_2 \rangle \downarrow \langle \mathrm{v}_2, \sigma_2 \rangle}{\langle \pi, \sigma, \mathrm{s}_1\ \texttt{;}\ \mathrm{s}_2 \rangle \downarrow \langle \mathrm{v}_2, \sigma_2 \rangle}$$

Figure 3

Questions

1. Figure 3 does not specify a semantics for the `return` statement. Suggest a judgement rule for `return` and include it in your implementation. Does your semantics for `return` affect the other judgements in Figure 3. If so, explain how your changes propagate.

2. Cobra Cooperation has not yet designed a type system for Objective B. Suggest a judgement form for typing expressions, and give two interesting example rules.

3. How would you asses the current state of your prototype? How confident are you that the prototype is correct? What needs improvement in a production-ready interpreter?

Challenges

1. `Boa` had a print expression. Extend your semantics with `print` and describe the choices you made. What happens to the evaluation rule for expressions?

2. It is customary for procedural languages to provide a means of passing values between procedures. Extend the syntax of Objective B to accommodate pointers, implement pointers in your code.

3. Extend Objective B with arrays.

4. Extend Objective B with objects. What happens to your type system?