Java Interpreter CS345H Fall 2011 Due 9:30am Nov 22, 2011

Use Java to rewrite the parser and interpreter framework that was implemented in the previous assignments. Your interpreter should have eager application semantics and use deferred substitution. Call the new language JCFWAE (conditionals, functions, with, and arithmetic expressions).

You must do this assignment solo.

Requirements

- Use a parser generator (JavaCC, ANTLR, Cup) to implement your parser for your language. Because the language is no longer implemented in Racket, we will use a more traditional syntax, which is defined below.
- In your Java code a class name can only be used to declare a class, as the superclass of another class, or after the keyword "new". Classes may not be used as types. You must define interfaces. You may not explicitly test whether an object has a specific class (with "instanceof" or reflection). The only way to distinguish an object of one class from another is to invoke a method. The net effect is that you should have very few "if/case" case statements in your program, and these should only test primitive values.
- Your Java code must be structured so that new kinds of expressions (for example, to allow creation and use of pairs of values) can be added without rewriting the entire program. To test this, you should be able to delete the file that implements the "with" expression and still have your program be able to run test cases that don't make use of "with". This is an extensibility requirement. *NOTE: the parser does not have to not work correctly if you delete a file.* To test this requirement, you will have to use a hard-coded set of calls to constructors, or else comment out a line of the parser.
- The errors you should check for are follows:
  1) typing errors involving incorrect use of a value (addition of a function [e.g. 3 + fun(x) x], calling a number [e.g. 3(a, 5)], adding a boolean [3+true], etc). However, these errors must be checked without using if statements.
  2) Incorrect number of arguments, or duplicate identifiers in an argument list or with block. These tests can be done using if-statements.

To summarize, the intent of the assignment is to use a pure form of object-oriented programming and ensure that all data is manipulated in as extensible and abstract a fashion as possible.

Features to Implement

* Conditionals
* Multi-argument `fun`
* Multi-armed `with`
* Recursive functions when defined in a with, as in the previous assignment

Syntax of JCFWAER

The syntax of the JCFWAER language with these additional features can be captured with the following EBNF. The terminal symbols are marked in quotes. The quotes are not part of the language.

```
E ::= num | "true"  | "false"
    | id
    | E "(" E "," … "," E ")"
    | E "*" E      | E "/" E
    | E "+" E      | E "-" E
    | E "=" E      | E "<" E
    | "!" E
    | E "&&" E
    | E "||" E
    | "if" E "then" E "else" E
    | "with" id "=" E ";" … ";" id "=" E "in" E
    | "fun" "(" id "," … "," id ")" E
    | "(" E ")"
```

In this grammar, the ellipsis (...) appears between an optional separator and a nonterminal, means that the nonterminal can be repeated zero or more times with the separator between each occurrence. If there is no separator, then the nonterminal must occur at least once.

An `id` is an variable identifier as in Java, and `num` is an integer. You will need to support *boolean values* in addition to numbers and closures.

The productions in the grammar are written from *highest precedence* to *lowest.* All operations are left associative. Ambiguity with "`else`" should be handled in the normal way: an "`else`" refers to the closest "`if`" statement.

The "=" operator tests equality. It is also used in bindings. Your interpreter must check for errors involving invalid use of a value during interpretation: for example, applying a number as a function, or adding a number and a closure.

There is no assignment statement, hence no need for a store.

Test files will be sent out next week.