CS 784 Programming Languages

Fall 2011 • HW3: CodeGen • HW3 slides

The primary goal of this assignment is to gain familiarity with language processing concepts and techniques using the infrastructure developed for the EOPL text.

Language AE0

```
expr   ::= term   { + term }*
term   ::= factor  { * factor }*
factor ::= var   |   ( expr )
var    ::=  i | j | k
```

A grammar for AE0, arithmetic expressions of HW3, is given above. `i, j, k, +, *, (, )` are the terminals.

```
class Test {
   static int f(int i, int j, int k) {
     return  expr;
   }
   public static void main(String[] args) {
     System.out.println(f(2,4,8));
   }
}
```

The "realm" of our AE0 is the expr in Test.java above. The environment assigns i, j, k the values 2, 4, 8 respectively.

Here is how the above can be specified in SLLGEN (Appendix B EOPL3).

```
;; see p384 EOPL3
(define expr-scanner
  '((whitespace (whitespace) skip)
    (identifier ((or "i" "j" "k")) symbol)))

;; see p385 EOPL3
(define expr-grammar
  '((expr (term (arbno "+" term)) an-expr)
    (term (factor (arbno "*" factor)) a-term)
    (factor (var) var-factor)
    (factor ("(" expr ")") expr-factor)
    (var (identifier) a-var)))

;; see p387 EOPL3
(sllgen:make-define-datatypes expr-scanner expr-grammar)

(define scan&parse
  (sllgen:make-string-parser expr-scanner expr-grammar))

;; examples
(scan&parse "i")
(scan&parse "(k + k ) * i")
(scan&parse "(i * k) + (j + i)")
(scan&parse "i + j * (((k) + j) * i)")
```

Does there exist a grammar for this language that is LL(1) and doesn't make use of special constructs like arbno? If you wish, pursue this question. Parsing issues are outside the scope of CS784. We are using SLLGEN as a "blackbox".

Run

Write a function `run` in Scheme that takes an AE0 expression and outputs the value of the expression, assuming that i, j, and k are 2, 4, and 8 respectively. Convert the concrete expression into an AST prior to evaluation.

```
>(run "((i * i) + j * (k) + j)")
40
```

Function run should not make use of compile and interpret functions described below. Our run works on the AST of AE0 and evaluates the given expression. It does not use bytecodes.

Compile

Write a function `compile` in Scheme that takes an AE0 expression and outputs a semantically equivalent list of JVM bytecodes. Reuse the code for creating the AST from above for this part.

We map i, j, and k to registers 0, 1, and 2 of JVM. A typical JVM instruction encodes information about the type and the location of operands, and the nature of operation. For example, iload_2 stands for pushing the value of register 2 on top of the stack; istore_2 stands for moving the value from the top of the stack to register 2; iadd (resp. imul) stands for popping the top two integer values from the stack, adding (resp. multiplying) them, and pushing the result on top of the stack. These details are sufficient for this HW. (For further details, refer to The Java Virtual Machine Specification.)

```
>(compile "i + j * (((k) + j) * i)")
(iload_0 iload_1 iload_2 iload_1
  iadd iload_0 imul imul iadd)
```

To verify such translations, replace **expr** with one of our arithmetic expressions in "Test.java", compile it `javac Test.java`, and reverse engineer the class file using `javap -c Test`, focusing on method `int f(int, int, int)`.

On the leftt is a Java applet. (Make sure that your browser has Java plugin enabled.) The applet translates our expressions into Java bytecodes and displays the results. Note that the generated code reflects left associativity of "+" and "*".

Interpret

Write a function `interpret` in Scheme that takes the output of `compile` above and (hopefully) returns the same result as `run` when given the same expression.

(For the runtime stack object use Scheme's set! and let constructs.)

```
>(interpret (compile "((i * i) + j * (k) + j)"))
40
```

To repeat: "interpret" should not use AST, it should just interpret the byte codes emitted by "compile".

What to hand-in?

Submit on `gandalf.cs.wright.edu` as follows:

```
% ~pmateti/CS784/turnin HW3 ReadMe.txt codegen.scm
```

Your `codegen.scm` should be a ready-to-run Scheme file and include a `test-list` and `run-all` as typically done in the downloadable code of EOPL3. You may wish to use the unit test framework (check-expect func result) form from HtDP. For all test expressions, it should check that the identity `(run "EXPR") = (interpret (compile "EXPR"))` is satisfied.
