Quiz

Compiler Construction

December 16, 2009

Last Name : _____

First Name : _____

Exercise	**Points**	**Achieved Points**
1	20	
2	20	
3	18	
4	18	
5	24	
Total	100	

Exercise 1 : Lexical Analysis (20 points)

Our goal is to create a lexical analyzer that recognizes a sequence of the following tokens using the longest match rule: <=, =>, <=>, =, ==.
It should also skip whitespace characters between the tokens. (Denote the whitespace characters by the ␣ symbol.)
As an example, consider the string <==>␣<==<=>.
It should give the token stream <=,=>,<=,=,<=>

(a) Draw a deterministic automaton which accepts the tokens above, optionally followed by white spaces. Use a different accepting state for each token.

(b) Consider the following Scala-like pseudo code. A lexer object contains an input string and the automaton you previously drew. Its nextToken method should return the next token in the input. Write the corresponding pseudo code in the nextToken method, making sure you conform to the longest match rule. The read method, the Automaton class, and the automaton object are considered given. You should not modify them. You can however use the Lexer's index variable and add state or methods to the lexer.

```
class Automaton {
  /*method consumes: returns true if the automaton can consume caracter c in its current state*/
  def consumes?(c : Char) : Bool

  /*method nextState: executes one transition of the automaton*/
  def nextState(c : Char) : Unit

  /*method isFinal: returns Some(token) if in a final state, else returns None*/
  def isFinal? : Option[Token] }

class Lexer(s : String) {
  val input = s
  var index : Int
  val automaton : Automaton //The automaton you drew in the previous question

  /*method read: returns true and updates the state of the automaton when
  it is possible to process the next input character*/
  def read : Bool = {
    if automaton.consumes?(s[index]) {
      automaton.nextState(s[index])
      index = index + 1
      true }
    else false }

  //To complete:
  def nextToken : Token = {
    while (read)
      automaton.isFinal? match { ... }
      if (...) throw LexerError
      else {...} }
}
```

Exercise 2 : Parsing (20 points)

We consider a grammar for arithmetic expressions where the multiplication sign is optional. For example, $x\ y$ denotes $x * y$.

ex ::= *ex* + *ex* | *ex* * *ex* | *ex ex* | *ex* / *ex* | (*ex*) | *ID* | *INTLITERAL*

(a) Using first and follow sets, give an example that shows that the above grammar is not LL(1).

(b) Find a LL(1) grammar recognizing the same language as the grammar above. Make sure you encode the operator priorities: multiplication and division have the same priority, and it is higher than for addition.

(c) Using your new grammar, draw the parse tree for the following expression: $3 * x - 2\ z$.

(d) Compute the first and follow sets for each symbol in your new grammar and show that your new grammar is LL(1).

Exercise 3 : Type Checking (18 points)

Consider a variant of the Tool programming language, which we call μTool, that has no support for arrays, class fields or local variables. Because of these constraints, classes in μTool contain only methods. These methods contain nothing but a return statement.
μTool supports inheritance, on the other hand, and the subtyping relation is given by:

$$\frac{}{T <: T} \qquad \frac{T_1 <: T_2 \quad T_2 <: T_3}{T_1 <: T_3} \qquad \frac{\text{class } T_1 \text{ extends } T_2 \ \{\ldots\}}{T_1 <: T_2}$$

In Tool, we allowed method overriding in subclasses if the number and the types of the arguments matched, and if the return type was identical. In μTool on the other hand, the criteria for valid overriding are the following. If class B extends class A, and the method m is defined in both classes, then:

- The number of arguments of A.m and B.m must be the same.
- The return type of B.m must be a subtype of the return type of A.m.
- For each argument of m, the type of the argument in B.m must be a subtype of the type of the argument in A.m.

(a) Overriding in μTool as we just presented it is not sound. Give an example μTool program which typechecks according to the rules we gave, but which crashes at runtime with a "method not found" exception.

(b) Find an alternative set of overriding rules that do not have the problem you identified, and such that the following program is valid.

```
object Prog { def main() : Unit = { println("Hello"); } }

class A {
  def foo(b : B) : A = {
    return b;
  }
}

class B extends A {
  def foo(a : A) : B = {
    return this;
  }
}
```

Exercise 4 : Code Generation (18 points)

Consider the following code:

```
while((x > 0) => b && c) {
  b = (x != 1);
  x = x / 2;
}
```

(a) Show a sequence of Java Virtual Machine code that you would generate for this code, taking into account the following points:

- The operation "v => u" translates as "(!v) || u".
- The variables b and c are booleans, and are stored at local positions 1 and 2 respectively.
- The variable x is an integer stored at local position 3.
- The stack should be empty before and after the loop.

If you wish, you can use Cafebabe abstract bytecodes (as in the project).

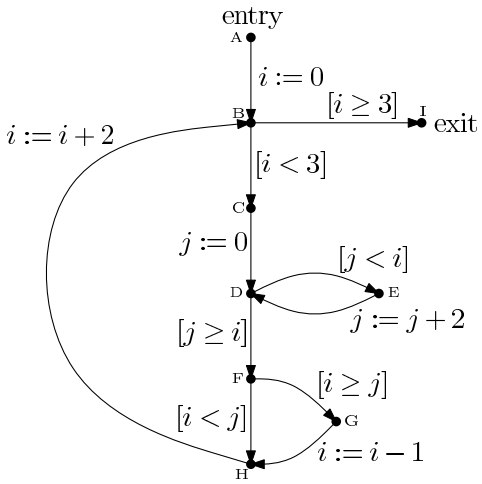
Use symbolic labels to denote the targets of jumps.

Here are some (abstract) bytecodes that may be useful (we do not claim that all of them are useful and we do not claim that they are sufficient):

```
ILoad(slot : Int)
IStore(slot : Int)
Ldc(value : Int)
IADD, IAND, IOR, IDIV, IMUL
If_ICmpGt(target : String)
Goto(target : String)
```

Exercise 5 : Data-Flow Analysis (24 points)

Consider the following control flow graph:



(a) Write down a program (fragment) that this graph could represent.

(b) We want to run an analysis that tracks the sign of the variables. We will use the following abstract domain: $L = \{\top, \bot, 0, \ominus, \oplus\}$. We assume that the runtime values are unbounded, and the concretization function is as follows:

$$\begin{aligned}\gamma : L &\to \mathbb{Z} \\ \bot &\mapsto \emptyset \\ \ominus &\mapsto \{-\infty, \ldots, -2, -1, 0\} \\ 0 &\mapsto \{0\} \\ \oplus &\mapsto \{0, 1, 2, \ldots, \infty\} \\ \top &\mapsto \mathbb{Z}\end{aligned}$$

Draw the diagram of the lattice $\sqsubseteq$ relation for the abstract domain L (for one variable). Use this diagram to fill the following table by putting **true** in each entry for which $x \sqsubseteq y$ holds, and **false** in the table entries where $x \sqsubseteq y$ does not hold. We have correctly filled one element of the table for you.

$x \setminus y$	$\bot$	0	$\ominus$	$\oplus$	$\top$
$\bot$					
0			**true**		
$\ominus$					
$\oplus$					
$\top$					

(c) Since our program deals with two variables, our abstract domain is $L \times L$, where the first member of the pair represents the abstract value for i and the second the one for j. Consider the transfer function partially given by:

edge label	$L \times L$	$L \times L$
$i := 0$	(l_i, l_j)	$(0, l_j)$
$i := i + cst$	$(0, l_j)$	$(\oplus, l_j)$
$i := i + cst$	$(\ominus, l_j)$	$(\top, l_j)$
$i := i + cst$	$(\oplus, l_j)$	$(\oplus, l_j)$
$i := i + cst$	$(\top, l_j)$	$(\top, l_j)$
$i := i - cst$	$(0, l_j)$	$(\ominus, l_j)$
$i := i - cst$	$(\ominus, l_j)$	$(\ominus, l_j)$
$i := i - cst$	$(\oplus, l_j)$	$(\top, l_j)$
$i := i - cst$	$(\top, l_j)$	$(\top, l_j)$
$[i < cst]$	$(0, l_j)$	$(0, l_j)$
$[i < cst]$	$(\ominus, l_j)$	$(\ominus, l_j)$

edge label	$L \times L$	$L \times L$
$[i < cst]$	$(\oplus, l_j)$	$(\oplus, l_j)$
$[i < cst]$	$(\top, l_j)$	$(\top, l_j)$
$[i \geq cst]$	$(0, l_j)$	$(\bot, \bot)$
$[i \geq cst]$	$(\ominus, l_j)$	$(\bot, \bot)$
$[i \geq cst]$	$(\oplus, l_j)$	$(\oplus, l_j)$
$[i \geq cst]$	$(\top, l_j)$	$(\oplus, l_j)$
$[i < j]$	$(0, 0)$	$(\bot, \bot)$
$[i < j]$	$(\oplus, 0)$	$(\bot, \bot)$
$[i < j]$	$(\top, 0)$	$(\ominus, 0)$
$[i < j]$	$(0, \ominus)$	$(\bot, \bot)$
$[i < j]$	$(\oplus, \ominus)$	$(\bot, \bot)$
$[i < j]$	$(\top, \ominus)$	$(\ominus, \ominus)$

edge label	$L \times L$	$L \times L$
$[i < j]$	$(0, \top)$	$(0, \oplus)$
$[i < j]$	$(\oplus, \top)$	$(\oplus, \oplus)$
$[i < j]$	(l_i, l_j)	(l_i, l_j)
$[i \geq j]$	$(\ominus, 0)$	$(0, 0)$
$[i \geq j]$	$(\top, 0)$	$(\oplus, 0)$
$[i \geq j]$	$(0, \oplus)$	$(0, 0)$
$[i \geq j]$	$(\ominus, \oplus)$	$(0, 0)$
$[i \geq j]$	$(\top, \oplus)$	$(\oplus, \oplus)$
$[i \geq j]$	$(0, \top)$	$(0, \ominus)$
$[i \geq j]$	$(\ominus, \top)$	$(\ominus, \ominus)$
$[i \geq j]$	(l_i, l_j)	(l_i, l_j)

Here *cst* denotes a strictly positive integer literal (ie. $cst > 0$). The rules for assignments to j are similar to the ones for i.

Using this transfer function, apply the sign analysis to the control flow graph until you reach a fixpoint. Assume that the initial values for i and j at point A are -1 and 1 respectively, and start with $(\bot, \bot)$ everywhere else. Give for each program point the corresponding final value in the abstract domain $L \times L$.

(d) Finally, give the transfer function for `i := i * j`.

END OF QUESTIONS

NO QUESTIONS ON THIS PAGE