COMP 317: Semantics of Programming Languages

Problem Sheet 1

UNIVERSITY OF LIVERPOOL

Attempt these questions *before* your tutorial session.

---

1. Give a BNF definition of decimal numerals (i.e., numbers in "base 10"), and give a semantics for these numerals, along the lines of the binary-numerals example from the lectures and first hand-out. How could you avoid having "leading zeros"? (I.e., how could you define the syntax so that `317` is a valid numeral, but `0317` is not?)

2. Suppose we had defined the syntax of binary numerals as follows

   ```
   <Digit>    ::=  0 | 1
   <Numeral>  ::=  <Digit> | <Digit><Numeral>
   ```

   What changes would be needed to define the semantics of numerals?

3. An **alphanumeric character** is either a digit (0,1,...,9) or a letter (a,b,...,z). In many programming languages, variable names can be any string of alphanumeric characters that doesn't begin with a digit. Give a BNF description of a syntactic class `<Var>` of such variable names.

4. Give a BNF definition of Java method signatures; i.e., the method declarations that can be given in an interface, for example:

   ```
   public void myMethod(int i);

   String concatenate(String s1, String s2);
   ```

   You may assume that all parameters either have type `int` or `String`, and that return-types are either `int`, `String` or `void` (why would it be more complicated to allow any type or class name to be used?). (Note that you may want to use your solution to the previous question to specify permissible method names.)

5. Define (inductively) an operation `increment` that takes a binary numeral as input, and returns as output the binary numeral that is one greater than the input numeral. For example, we should have `increment(100) = 101` and `increment(111) = 1000`. Define also addition on binary numerals.

6. Give a denotational semantics for Boolean expressions by defining inductively the function [[*T*]] that maps states to truth values (define this function by induction on the structure of the Boolean expression *T*).

7. Simplify the following as far as possible:
   1. [[`2 * ('x + 13)`]](init)
   2. [[`5 * (1 + 'x + 'y)`]](*S*)
   3. [[`'x < 'x + 1`]](*S*)

8. Show that for any expressions *E1*, *E2* and *E3*, and for any Store *S*, the following are true:
    1. [[`E1 + (E2 + E3)`]](*S*) = [[`(E1 + E2) + E3`]](*S*)
    2. [[`E1 * (E2 + E3)`]](*S*) = [[`E1 * E2 + E1 * E3`]](*S*)

Grant Malcolm