Exercises on Compilers

Jordi Cortadella

February 7, 2022

Lexical Analysis

---

1. Define a regular expression for the floating point numbers. Examples:

   ```
   3.1416    -3e4    +1.0e-5    .567e+8
   ```

   In case of using a `'.'`, a decimal digit is always requested.

   Solution:
   Let us define some previous expressions:

   ```
   DIGIT = '0'..'9'
   SIGN = '+' | '-'
   DOT = '.'
   EXPONENT = ('e' | 'E') SIGN? DIGIT+
   ```

   Let us call `FLOAT` the regular expression for the floating point numbers. Then,

   ```
   FLOAT = MANTISA_WITH_DOT EXPONENT? | MANTISSA_WITHOUT_DOT EXPONENT
   MANTISA_WITH_DOT = SIGN? DIGIT* DOT DIGIT+
   MANTISA_WITHOUT_DOT = SIGN? DIGIT+
   ```

   There are other valid solutions. In any case, the crucial question that one must ask for any possible solution is: does it admit the empty string? If the solution admits the empty string, then it is incorrect. Notice that the proposed solution does not admit the empty string.

---

2. Define a regular expression for all strings from the alphabet $\{a, b, c\}$ in which the first appearance of symbol b is always preceeded by at least one appearance of symbol a. Try the expression to be as simple as possible.

   Solution:
   A possible solution is as follows: $c^*\,(a(a|b|c)^*)?$

---

3. Define a regular expression for all strings of lowercase letters that contain five vowels in order (each vowel must appear exactly once). Example: `zfaehipojksuj`.

   Solution:
   Let `STRING` be the regular expression that meets the specification. A simple solution is as follows:

   ```
   CONS = 'b' | 'c' | 'd' | 'f' | ... | 'z'
   STRING = CONS* 'a' CONS* 'e' CONS* 'i' CONS* 'o' CONS* 'u' CONS*
   ```

---

4. Define a regular expression for all strings in which the letter are in ascending lexicographical order. Examples: `afhmnqsyz`, `abcdz`, `dgky`, ... However, the string `bdeaz` does not belong to the language.

   Solution:
   This simple regular expression meets the specification:

   ```
   'a'? 'b'? 'c'? 'd'? ... 'z'?
   ```

5. Define a regular expression for all strings that represent financial quantities in American notation. Examples:

```
$**2,345.67    $12,452,183.16    $****12    $**0
```

These have a leading dollar sign (`$`), an optional string of asterisks (`*` - used on checks to discourage a fraud), a string of decimal digits, and an optional fractional part consisting of a decimal point (.) and two decimal digits. The string of digits to the left of the decimal point may consist of a single zero (`0`). Otherwise it must not start with a zero. If there are more than three digits to the left of the decimal point, groups of three (counting from the right) must be separated by commas (`,`).

Solution:
Let `QUANTITY` be the regular expression that meets the specification.

```
DIGIT = '0'..'9'
NZ_DIGIT = '1'..'9'
DIGITS3 = ',' DIGIT DIGIT DIGIT
INTEGER = '0' | NZ_DIGIT DIGIT? DIGIT? DIGITS3*
FRACTIONAL = '.' DIGIT DIGIT
QUANTITY = '\$' '*'* INTEGER FRACTIONAL?
```

6. Define a regular expression for all strings that represent inexact constants in Scheme. Scheme allows real numbers to be explicitly inexact(imprecise). A programmer who wants to express all constants using the same number of characters can use sharp signs (`#`) in place of any lower-significance digits whose values are not known. A base-ten constant without exponent consists of one or more digits followed by zero of more sharp signs. An optional decimal point can be placed at the beginning, the end, or anywhere in between. Examples:

```
35##    35#.##    356.3##    .35##    35.    35#.
```

Solution:
Let `CONSTANT` be the regular expression that meets the specification.

```
DIGIT = '0'..'9'
HASH = '#'
DOT = '.'

CONSTANT = DIGIT* DOT? DIGIT+ HASH*  |  DIGIT+ HASH* DOT? HASH*
```

Other solution might be possible, but not a solution that accepts the empty string. Notice that the proposed solution always has one digit at least.