Formal Languages and Compilers

23 January 2020

Using the JFLEX lexer generator and the CUP parser generator, realize a JAVA program capable of recognizing and executing the programming language described in the following.

Input language

The input file is composed of two sections: *header* and *execution* sections, separated by means of the sequence of characters "`####`". Comments are possible, and they are delimited by the starting sequence "`[[-`" and by the ending sequence "`-]]`".

Header section: lexicon

The *header* section can contain 3 types of tokens, each terminated with the character ";":

- <`tok1`>: is composed of 3, 9 or 27 unsigned hexadecimal numbers. Each hexadecimal number is composed of 4 or 7 characters. The hexadecimal numbers are separated by the characters "*", "$", or "%".
- <`tok2`>: starts with the character "?", followed by an integer and odd number between -37 and 2425 or by a word composed by an even number, at least 4, of uppercase letters (e.g., `EXAM`). This first part of the token can be optionally followed by a word composed of characters `+` or `-`, without consecutive equal symbols.
- <`tok3`>: starts with the character "`$`", followed by a date in the format "<month>/DD/YYYY" (where <month> can be "Jan.", "Feb.", "Mar.", "Apr."), optionally followed by the character ":" and a hour in the format "HH:MM" between 08:10 and 18:32. The date must be between "Jan./09/2020" and "Apr./23/2020". Remember that the month of February has 29 days.

Header section: grammar

In the *header section* tokens can appear in any order. In addition, <tok1> can appear **0 or more times**, <tok2> must appear **exactly 2 times**, and <tok3> must appear **exactly 1 time**.

Execution section: grammar and semantic

The *execution section* starts with the `INIT` instruction followed by a list of <commands>. The list can be possibly **empty** or with an **even** number of elements.

The `INIT` instruction is the word `INIT` followed by a <weight>, a comma ",", a <volume> and a semicolon ";". A <weight> is the word `WEIGHT`, an <exp> (see later), and a <w_unit> (i.e., "g" grams or "kg" kilograms, where 1 kg=1000 g). A <volume> is the word `VOLUME`, an <exp> (see later), and a <v_unit> (i.e., "l" liters or "hl" hectoliters, where 1 hl=100 l). This instruction sets the initial values of *weigth* and *volume*, which represent the initial state, and prints them. <weight> and <volume> can appear in the instruction in reversed order, or they can be absent (both or only one). In the latter case, their default values are 100 l and 100 g.

<exp> is a typical mathematical expression with two operators, namely "+" (sum) and "*" (multiplication). Operands are *unsigned integer* numbers or a <fz>. A <fz> is the word `FZ`, a "(", the word `MIN` or `MAX`, a comma ",", a list of <exp> separated with commas ",", and a ")". In the case the first argument is `MIN`, <fz> returns the *minimum* between the list of <exp>, otherwise in the case of `MAX` it returns the *maximum*.

Commands defined by the language are:

- `OBJECT <id_obj> ATTRIBUTES {<list_of_attr>};`. <list_of_attr> is a list of <attr> separated by commas, where each attribute <attr> is an <id_attr>, a "=", and an <exp>. The two id have the same regular expression of C identifiers. This command stores all the names of the objects (<id_obj>), and the attributes (<id_attr>) with the related values (<exp>) in a global symbol table. **This symbol table is the only global data structure allowed in all the examination, and it can be written only by the OBJECT command**.
- `IF <id_obj>.<id_attr> {<list_cond>};`. <list_cond> is a not empty list of <cond>. A <cond> is a <type>, a "{", a <list_of_actions> and a "}". Two types of <cond> exist. <cond> of type *equal* is the symbol "=" followed by an <exp>. If the value of attribute identified by <id_obj>.<id_attr> is equal to <exp>, the actions listed in <list_of_actions> are executed. Instead, <cond> of type *range* are a "[", followed by $<exp_1>$ (i.e., an <exp>), a ",", $<exp_2>$ (i.e., an <exp>), a "]". If the value of the attribute identified by <id_obj>.<id_attr> is greater or equal than $<exp_1>$ and less or equal than $<exp_2>$, the actions listed in <list_of_actions> are executed. <list_of_actions> is a not empty list of <actions>, where a <action> is the word `MOD`, followed by `WEIGHT` (which modifies the current value of *weight*) or by `VOLUME` (which modifies the current value of *volume*), an <exp> and a <modifier>. If <modifier> is equal to `ADD` the value of <exp> is added to the current value of *weight* or *volume*, otherwise if <modifier> is `SUB` the value of <exp> is subtracted. The current value of *weight* and *volume* cannot be stored in global variables: **use synthesized and inherited attributes to manage the evolution of their values over time**. If needed, in your solution you can suppose that at most one <cond> is true for each `IF` command.

`INIT`, `OBJECT` and `IF` instruction/commands print the current value of *weight* and *volume*. See the example.

Goals

The translator must execute the language, and it must produce the output reported in the example.

Example

Input:

```
?-19 ;                [[- tok2 -]]
$Feb./29/2020:11:00;  [[- tok3 -]]
?LANGUAGE+-+-+ ;      [[- tok2 -]]
3A22*2fa3%123abcD ;   [[- tok1 -]]
####   [[- division between header and execution sections -]]
INIT WEIGHT 320+30 g, VOLUME 3 hl ;
[[- OUTPUT: WEIGHT=350 VOLUME=300 -]]

[[- Other possibilities -]]
[[- INIT VOLUME 2 hl , ;   WEIGHT=100 VOLUME=200 -]]
[[- INIT , ;               WEIGHT=100 VOLUME=100 -]]

OBJECT obj1 ATTRIBUTES { x=3+2*2, y=FZ(MIN,5,10)+1, z=1+FZ(MIN, 2+1, 3, FZ(MAX, 0, 1) )  };

IF obj1.y {           [[- obj1.y=6 -]]
  = FZ(MAX,1,2,3) {   [[- FALSE: obj1.y=6 NOT EQUAL TO FZ(MAX,1,2,3)=3 -]]
    MOD WEIGHT 3 ADD; }
  [ 1+1 , 3*2 ] {     [[- TRUE obj1.x=6, in interval [2, 6] -]]
    MOD VOLUME 20*FZ(MAX, 0, 2) SUB; [[- OUTPUT: WEIGHT=350 VOLUME=260 -]]
    MOD WEIGHT 50 SUB;               [[- OUTPUT: WEIGHT=300 VOLUME=260 -]]
    MOD VOLUME 10 ADD;               [[- OUTPUT: WEIGHT=350 VOLUME=270 -]]
  }
};
```

Output:

```
WEIGHT=350 VOLUME=300
WEIGHT=350 VOLUME=260
WEIGHT=300 VOLUME=260
WEIGHT=300 VOLUME=270
```

Weights: Scanner 8/30; **Grammar** 8/30; **Semantic** 10/30