June 18, 2012

Compilers

Surname, Name	
Email	

1. Specify the extended construction rule of Thompson for the operator ++, defined as repetition $2n$ times, $n \in [1, 2, ...]$. Then, draw the tree of the regular expression **r = ((a | b)$^{++}$ c)***. Finally, based on the construction of Thompson, outline the NFA recognizing the regular language of **r**.

2. Given the following grammar **G** in BNF notation,

$A \rightarrow A$ **a** $B \mid B$
$B \rightarrow A$ **b** | **a**

we ask to:

- Transform **G** into a non left-recursive grammar **G'** (equivalent to **G**);
- Generate the LL(1) parsing table of **G'**.
- Based on the parsing table, determine whether **G'** is LL(1).

3. After constructing the parsing automaton for grammar **G** defined at point 2, determine whether **G** is SLR(1).

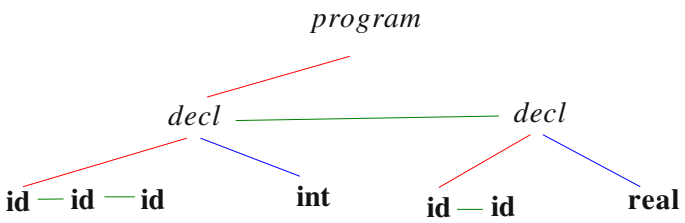
4. Codify in *Yacc* the generator of the ternary abstract trees of the language defined by the following BNF,

program → *decl-list*
decl-list → *decl* *decl-list* | *decl*
decl → *type id-list*
type → **int** | **real**
id-list → **id** , *id-list* | **id**

based on the following abstract EBNF:

program → { *decl* }$^{+}$
decl → { **id** }$^{+}$ (**int** | **real**)

```
int a, b, c;
real x, y;
```



5. Specify the (extended) attribute grammar relevant to the following BNF,

program → *stat-list*
stat-list → *stat* *stat-list* | *stat*
stat → *declaration* | *assignment* | *loop*
declaration → *type id-list*
type → **int** | **real** | **bool**
id-list → **id** , *id-list* | **id**
assignment → **id** = *expr*
expr → *expr* + *expr* | *expr* == *expr* | **id** | **intconst** | **realconst** | **boolconst**
loop → **while** *expr* **do** *stat*

based on the following semantic constraints:

- Variable names are unique;
- Mixed expressions are not allowed.

Notes:

- A symbol table is used to catalog variables by means of the following functions:
  `void insert(name, type)`
  `Type lookup(name)`: return the type of variable `name` (`INT`, `REAL`, `BOOL`) if cataloged, otherwise `NULL`;
- In case of semantic error, function `semerror()` is called, which terminates the analysis.

6. Considering the BNF given in point 5, specify the synthesis of P-code by means of an attribute grammar, where the only attribute is `code`, based on the following assumptions:

- Within nodes of the syntax tree, terminals **id**, **intconst**, **realconst**, and **boolconst** are associated with the corresponding lexical string `lexeme`;
- To generate the code, the following auxiliary functions are used:
  `Code makecode(String operation)`: builds the instruction `operation` (without arguments);
  `Code makecode2(String operation, String argument)`: builds the instruction `operation` applied to `argument`;
  `Code catcode(Code code1, Code code2, ...)`: builds the concatenation of `code1`, `code2`, ...;
  `String getoid(String name)`: returns the object identifier of variable name;
  `String newlab()`: returns a new label;
- The P-machine includes the following set of instructions:
  - `ENT`: enter program (the first instruction of the generated code);
  - `NEW` *oid*: create variable identified by object identifier *oid*;
  - `LDA` *oid*: load address of variable identified by object identifier *oid*;
  - `LOD` *oid*: load value of variable identified by object identifier *oid*;
  - `LDI` *value*: load integer *value*;
  - `LDR` *value*: load real *value*;
  - `LDB` *value*: load boolean *value*;
  - `ADD`: *addition*;
  - `EQU`: *equality*;
  - `STO`: store;
  - `LAB` *label*: create *label*;
  - `GOF` *label*: conditional (to false) jump;
  - `GOT` *label*: unconditional jump;
  - `HLT`: halt program (the last instruction of the generated code).