June 18, 2013

Compilers

Surname, Name	
Student identifier	

1. Specify the extended construction rule of Thompson for the unary operator `#`, defined as repetition of an odd number *n* times, $n \in [1, 3, 5, ...]$. 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. Codify the recursive-descent parser of the language defined by the following EBNF, also checking that phrases end with an `EOF` (end-of-file).

program → {*stat* **;**}
stat → *def-stat* | *assign-stat* | *loop-stat*
def-stat → **def id** {**, id**} **as** *type*
type → **integer** | **string** | **array [intconst .. intconst] of** *type*
assign-stat → **id** = *const*
const → **intconst** | **strconst** | *array-constructor*
array-constructor → **arr**[*const* {, *const*}**]**
loop-stat → **for id from intconst to intconst do** $\{$*stat*$\}^{+}$ **end**

3. Outline the LR(1) parsing table relevant to the following BNF.

A → *A* **a** *B* | **ε**
B → **b** | **ε**

Then, trace the LR(1) parsing of phrase **a b**. Finally, draw the corresponding syntax tree based on the traced parsing actions.

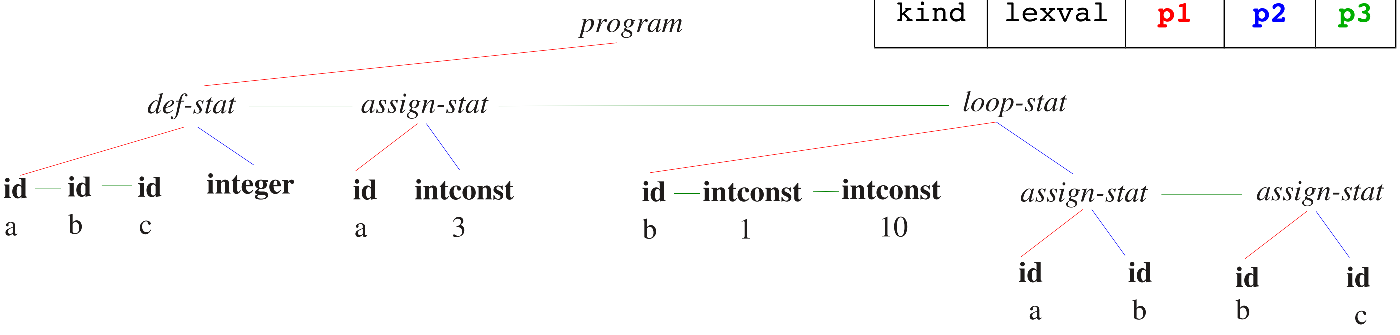
4. Codify in *Yacc* the generator of the ternary abstract trees based on the following BNF and structures:

program → *stat-list* | **ε**
stat-list → *stat* **;** *stat-list* | *stat* **;**
stat → *def-stat* | *assign-stat* | *loop-stat*
def-stat → **def** *id-list* **as** *type*
id-list → **id,** *id-list* | **id**
type → **integer** | **string**
assign-stat → **id** = *const*
const → **intconst** | **strconst**
loop-stat → **for id from intconst to intconst do** *stat-list* **end**

```
def a, b, c: integer;
a = 3;
for b from 1 to 10 do
  a = b;
  b = c;
end;
```

Node structure

kind	lexval	p1	p2	p3



5. Specify the (extended) attribute grammar relevant to the BNF defined in point **4**, based on the following semantic constraints:

- Variable names are unique;
- Referenced variables shall exist;
- In loop, the counting variable is of type integer;
- In the range [*n* .. *m*] of a loop, *m* > *n*;
- Variables are assigned with constants of the same type.

Notes:

- Lexical values of terminals are `ival` (integer) and `sval` (string);
- A symbol table is used to catalog variables by means of the following functions:
  `void insert(name, type)`: insert variable `name` with `type`;
  `Type lookup(name)`: returns the type of variable `name` (`INT`, `STR`) if cataloged, otherwise `NULL`;
- In case of semantic error, function `semerror(string msg)` is called, which prints a pertinent error message `msg`, and then terminates the analysis.

6. With reference to the BNF given in point **4**, and the corresponding topology of the abstract tree, codify a procedure of P-code generation for a virtual machine involving the following set of instructions:

- `NEW` *id*: allocate variable named *id*;
- `LDA` *id*: load address of variable named *id*;
- `LOD` *id*: load value of variable named *id*;
- `LDI` *value*: load integer *value*;
- `LDS` *value*: load string *value*;
- `ADD`: addition;
- `SUB`: subtraction;
- `MUL`: multiplication;
- `DIV`: division;
- `STO`: store;
- `LAB` *label*: create *label*;
- `EQU`: equality;
- `LTH`: less than;
- `GTH`: greater than;
- `JMF` *label*: conditional (to false) jump;
- `JMP` *label*: unconditional jump;
- `HLT`: halt program (the last instruction of the generated code).

Note: Assume that the counting variable cannot be assigned within the body of the loop.