Solver: Nguyen Phan Huy

Email Address: huy004@e.ntu.edu.sg

1.

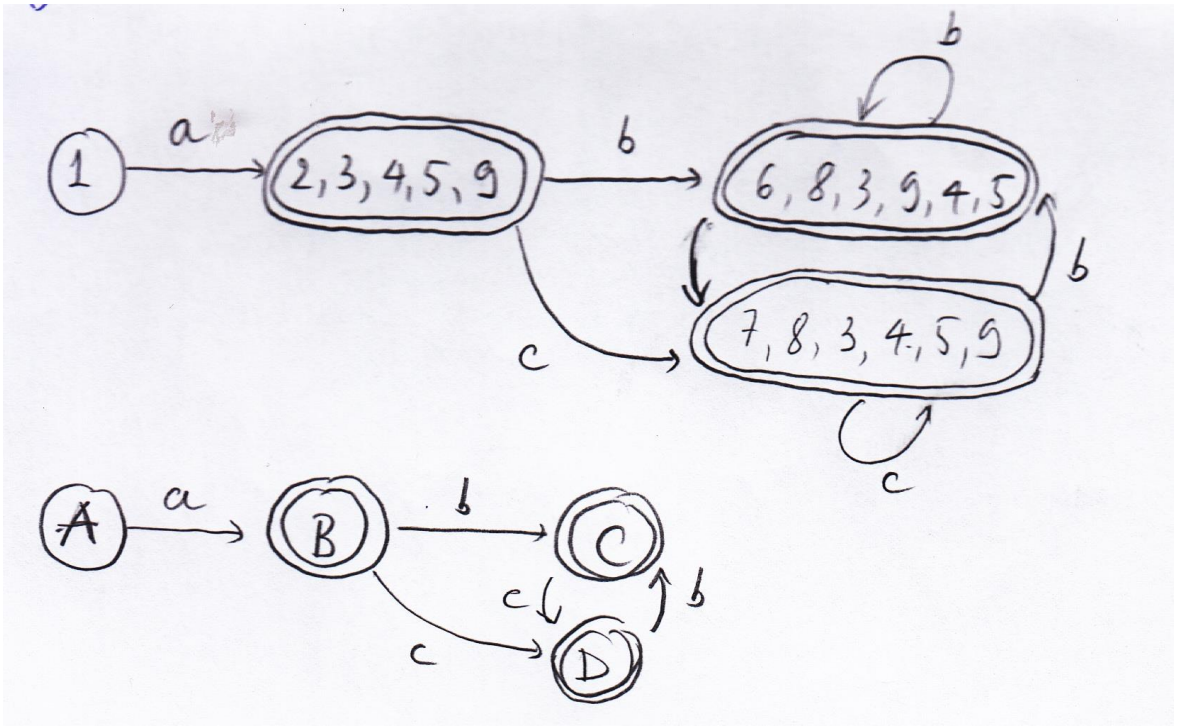
(a) (lecture 1)

- Phrase 1: Analysis
  - Lexical analysis
  - Syntax analysis
  - Semantic analysis
- Phrase 2: Synthesis
  - Optimization
  - Code generation

(b) (lecture 2)

- In a DFA, given the current state, and the next character, we can determine the next state without ambiguity. This is the reason why DFA can be represented by a transition table and coded in a table-driven form
- In a transition table, for each pair of (state, character), the next state is specified. Headers value for row and column are the characters and state list, while the value inside the cell is the state for each pair.
- In an implementation of a DFA, the next state can be retrieved from the table. The parsing process is under a loop of reading in the next character and determining the next state.

(c) Regular expression: a (b|c)*



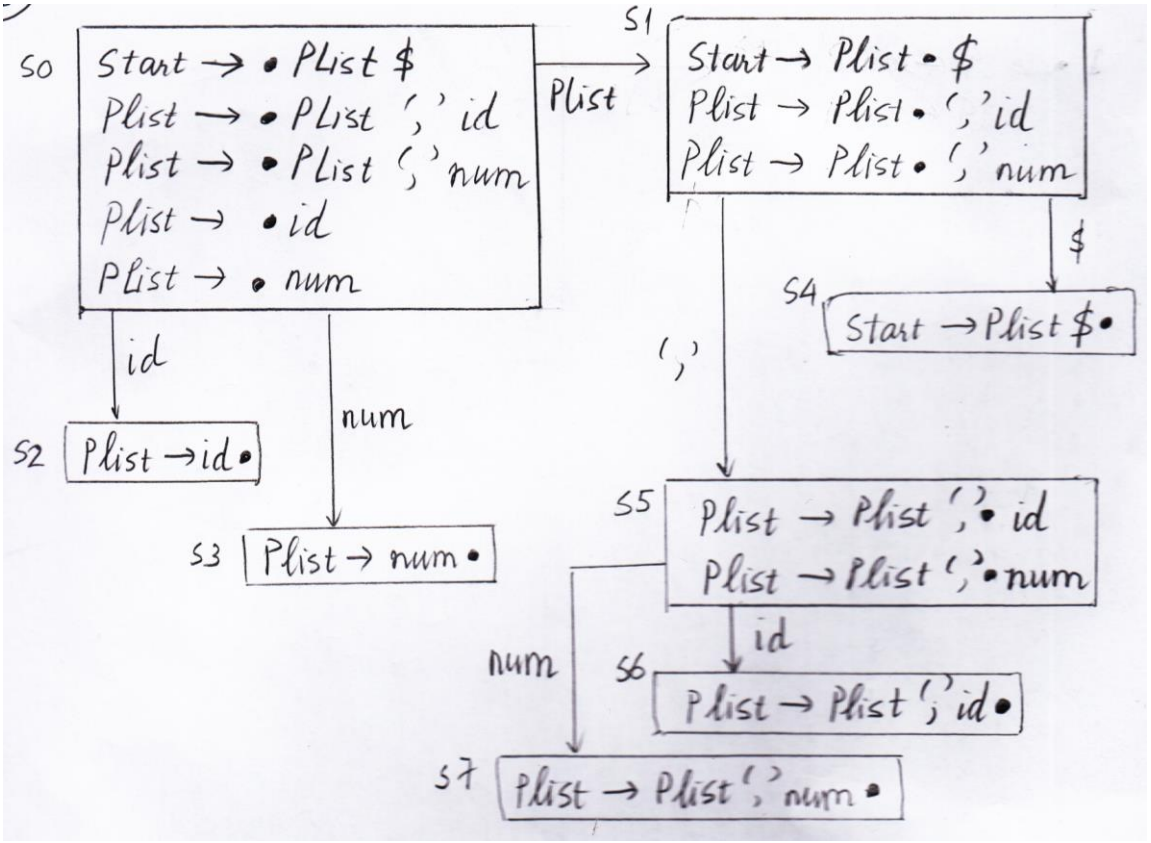
2. (a)

RHS -> RHS Child
| λ
Child -> ID ':' ID
| '<' ID ':' BOOLEAN '>'
| '<' ID ':' INT '>'
| '<' ID ':' CHAR '>'

(b)

(i) LR(0) automaton is an implementation of bottom-up parsing which needs to be used together with a stack of states, hence, it is called a pushdown automaton.

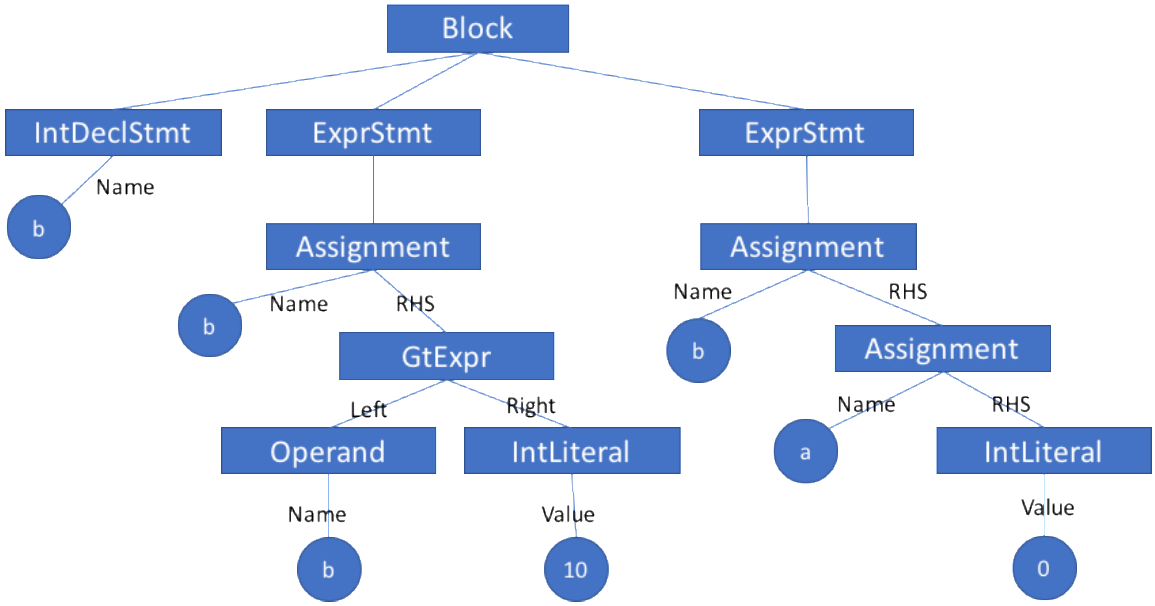
(ii)



(iii)

Plist -> Phead PTail
PTail -> Plist | λ
Phead -> id | num

3. (a)



(b)

- Line 1: a 'VarDecl', checking if there is any declaration of the same name in the scope. Result: no VarDecl of the same name 'a'. Checking return true.
- Line 2: There is a 'VarName' – 'a', checking if the variable has been declared. Result: Declaration of 'a' is found on line 1. Checking return true.
- Line 3: a 'VarDecl', checking if there is any declaration of the same name in the scope. Result: no VarDecl of the same name 'b'. Checking return true.
- Line 4: There are two 'VarName', 'a' and 'b', checking if those variables have been declared. Result: found declarations for both. Checking return true.
- Line 5: same as line 4.

(c)

- Line 2: An 'Assignment'. typecheck() is called on left and right, both return trues. Get types of Left and Right. Left returns Int (it traces the VarDecl in line 1 and return the type). Right returns int (it's a IntLiteral). Types of Left and Right are the same, return True.

- Line 4: An 'Assignment'. typecheck() is called on left and right.
    + typecheck() on left return true. (a varname)
    + typecheck() on Right – a GtExpr. typecheck() is called on left and right:
        + typecheck() on left returns true (a VarName)
        + typecheck() on right returns true (a IntLiteral)

+ Left and Right have same type of Int. Returns true
+ Type of left is Int, type of right is Boolean. Not matched. Return false.

- Line 5: An 'Assignment', type check process is similar to line 4.
  + typecheck() on left returns true
  + typecheck() on right – an Assignment will call typecheck() on its left and right. Both returns true.
  + left and right don't have the same type (Int and Assignment). Return false.

4. (a)
   - Translating directly from the AST to executable code can sometimes be difficult, having an IR allows the compiler to break up the transformation into simpler components
   - Some compilers (such as gcc or llvm) support multiple different target platforms. Platform-independent optimisations are done on an IR, then the optimised IR is converted to native executable code.

   (b)
   (i)
```
L1  $t1 = y + 1;
    $z = z * $t1;
    $t2 = x*x;
    x = $t2 + 1;
    if (x >= z) goto L1;
```
   (ii)
```
L1  load y;
    load 1;
    add;
    store $t1;
    load $t1;
    load z;
    mul;
    store z;
    load $x;
    load $x;
    mul;
    store $t2;
    load $t2;
    load 1;
    add;
    store x;
    load x;
    load z;
```

```
        ifgt L1;
```

Optimized:

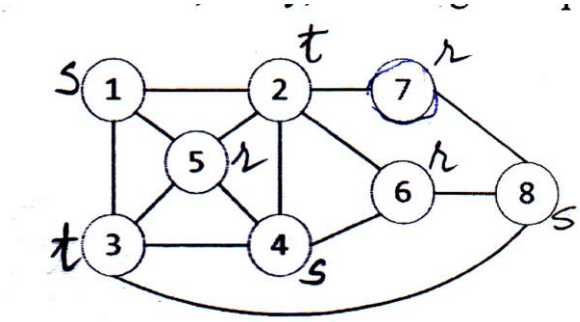
```
L1      load y;
        load 1;
        add;
        ~~store $t1;~~
        ~~load $t1;~~
        load z;
        mul;
        store z;
        load $x;
        ~~load $x;~~ dup;
        mul;
        ~~store $t2;~~
        ~~load $t2;~~
        load 1;
        add;
        ~~store x;~~
        ~~load x;~~
        load z;
        ifgt L1;
```

(iii)

Instruction		**Stack height**
L1	load y;	1
	load 1;	2
	add;	1
	load z;	2
	mul;	1
	store z;	0
	load $x;	1
	dup;	2
	mul;	1
	load 1;	2
	add;	1
	load z;	2
	ifgt L1;	0

Maximum height = 2

(c)



Stack

4
3
2
1
5 *: spilled candidate
6
8
7

5. (a) For a variable x to be alive after a node n in CFG, there must exist a path p from n to EXIT such that on p, x is read before it is reassigned (Liveness Analysis, Lecture 6)
(b)
- x is live since node (3) reads its value before x is reassigned
- y is live since node (6) reads its value before y is reassigned
- z is dead since there is no path from (5) to exit which read value of z before it is reassigned (node (6) reassigns value of z before we reach any other node)

(c)

$out_A(1) = in_A(1)$
$out_A(2) = in_A(2) \setminus \{y+z\} \cup \{y+1\}$
$out_A(3) = in_A(3)$
$out_A(4) = in_A(4) \setminus \{x-1\}$
$out_A(5) = in_A(5) \setminus \{y+z, y+1, y*2\}$
$out_A(6) = in_A(6) \setminus \{y+z\} \cup \{y+2\}$
$out_A(7) = in_A(7)$
$out_A(8) = in_A(8)$

$in_A(1) = \varnothing$
$in_A(2) = out_A(1)$
$in_A(3) = out_A(2) \cap out_A(6)$
$in_A(4) = out_A(3)$
$in_A(5) = out_A(4)$
$in_A(6) = out_A(5)$
$in_A(7) = out_A(3)$

$in_A(8) = out_A(7)$

(d)

Work list	**$out_A(m)$**	**$in_A(n)$**
1,2,…,8	$out_A(1) = \varnothing$	$in_A(2) = out_A(1) = \varnothing$
2,3,…,8	$out_A(2) = \{y+1\}$	$in_A(3) = out_A(2) \cap out_A(6) = \{y+1\}$
3,4,…,8	$out_A(3) = \{y+1\}$	$in_A(4) = out_A(3) = \{y+1\}$ $in_A(7) = out_A(3) = \{y+1\}$
4,5,…,8	$out_A(4) = \{y+1\}$	$in_A(5) = out_A(4) = \{y+1\}$
5,6,7,8	$out_A(5) = \varnothing$	$in_A(6) = out_A(5) = \varnothing$
6,7,8	$out_A(6) = \{y+2\}$	$in_A(3) = out_A(2) \cap out_A(6) = \varnothing$ (changed)
7,8,3	$out_A(7) = \{y+1\}$	$in_A(8) = out_A(7) = \{y+2\}$
8,3	$out_A(8) = \{y+2\}$	
3	$out_A(3) = \varnothing$	$in_A(4) = out_A(3) = \varnothing$(changed) $in_A(7) = out_A(3) = \varnothing$(changed)
4, 7	$out_A(4) = \varnothing$	$in_A(5) = out_A(5) = \varnothing$(changed)
7, 5	$out_A(7) = \varnothing$	$in_A(8) = out_A(7) = \varnothing$(changed)
5, 8	$out_A(5) = \varnothing$	$in_A(6) = out_A(5) = \varnothing$(unchanged)
8	$out_A(8) = \varnothing$	

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺