Chapter 3-4 and Exercises.
Written by : Dr. Chantana Chantrapornchai

Chapter 3 Summary

-Understand BNF, CFG representations.
-Understand how to create parse tree, how to show ambiguous grammar.
-Understand the concept of leftmost derivation, rightmost derivation.
-Understand left association, right association, precedence level in BNFs.
-Understand static semantic using the attribute grammar.
-Understand dynamic semantic using operational semantic, axiomatic semantic and denotational semantic.
-Example of static type checking using validity function.

Chapter 4 Summary

-Understand language recognization using top down and bottom up style.
-Understand lexemes, tokens definitions.
-Able to read DFA and regular expression. Derive code using DFA to check lexemes.
-Know to create recursive descent parsing.
-Know to check syntax for LL(1) and LR(1).

Exercise: Chapter 3

1. Given the following grammar,

```
<S>  ⟶  <A>
<A>  ⟶  <A> + <A> | <A> * <A> | id
<id> ⟶  a| b |c
```

Show that the grammar is ambiguous.

2. Based on the grammar

Assignment → *Identifier* = *Expression* ;
Expression → *Term* | *Expression* + *Term* | *Expression* − *Term*
Term → *Factor* | *Term* * *Factor* | *Term* / *Factor*
Factor → *Identifier* | *Literal* | (*Expression*)

Create a parse tree for (a) *2*x + 3*y* (b) *x + y*(x+2)*

3. Show that the following grammar

IfStatement → if (*Expression*) *Statement* |
if (*Expression*) *Statement* else *Statement*
Statement → *Assignment* | *IfStatement*

is ambiguous for the code

```
if (y > 0)
    if (x > 0) x = x+1;
else  y  = y+1;
```

4. Why attribute grammar is said to be static? Also explain the following terms.
   a. Inherit attribute
   b. Synthesis attribute
   c. Expected type
   d. Actual type
5. What are static semantic, and dynamic semantic? How are they differed?

6. In C Language , what is the operational semantic of

```
while (exp1)
  statements;
```

7. Show the operational semantic of the following code.

```
i = 5;
x = y =0;
do
{   if (x < 3)
        x = x+y*i;
      else
        x = x+y*i*2;
      i--;
}
while (i > 0);
```

8. In C Language , what is the operational semantic of

```
i=0; j =1; x = 10;
do
{
   i = i+j;
   if (j %2 ==0 )
    j = j+3;
   else j = j+2;
   x--;
} while (x > 0);
```

9. In Pascal Language , what is the operational semantic of the repeat-until loop?

```
repeat
  statements;
until (exp1);
```

10. Consider the syntax of if-statement.

$< \text{if\_stmt} > \rightarrow \text{if} \; < \text{exp} >< \text{stmt} >$

$< \text{exp} > \rightarrow < \text{var} > \; > \; < \text{var} > | \; < \text{var} >$

$< \text{var} > \rightarrow \text{A} \mid \text{B} \mid \text{C}$

Consider the attribute grammar of the above rules.

1. Syntax rule : $< \text{if\_stmt} > \rightarrow \text{if} \; < \text{exp} >< \text{stmt} >$

   Semantic rule: $< exp > .expected\_type \leftarrow boolean$

2. Syntax rule : $< \text{exp} > \rightarrow < \text{var} > \; > \; < \text{var} >$

   Semantic rule:

$< exp > .actual\_type \leftarrow \text{if} < \text{var} > [2].\text{actual\_type} = \text{int} \;\&\&$

$\text{var}[3].\text{actual\_type} = \text{int}$

$\text{then} \;\; \text{boolean}$

else undef_type

predicate:

$< \text{exp} > .actual\_type == < \text{exp} > .expected\_type$

3. Syntax $< \text{exp} > \rightarrow < \text{var} >$
   Semantic $< \text{exp} > .actual\_type \leftarrow < \text{var} > .actual\_type$
   predicate:
   $< \text{exp} > .actual\_type == < \text{exp} > .expected\_type$
4. Syntax $< \text{var} > \rightarrow A \mid B \mid C$
   Semantic $< \text{var} > .actual\_type \leftarrow look\_up(< \text{var} > .type)$

Suppose that we declare

```
boolean A,C;
int B;
```

Show your work in analyzing the statement

```
if (A > B)
    C++;
```

using the above attribute grammar. (Ignore the statement portion of the rule 1.) Can you compile the program? What are synthesized attributes and inherited attributes?

11. For the grammar as following

```
bool_exp =>  (Var == exp)
exp => Var op Var  | Var
op =>  &&  |  ||
Var => A | B
```

Show how to extend this for attribute grammar.

12. Find the weakest precondition of

```
if (x>0)
{
       y  = y-1;
       x = y -2;
}
else x = y-1;
```

where the post condition is {x > 10}.

13. What is the *loop invariant?* What are its properties?
14. Consider the code

```
while (y != x)
    y = y-1;
```

Find I and prove the correctness of the *while* loop using your I. Show your work.

15. Compute the weakest precondition and postcondition of the following sequence: (5pts)

```
A := 3 * (2 * B + A);
B := 2 * A - 1
{ B > 5 }
```

16. Consider the code.

```
int Sum (int n)
{
        int s=0;
        int i=0;
        while (i < n)
        {
            i =i+1;
            s = s+i;
        }
        return s;
}
```

If we divide the code into portions,

```
{Q}
  int s=0;
  int i=0;          } initialization
{P}
  while (i < n)
  {
        i =i+1;
        s = s+i;    } loop body
  }
{ (not test) and P}
  return s;         <-- finalization
{R}
```

Suppose Q = $\{n \geq 0\}$ and R = $\{s = 0+1+..+n\}$

And Loop invariant = $\{0 \leq i \wedge i \leq n \wedge s = 0+1+..+i\}$

Prove initialization , finalization, and loop body using axiomatic semantic.

17. Given the denotational semantic for binary number as follows:

1. $M('0') = 0$
2. $M('1') = 1$
3. $M(< bin\_num >'0') = 2 \times M(< bin\_num >)$
4. $M(< bin\_num >'1') = 2 \times M(< bin\_num >)+1$

Show your work of calculating the semantic for binary number 111.

18. Consider the C-Like logical pretest loop.

```
x = 1;
i = 1;
while (i< N)
{
        x = x * 2 +i;
        i = i+1;
}
```

Assume the state S = { <i,1>, <x,1>,<N,3>}. Show your work of computing the semantic of the loop using S based on M_l defined class.

```
M_l (while B do L,S) =
  if M_b (B,S) = undef
    then error
    else  if M_b(B,S) = false
    then  S
    else  if M_sl (L,S) = error
    then error
    else M_l(while B do L, M_sl (L,S))
```

19. What are the values of the following functions if *tm*={ <y,int>,<x,int>}?
    a. typeOf(2+y*5,*tm*)
    b. typeOf(x-y > 3*y,*tm*)
20. Given *tm*={ <i,int>,<even,int>}, compute the validity function of the following code.

```
if (i%2 == 0) even = even+1;
while (even > 0)
   even = even -1;
```

21. Consider the following code.

```
int main(int argc, char *argv[])
{
      int x=0;
      int i,j;
      i=0; j=5; //1
      while ( i < j)
      {
            i++;  j--;
            x += i+j;
      } //2
}
```

Using the denotational semantic.

(a) Write the initial state of the program at line 1.
(b) Trace the semantic of the program assuming the meaning of M_assign, M_exp, and M_Loop. Show your work.
(c) Write down the state of the program at line 2.

22. Imitate the following semantic of M_bin. Write the denotational semantic of M_dec.

$$1.\ \mathrm{M}(\text{'0'}) = 0$$

$$2.\ \mathrm{M}(\text{'1'}) = 1$$

$$3.\ \mathrm{M}(< bin\_num > \text{'0'}) = 2 \times \mathrm{M}(< bin\_num >)$$

$$4.\ \mathrm{M}(< bin\_num > \text{'1'}) = 2 \times \mathrm{M}(< bin\_num >) + 1$$

Exercise : Chapter 4

23. Based on the grammar:
    Create a rightmost derivation and a parse tree for *x* 2+ y (10 pts)*

    Is the grammar left association or right association?
    For the rule T→F | T * F
    Eliminate the left recursion.

24. Give an example of regular expressions.
25. What are benefits of using BNF form?
26. What are benefits of separating lexical analysis and syntax analysis?
27. What is a token lookahead?
28. Why do we need to get rid of left factor and left recursion in topdown parsers?
29. Given the following grammar, show your work in eliminating the left factor.

    A →Ab | Ac

    A → x

    Assume A is a nonterminal, b,c,x are terminals.

30. The chapter gives an example of using *endif* marker for solving ambiguous grammar for *if* statement. What are pros and cons?

31. Consider the following parsing table and grammar.

    1.E →E+T

    2. E → T

    3.T →T*F

    4.T → F

    5.F → (E)

    6.F → id

    Show your work in parsing the following expression:
    - ***id+(id*id)***
    - ***id* (id+id)***

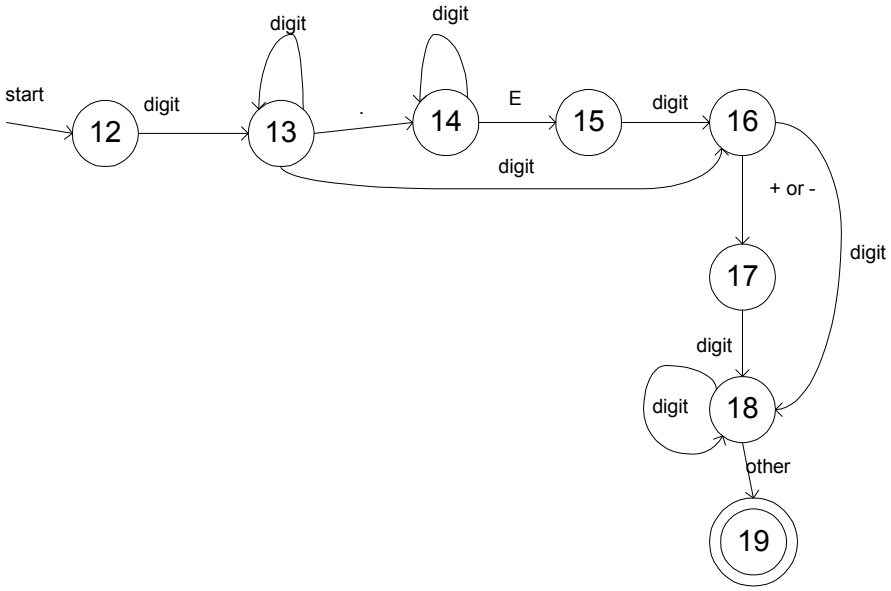
    in the following format.

Stack	Input	Action

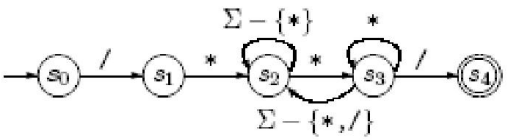
32. Consider the following state diagram. And the following utility subprogram. Write a section of code for lexical analysis part that implements the following state diagram.

- **`getChar`** - gets the next character of input, puts it in **`nextChar`**, determines its class and puts the class in **`charClass`**
- **`addChar`** - puts the character from **`nextChar`** into the place the lexeme is being accumulated, **`lexeme`**
- lookup - determines whether the string in **`lexeme`** is a reserved word (returns a code)

(a)



(b)



33. Consider the following grammar.

S → S + E | a
E → id
(a) Eliminate left recursion. Rewrite the grammar.

(b) Write the recursive descent parser for the grammar.

34. Consider the parsing table for LL Grammar given as following.

nonterminal	Input Symbol					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F →id			F →(E)		

Show your work in parsing id*(id)

Stack	Input	output