# Compiler

## Exercises for chapter 4

# 4.2.1

Consider the context-free grammar:

```
S -> S S + | S S * | a
```

and the string aa + a*.

1. Give a leftmost derivation for the string.
2. Give a rightmost derivation for the string.
3. Give a parse tree for the string.
4. ! Is the grammar ambiguous or unambiguous? Justify your answer.
5. ! Describe the language generated by this grammar.

## Answer

1. S =lm=> SS* => SS+S* => aS+S* => aa+S* => aa+a*
2. S =rm=> SS* => Sa* => SS+a* => Sa+a* => aa+a* 3.
3. Unambiguous
4. The set of all postfix expressions consist of addition and multiplication

---

# 4.2.2

Repeat Exercise 4 . 2 . 1 for each of the following grammars and strings:

1. S -> 0 S 1 | 0 1 with string 00011l.
2. S -> + S S | * S S | a with string + * aaa.
3. ! S -> S (S) S | ε with string (()())
4. ! S -> S + S | S S | (S) | S * | a with string (a+a)*a
5. ! S -> (L) | a 以及 L -> L, S | S with string ((a,a),a,(a))
6. !! S -> a S b S | b S a S | ε with string aabbab
7. The following grammar for boolean expressions:

```
8. bexpr -> bexpr or bterm | bterm
9. bterm -> bterm and bfactor | bfactor
10. bfactor -> not bfactor | (bexpr) | true | false
```

## Answer

1. S =lm=> 0S1 => 00S11 => 000111
2. S =rm=> 0S1 => 00S11 => 000111
3. Omit

4. Unambiguous

5. The set of all strings of 0s and followed by an equal number of 1s

2、

1. S =lm=> +SS => +*SSS => +*aSS => +*aaS => +*aaa

2. S =rm=> +SS => +Sa => +*SSa => +*Saa => +*aaa

3. Omit

4. Unambiguous

5. The set of all prefix expressions consist of addition and multiplication.

3、

1. S =lm=> S(S)S => (S)S => (S(S)S)S => ((S)S)S => (()S)S => (()S(S)S)S =>
(()(S)S)S => (()()S)S => (()())S => (()())

2. S =rm=> S(S)S => S(S) => S(S(S)S) => S(S(S)) => S(S()) => S(S(S)S()) =>
S(S(S)()) => S(S()()) => S(()()) => (()())

3. Omit

4. Ambiguous

5. The set of all strings of symmetrical parentheses

4、

1. S =lm=> SS => S*S => (S)*S => (S+S)*S => (a+S)*S => (a+a)*S => (a+a)*a

2. S =rm=> SS => Sa => S*a => (S)*a => (S+S)*a => (S+a)*a => (a+a)*a

3. Omit

4. Ambiguous

5. The set of all string of plus, mupplication, 'a' and symmetrical parentheses, and
plus is not the beginning and end of the position, multiplication is not the
beginning of the position

5、

1. S =lm=> (L) => (L, S) => (L, S, S) => ((S), S, S) => ((L), S, S) => ((L, S), S, S) =>
((S, S), S, S) => ((a, S), S, S) => ((a, a), S, S) => ((a, a), a, S) => ((a, a), a, (L))
=> ((a, a), a, (S)) => ((a, a), a, (a))

2. S =rm=> (L) => (L, S) => (L, (L)) => (L, (a)) => (L, S, (a)) => (L, a, (a)) => (S, a, (a)) => ((L), a, (a)) => ((L, S), a, (a)) => ((S, S), a, (a)) => ((S, a), a, (a)) => ((a, a), a, (a))
3. Omit
4. Unambiguous
5. Something like tuple in Python

6、
1. S =lm=> aSbS => aaSbSbS => aabSbS => aabbS => aabbaSbS => aabbabS => aabbab
2. S =rm=> aSbS => aSbaSbS => aSbaSb => aSbab => aaSbSbab => aaSbbab => aabbab
3. Omit
4. Ambiguous
5. The set of all strings of 'a's and 'b's of the equal number of 'a's and 'b's
7、 Unambiguous, boolean expression

---

## 4.2.3

Design grammars for the following languages:
1. The set of all strings of 0s and 1s such that every 0 is immediately followed by at least one 1.
2. ! The set of all strings of 0s and 1s that are palindromes; that is, the string reads the same backward as forward.
3. ! The set of all strings of 0s and 1s with an equal number of 0s and 1s.
4. !! The set of all strings of 0s and 1s with an unequal number of 0s and 1s.
5. ! The set of all strings of 0s and as in which 011 does not appear as a substring.
6. !! The set of all strings of 0s and 1s of the form xy, where x<>y and x and y are of the same length.

### Answer

1、

```
S -> (0?1)*
```

2、

```
S -> 0S0 | 1S1 | 0 | 1 | ε
```

3、

```
S -> 0S1S | 1S0S | ε
```

5、

```
S -> 1*(0+1?)*
```

---

## 4.2.4

There is an extended grammar notation in common use. In this notation, square and curly braces in production bodies are metasymbols (like -> or |) with the following meanings:

1. Square braces around a grammar symbol or symbols denotes that these constructs are optional. Thus, production A -> X[Y]Z has the same effect as the two productions A -> XYZ and A -> XZ.
2. Curly braces around a grammar symbol or symbols says that these symbols may be repeated any number of times, including zero times. Thus, A -> X{YZ} has the same effect as the infinite sequence of productions A -> X, A -> XYZ, A -> XYZYZ, and so on.

Show that these two extensions do not add power to grammars; that is, any language that can be generated by a grammar with these extensions can be generated by a grammar without the extensions.

### Proof

| | |
|---|---|
| A -> X[Y]Z | A -> XZ \| XYZ |
| A -> X{YZ} | A -> XB<br>B -> YZB \| ε |

## 4.2.5

Use the braces described in Exercise 4.2.4 to simplify the following grammar for statement blocks and conditional statements:

```
stmt -> if expr then stmt else stmt
| if stmt them stmt
```

```
| begin stmtList end
stmtList -> stmt; stmtList | stmt
```

### Answer

```
stmt -> if expr then stmt [else stmt]
| begin stmtList end
stmtList -> stmt [; stmtList]
```

---

## 4.2.6

Extend the idea of Exercise 4.2.4 to allow any regular expression of grammar symbols in the body of a production. Show that this extension does not allow grammars to define any new languages.

### Proof

Every regular grammar has a corresponding not extended grammar

---

## 4.2.7 !

A grammar symbol X (terminal or nonterminal) is useless if there is no derivation of the form S =*=> wXy =*=> wxy. That is, X can never appear in the derivation of any sentence.

1. Give an algorithm to eliminate from a grammar all productions containing useless symbols.
2. Apply your algorithm to the grammar:

```
3. S -> 0 | A
4. A -> AB
5. B -> 1
6. `
```

---

## 4.2.8

The grammar in Fig. 4.7 generates declarations for a single numerical identifier; these declarations involve four different, independent properties of numbers.

```
stmt -> declare id optionList
optionList -> optionList option | ε
option -> mode | scale | precision | base
mode -> real | complex
scale -> fixed | floating
precision -> single | double
base -> binary | decimal
```

1. Generalize the grammar of Fig. 4.7 by allowing n options Ai, for some fixed n and for i = 1,2... ,n, where Ai can be either ai or bi· Your grammar should use only 0(n) grammar symbols and have a total length of productions that is O(n).

2. ! The grammar of Fig. 4.7 and its generalization in part (a) allow declarations that are contradictory and/or redundant, such as

declare foo real fixed real floating

We could insist that the syntax of the language forbid such declarations; that is, every declaration generated by the grammar has exactly one value for each of the n options. If we do, then for any fixed n there is only a finite number of legal declarations. The language of legal declarations thus has a grammar (and also a regular expression), as any finite language does. The obvious grammar, in which the start symbol has a production for every legal declaration has n! productions and a total production length of O(n x n!). You must do better: a total production length that is O(n2^n)

3. !! Show that any grammar for part (b) must have a total production length of at least 2n.

4. What does part (c) say about the feasibility of enforcing nonredundancy and noncontradiction among options in declarations via the syntax of the programming language?

## Answer

1、

```
stmt -> declare id optionList
optionList -> optionList option | ε
```

```
option -> A_1 | A_2 | … | A_n
A_1 -> a_1 | b_1
A_2 -> a_2 | b_2
…
A_n -> a_n | b_n
```