Midterm Exam

Compilers, 2022, Term 2

[NB]

1. Explanation should be provided. (답만 기입하면 안됨. 설명 필수)
2. Take photos and upload images if you need. (필기 답안은 사진 Upload)
3. **0 Score** on copy, source and any caught cheatings. (부정행위 **0점 처리**)

1. Answer the questions

A. What is the difference between a compiler and an interpreter?

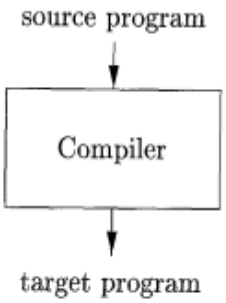


Figure 1.1: A compiler

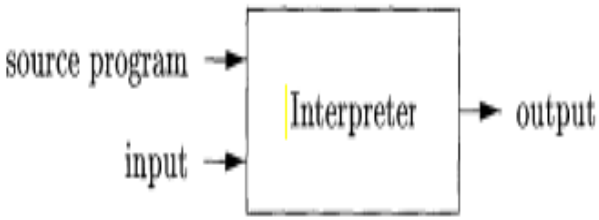


Figure 1.3: An interpreter

Compiler and interpreter are both types of programming language processors.

The compiler converts source code into object code, in other words source program into target program.

However interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user. In

other words, interpreter produces output –not object code- as a soruce code's translation.

B. What are important qualities of compilers?

The important elements to improve qualities of compilers are correct recognization of program constructs, speed of compilation and cognifition of source code, preservation the meaning of the code, Good error reporting and handling, Code debugging help etc.

C. Why are compilers commonly split into multiple passes?

The kind of compilers's split method is three – one pass, two pass, multiple pass. Advantage of multiple passes compared to other methods is machine Independent and more expressive. So multiple passes method is commonly used in many of compilers.

D. What is "abstract" about an abstract syntax tree?

Mean of "abstract" in abstract syntax tree is it does not detailly represent than it appears in real syntax. Real syntax trees -like concrete syntax trees, parse trees- are built by a parser during the source code translation and compiling process. However abstract syntax trees contain an abstract representation of a program's syntax.

E. What is "context-free" about a context-free grammar?

Mean of "context-free" in context-free grammar is it is regardless of the context of a nonterminal if that can be applied production rules.

2. **Do a lexical analysis of the following source codes.**

Divide the following C++ program:

```
float limitedSquare(x){float x;
  /* returns x-squared, nut never more than 100 */
  return (x <= -10.0 || x >= 10.0) ? 100 : x*x;
}
```

Hint

```
<float> <id, limitedSquaare> <(> <id, x> <)> <{>
```

<float> <id, limitedSquaare> <(> <id, x> <)> <{>
<float> <id, x>
<return> <(> <id, x> <op,"<="> <num, -10.0> <op, "||"> <id, x> <op, ">="> <num, 10.0> <)> <op, "?"> <num, 100> <op, ":"> <id, x> <op, "*"> <id, x>
<}>

3. **Answer the questions**

Describe the languages denoted by the following regular expressions:

1. a(a|b)*a
2. ((ε|a)b*)*
3. (a|b)*a(a|b)(a|b)
4. a*ba*ba*ba*
5. !! (aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*

(Please ignore !! in the question 5)

1. **String of a's and b's that start and end with a.**
2. **String of a's and b's.**
3. **String of a's and b's that the character third from the last is a.**
4. **String of a's and b's that only contains three b.**
5. **String of a's and b's that has a even number of a and b.**

4. Convert to DFA's the NFA's of

A. Fig. 3.26

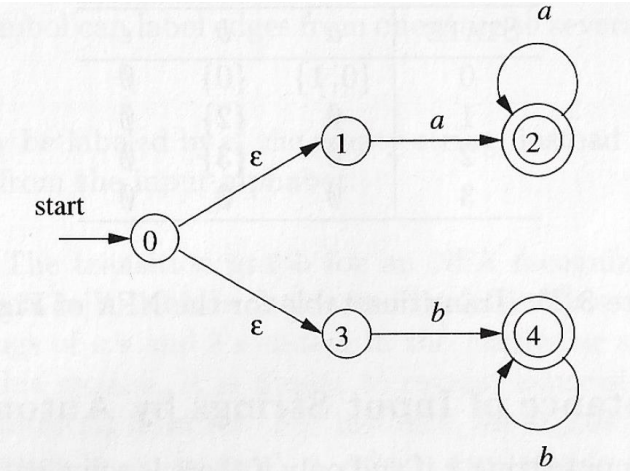
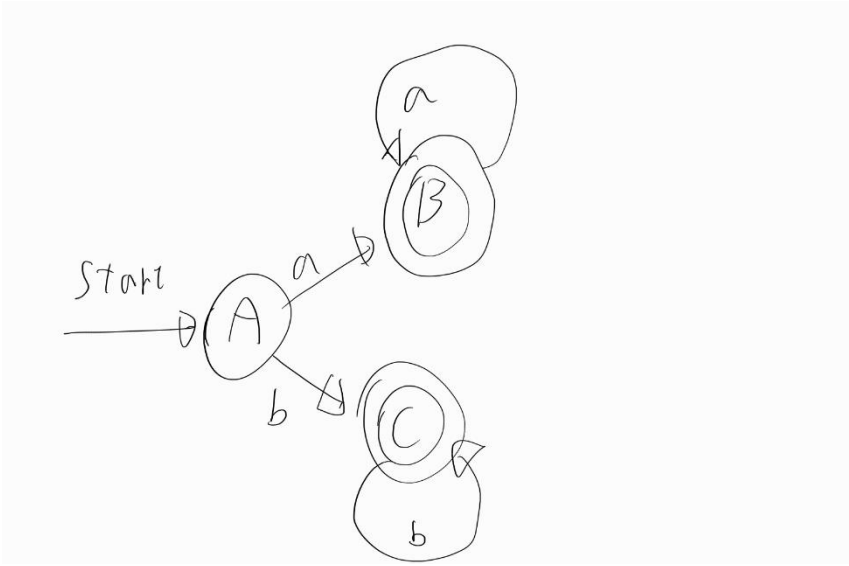


Figure 3.26: NFA accepting $\mathbf{aa^*|bb^*}$

Transition table

NFA State	DFA State	a	B
{0, 1, 3}	A	B	C
{2}	B	B	∅
{4}	C	∅	C

DFA



B. Fig. 3.29

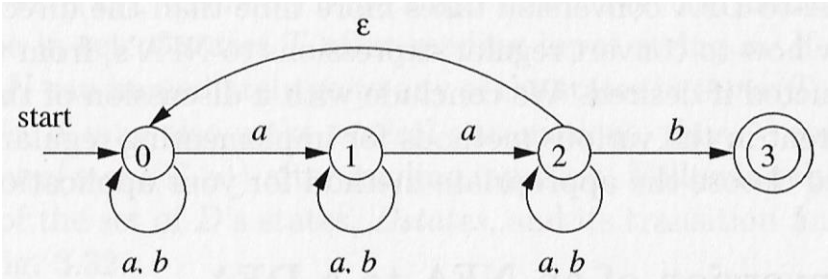
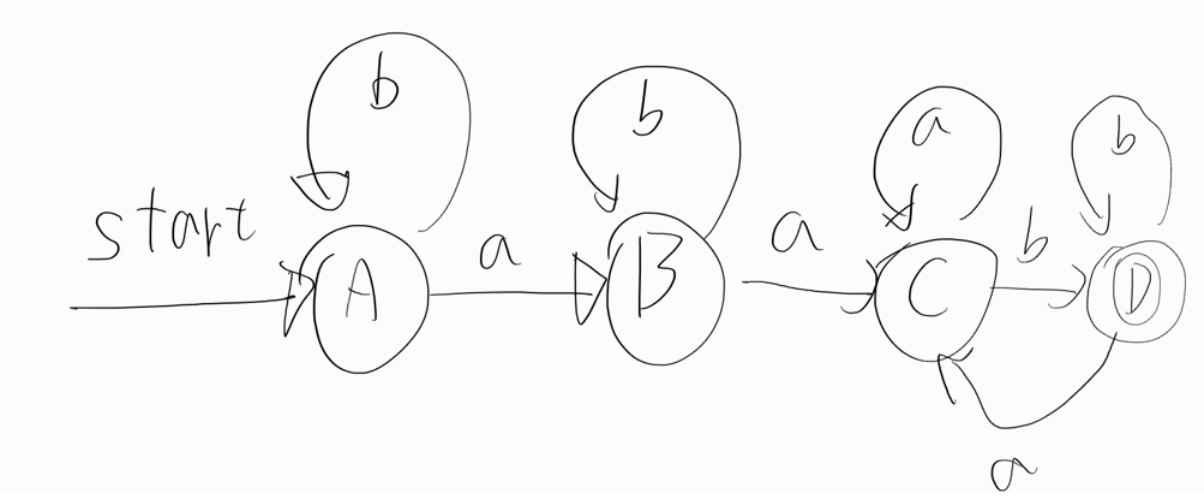


Figure 3.29: NFA for Exercise 3.6.3

NFA State	DFA State	a	b
{0}	A	B	A
{0, 1}	B	C	B
{0,1,2}	C	C	D
{0,2,3}	D	C	D

DFS



C. Fig. 3.30

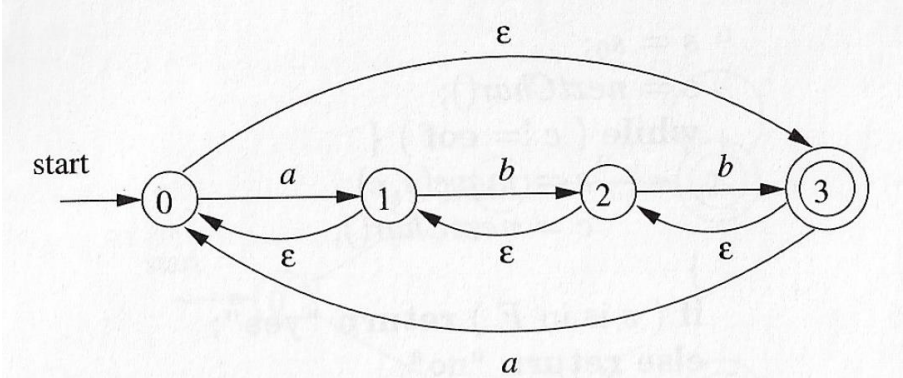
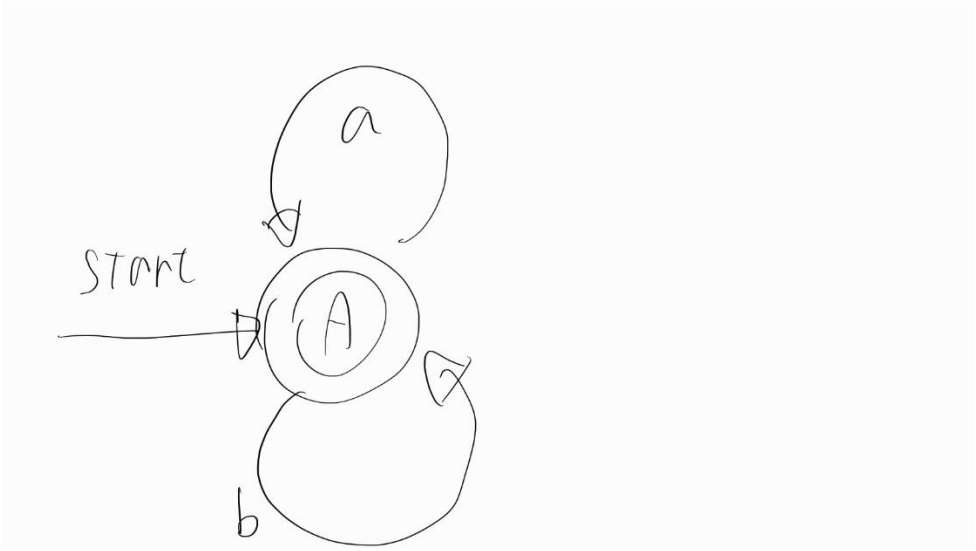


Figure 3.30: NFA for Exercise 3.6.4

NFA State	DFA State	a	B
{0,1,2,3}	A	A	A

DFS



5. Answer the questions

A.

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.

1. Give a leftmost derivation for the string.

   S =leftmost

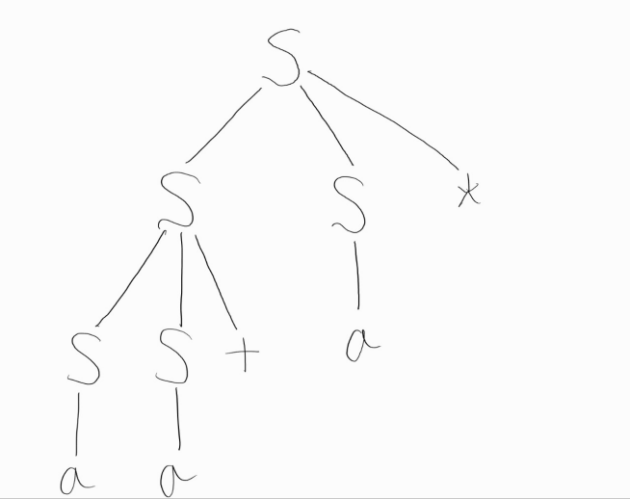
   => SS* => SS+S* => aS+S* => aa+S* => aa+a*

2. Give a right most derivation for the string.

   S =rightmost

   => SS* => Sa* => SS+a* => Sa+a* => aa+a*

3. Give a parse tree for the string.



B. Answer 1, 2, 3 & 5

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.

1. **S -> (0?1)***
2. **S -> 0S0 | 1S1 | 0 | 1 | ε**
3. **S -> 0S1S | 1S0S | ε**
5. **S -> 1*(0+1?)***

6. Answer the questions

A. Describe a regular expression for more than or equal to 2 digit odd integer numbers (두자리 이상의 홀수를 정규표현식으로 나타내시오.)

Suppose that our alphabet is all ASCII characters. A regular expression for odd integer numbers is (+|-)?(0|1|2|3|4|5|6|7|8|9)*(1|3|5|7|9)

And it can express by that : (+|-)?[0-9]*[13579]

B. The syntax for regular expressions does not carry over to CFGs which cannot use *, or parentheses. Convert a regular expression S → a*b into a CFG

The syntax for regular expressions does not carry over to CFGs. So can't use , |,or parentheses. So it can express by that : S -> Ab

~~**C.** Given the grammar P → ε | PP | (P) for balanced parentheses, the ambiguity can be resolved like below. Complete the derivation rule for S.~~

7. **Answer the questions**

A. What are problems of BFS (Breadth First Search)?

Problems of BFS is Slow, in other words it Enormous time and memory usage. It can induce lots of wasted effort and high branching factor.

B. What are problems of Leftmost BFS ?

Leftmost BFS may fail on right-recursive grammars something like "caaaaaaaaaa". And it uses exponential memory. This Worst-case runtime is exponential. And Worst-case memory usage is exponential

C. What are problems of Leftmost DFS (Depth First Search)?

Leftmost DFS may fail on left-recursive grammars. **Worst-case runtime is exponential. Worst-case memory usage is linear.**

~~**D.** Eliminate left recursion of the following expression~~

8. Answer the questions

A. 다음 문법에 대해 각 논터미널 기호의 FIRST 와 FOLLOW 를 구하시오

S → aSe

S → B

B → bBe

B → C

C → cCe

C → d

First set

Step	first_set							
	S	B	C	a	b	c	d	e
(1) First loop	∅	∅	∅					
(2) Second (nested) loop	{a}	{b}	{c,d}	{a}	{b}	{c}	{d}	{e}
(3) Third loop, production 2	{a,b}	{b}	{c,d}	{a}	{b}	{c}	{d}	{e}
(4) Third loop, production 4	{a,b}	{b,c,d}	{c,d}	{a}	{b}	{c}	{d}	{e}
(5) Third loop, production 2	{a,b,c,d}	{b,c,d}	{c,d}	{a}	{b}	{c}	{d}	{e}

Follow set

Step	Followset		
	S	B	C
(1)Initialization	{ λ }	∅	∅
(2)Process S in production 1	{θ, λ }	∅	∅
3)Process B in production 2	{θ, λ }	{θ, λ }	∅

2)Process B in production 3	NO CHANGES		
2)Process C in production 4	{θ, λ }	{θ, λ }	{θ, λ }
2)Process in C production 5	NO CHANGES		

9. Fill in the blanks

A. The handle of a parse tree T is the (**leftmost**) complete cluster of leaf nodes.

B. A left-to-right, bottom-up parse works by iteratively searching for a handle, then (**reducing**) the handle.

C. The bottom-up parsers we will consider are called (**shift**)/reduce parsers.

- Contrast with the LL(1) predict/(**match**) parser.

D. Idea: Split the input into two parts:

- (**Left**) substring is our work area; all handles must be here.
- (**Right**) substring is input we have not yet processed; consists purely of terminals.

E. At each point, decide whether to:

- Move a terminal across the split (**shift**)

- Reduce a handle (**reduce**)

F. Consequently, shift/reduce parsing means

- The Shift: Move a terminal from the (**right**) to the (**left**) area.
- Reduce: (**Replace**) some number of symbols at the (**right**) side of the left area

10. Build a LL(1) grammar for expressions consisting of variables, infix binary addition and multiplication (with usual priorities), and parentheses. Note that we do not care about associativity of binary operations at the level of the parse tree.

```
S  ::= T EOF
T  ::= F RT
RT ::= + T | ε
F  ::= B RF
RF ::= * F | ε
B  ::= id | ( T )
```

Compute FIRST, FOLLOW sets & complete a parsing table.

- **FIRST Sets**

FIRST(B) = { id , (}

FIRST(RF) = { * }

FIRST(F) = { id , (}

FIRST(RT) = { + }

FIRST(T) = { id , (}

FIRST(S) = { id , (}

- **FOLLOW Sets**

FOLLOW(S) = { }

FOLLOW(T) = { EOF ,) }

FOLLOW(RT) = { EOF ,) }

FOLLOW(F) = { + , EOF ,) }

FOLLOW(RF) = { + , EOF ,) }

FOLLOW(B) = { * , + , EOF ,) }

- **Parsing table**

	Id	+	*	(	)	EOF
S	1			1		
T	1			1		
RT		1			2	2
F	1			1		
RF		2	1		2	2
B	1			2		

<The end of the exam>