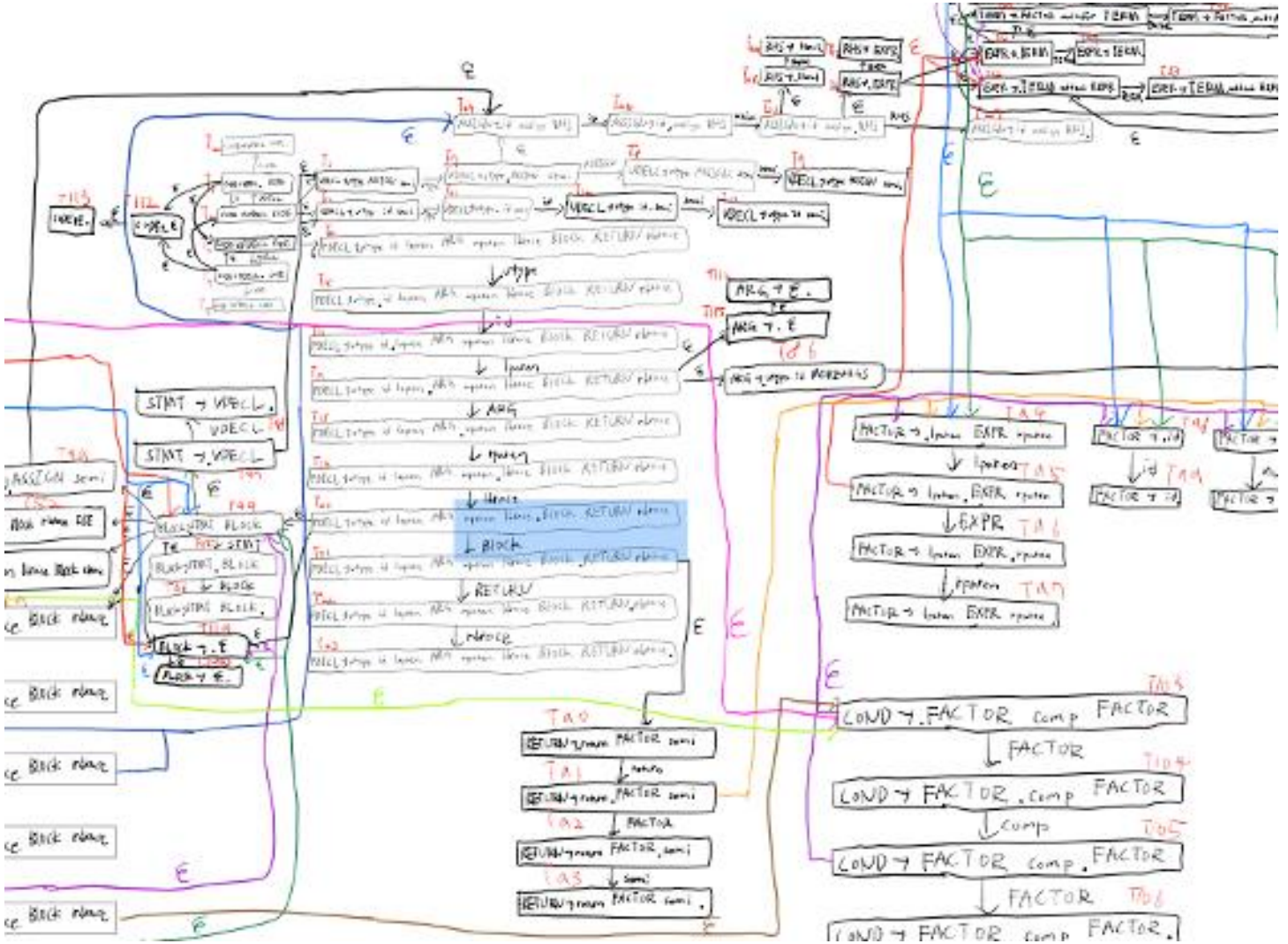
Syntax Analyzer 구현

TEAM 안승재, 우승진

1. NFA 작성

- CFG를 기반으로 NFA를 그렸습니다. T0부터 T122까지 123개가 나왔으며 모든 경우의 수를 포함하여 그렸습니다.



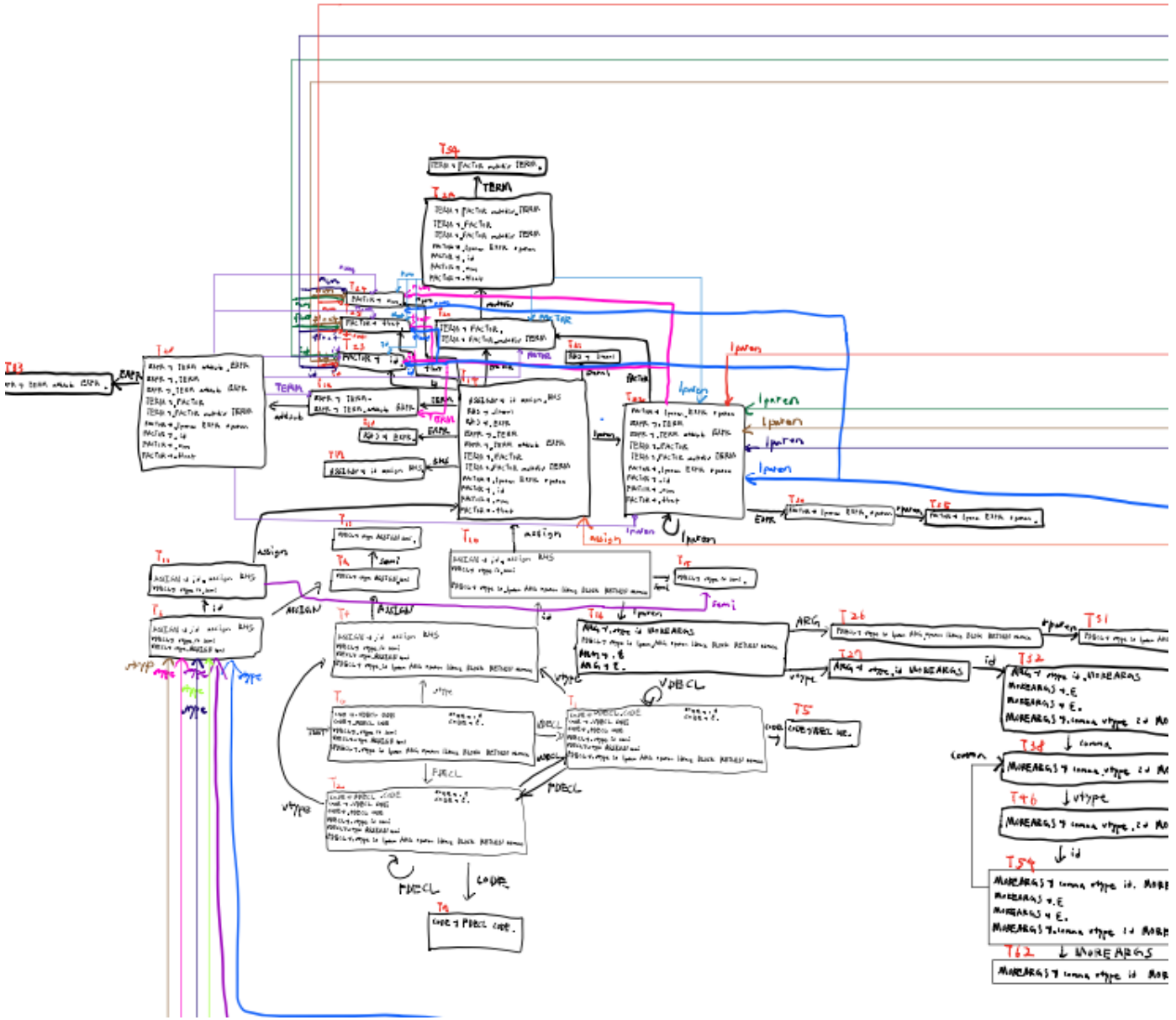
2-1. NFA to DFA

- 위의 NFA를 기반으로 모든 스테이트에서 스테이트로 가는 경우의 수를 엑셀 테이블로 나타내었습니다. 84개의 총 스테이트를 가지고 있습니다.
- 알파벳, 이전스테이트 에서 무슨 터미널 또는 터미널을 만나 T(n)안의 알파벳이 나오며 그 알파벳으로 최종 여러 알파벳을 구하고 그 최종 알파벳에 맞는 스테이트를 표시해주었습니다.
- 또 이제 add -> add. 다른 터미널또는 논터미널을 못받아 들이는 경우는 스테이트 옆에 E를 처리해서 이 스테이트에서 다른 논터미널 또는 터미널이 받는 것을 불가능하다고 표시하였습니다.

이전알파벳	이전스테이트	터미널 또	T(N) n안의	최종 알파벳	다음 스테이트(3,8,12 없이 84개)				
X	START	X		0,3,6,10,14,112,113	0				
0,3,6,10,14,112,113	0	VDECL	1	1,0,3,6,10,14,112,113	1				
0,3,6,10,14,112,113	0	FDECL	4	4,0,3,6,10,14,112,113	2				
0,3,6,10,14,112,113	0	vtype	7,11,15	7,11,15,24	4				
1,0,3,6,10,14,112,113	1	VDECL	1	1,0,3,6,10,14,112,113	1				
1,0,3,6,10,14,112,113	1	FDECL	4	4,0,3,6,10,14,112,113	2				
1,0,3,6,10,14,112,113	1	CODE	2	2	5	E			
1,0,3,6,10,14,112,113	1	vtype	7,11,15	7,11,15,24	4				
4,0,3,6,10,14,112,113	2	VDECL	1	1,0,3,6,10,14,112,113	1				

2-2. DFA

- 앞에서의 엑셀표를 기반으로 모든 스테이트와 스테이트들의 관계를 표시해 주었습니다.
- 여기서도 스테이트는 T0부터 T86까지 있으나 3,8,12 스테이트는 존재하지 않아 84개의 총 스테이트를 가지고 있습니다.



3. FollowSet 작성

- CFG를 보고 각 non-terminal에 대해서 FollowSet을 작성하였습니다.

Follow(CODE) = { $ }

Follow(VDECL) = Follow(STMT) ∪ Follow(CODE) ∪ First(CODE) = ∅
= { for, while, if, vtype, id, rbrace, return, $ }

Follow(ASSIGN) = { semi, rparen }

Follow(FDECL) = Follow(CODE) ∪ First(CODE) − ε = { vtype, $ }

Follow(ARG) = { rparen }

Follow(MOREARGS) = Follow(ARG) = { rparen }

Follow(BLOCK) = First(RETURN) − ε ∪ { rbrace } = { rbrace, return }

Follow(STMT) = First(BLOCK) − ε ∪ Follow(BLOCK) = { for, while, if, vtype, id, rbrace, return }

Follow(ELSE) = Follow(STMT) = { for, while, if, vtype, id, rbrace, return }

Follow(RHS) = Follow(ASSIGN) = { semi, rparen }

Follow(EXPR) = { rparen } ∪ Follow(RHS) = { semi, rparen }

Follow(TERM) = { addsub } ∪ Follow(EXPR) = { addsub, semi, rparen }

Follow(FACTOR) = { multdiv, comp, semi } ∪ Follow(COND) ∪ Follow(TERM) = { addsub, multdiv, semi, comp, rparen }

Follow(COND) = { rparen, semi }

Follow(RETURN) = { rbrace }

4. SLR Parsing Table

앞에서 작성한 FollowSet과 DFA를 기반으로 SLR Parsing Table을 Excel파일로 작성하였습니다.

		ACTION																			GOTO											
		vtype	num	float	iteral	id	if	else	while	for	return	addsub	multdiv	assign	comp	semi	comma	lparen	rparen	lbrace	rbrace	$	CODE	VDECL	FDECL	ARG	MOREARG	BLOCK	STMT	ASSIGN	RHS	EXPR	TERM	FACTOR	COND	RETURN	ELSE
3	STATE0	S4																				R(01-3)		STATE1	STATE2												
4	STATE1	S4																				R(01-3)	STATE5	STATE1	STATE2												
5	STATE2	S4																				R(01-3)	STATE7	STATE1	STATE2												
6	STATE3																																				
7	STATE4					S10																								STATE9							
8	STATE5																					R(01-1)															
9	STATE6					S11																								STATE9							
10	STATE7																					R(01-2)															
11	STATE8																																				
12	STATE9															S13																					
13	STATE10													S14		S15		S16																			
14	STATE11													S14		S15																					
15	STATE12																																				
16	STATE13	R(02-2)				R(02-2)	R(02-2)		R(02-2)	R(02-2)	R(02-2)										R(02-2)	R(02-2)															
17	STATE14		S24	S25	S21	S23												S22													STATE17	STATE18	STATE19	STATE20			
18	STATE15	R(02-1)				R(02-1)	R(02-1)		R(02-1)	R(02-1)	R(02-1)										R(02-1)	R(02-1)															
19	STATE16	S27																	R(05-2)							STATE26										
20	STATE17															R(03-1)			R(03-1)																		
21	STATE18															R(13-1)			R(13-1)																		
22	STATE19											S28				R(14-2)			R(14-2)																		
23	STATE20											R(15-2)	S29			R(15-2)			R(15-2)																		
24	STATE21															R(13-2)			R(13-2)																		
25	STATE22		S24	S25		S23												S22														STATE30	STATE19	STATE20			
26	STATE23											R(16-2)	R(16-2)		R(16-2)	R(16-2)			R(16-2)																		
27	STATE24											R(16-3)	R(16-3)		R(16-3)	R(16-3)			R(16-3)																		
28	STATE25											R(16-4)	R(16-4)		R(16-4)	R(16-4)			R(16-4)																		
29	STATE26																		S31																		
30	STATE27					S32																															
31	STATE28		S24	S25		S23												S22													STATE33	STATE19	STATE20			
32	STATE29		S24	S25		S23												S22														STATE34	STATE20			
33	STATE30																		S35																		
34	STATE31																			S36																
35	STATE32																S38		R(06-2)								STATE37									
36	STATE33															R(14-1)			R(14-1)																		
37	STATE34											R(15-1)				R(15-1)			R(15-1)																		
38	STATE35											R(16-1)	R(16-1)		R(16-1)	R(16-1)			R(16-1)																		
39	STATE36	S6				S61	S43		S44	S45	R(07-2)										R(07-2)			STATE39				STATE42	STATE40	STATE41							
40	STATE37																		R(05-1)																		
41	STATE38	S46																																			
42	STATE39	R(08-1)				R(08-1)	R(08-1)		R(08-1)	R(08-1)	R(08-1)										R(08-1)																
43	STATE40	S6				S61	S43		S44	S45	R(07-2)										R(07-2)			STATE39				STATE47	STATE40	STATE41							
44	STATE41															S48																					
45	STATE42										S50																									STATE49	
46	STATE43																	S51																			
47	STATE44																	S52																			
48	STATE45																	S53																			
49	STATE46					S54																															
50	STATE47										R(07-1)										R(07-1)															

5. 구현
(CFG 와 파싱테이블 읽기)

- 파싱테이블과 cfg를 자유롭게 사용하기 위해 엑셀파일을 읽어와 actionTable , goToTable , cgfTable 에 파싱테이블의 action부분, 파싱테이블의 goTo 부분 , cfg 부분을 3개로 나누었습니다.
- 그리고 각각 엑셀로 읽어와 테이블에 담았습니다.

```
#모든 그래머와 slr Table의 정보를 읽어오기
def setGrammarAndSLRTable(self):

    # actionTable 읽기
    self.actionTable = pd.read_excel("./excel/actionTable.xlsx",
                                     sheet_name = 'Sheet1',
                                     na_values = 'NaN',
                                     thousands = ',',
                                     nrows = 88)

    # goToTable 읽기
    self.goToTable = pd.read_excel("./excel/goToTable.xlsx", # write your directory here
                                   sheet_name = 'Sheet1',
                                   na_values = 'NaN',
                                   thousands = ',',
                                   nrows = 88)

    # cfgTable 읽기
    self.cfgTable = pd.read_excel("./excel/cfgTable.xlsx", # write your directory here
                                  sheet_name = 'Sheet1',
                                  na_values = 'NaN',
                                  thousands = ',',
                                  nrows = 18)
```

5. 구현
(Lexical analyzer 아웃풋 읽기)

- 이전에 있었던 lexical analyzer 에서 받아왔던 아웃 파일을 에러를 탐지하기 위한 lineNumber , 실제 타입을 확인하기 위해 tokenType, 실제 값은 tokenInput 에 넣어주었습니다.
- 기존 lexical analysis의 터미널과 이번 cfg의 터미널들이 차이가 있었으므로 tokenNameChange라는 함수를 통해 이번 cfg에 맞게 처리해주었습니다.

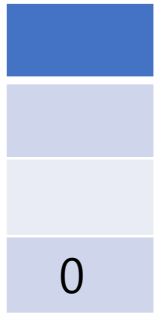
```
#lexical analyzer의 결과물을 읽어서 리스트에 넣는다
def openFileAndGetText(self):
  try:
    f = open("./input.c.out", 'r', encoding = "utf-8")
    inputLine = f.readline()
    while True:

      inputLine = f.readline()
      inputLine=inputLine.split("\n")[0]

      if inputLine =="":            #더이상 문장이 없으면 반복문 탈출
        break
      elif inputLine != '"':
        lineNumber = inputLine.split(",")[1]
        tokenType = inputLine.split(",")[2]
        tokenInput = inputLine.split(",")[3]
        #인풋값 처리
        tokenType, tokenInput = tokenNameChange(tokenType,tokenInput)
        #에러 처리
        if(tokenType=="Error"):
          print("인풋 파일에 에러가 있습니다")
          break;

        #whitespace 부분 무시
        elif(tokenType!="Whitespace"):
          #하나씩 하나씩 추가
          self.inputResult.append({"lineNumber":lineNumber , "tokenType" : tokenType , "tokenInput" : tokenInput})
    f.close()
    #마지막 토큰 추가
    self.inputResult.append({"lineNumber":0 , "tokenType" : "$" , "tokenInput" : "$"})
  except FileNotFoundError:
    print("해당 파일명이 없습니다.")
```

5. 구현
(파싱 알고리즘(1))

- 엑셀을 읽고 아웃풋 파일을 읽는 두가지 모든 준비가 완료되면, SLR파싱을 시작하게 됩니다.
- SLR파싱은 처음에 초기 스테이트 0으로 시작하며 이값을 스택에 넣어주고 , pointer를 통해 현재 아웃풋 파일의 토큰의 위치를 표시했습니다.

초기 스테이트 0

Pointer = 0

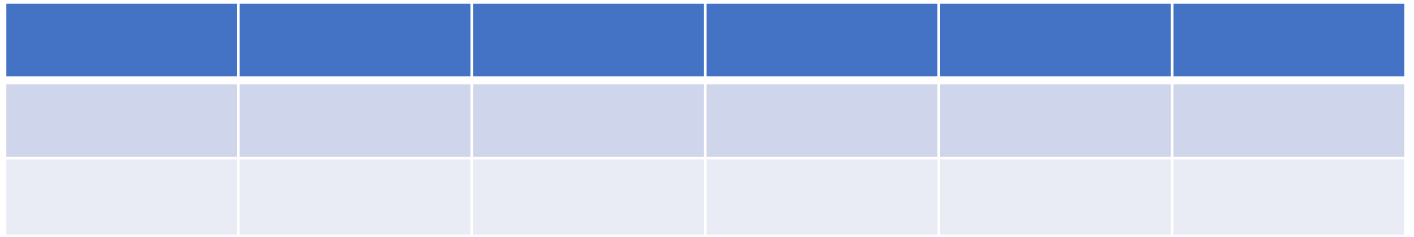


State Stack

Input Stack

5. 구현
(파싱 알고리즘(2))

- 이 파싱은 종료할때까지 계속 반복되는데요. 먼저 현재스테이트와 다음 인풋값을 확인하고 , 엑션테이블에 이값을 이용하여 LookUP을 진행합니다.
- 엑션테이블이 shift일 경우 그 스테이트와 넥스트 인풋을 각 스택에 추가하고 포인터를 한칸 옮겨줍니다.



Action Table

Lookup

현재스테이트, nextInput으로

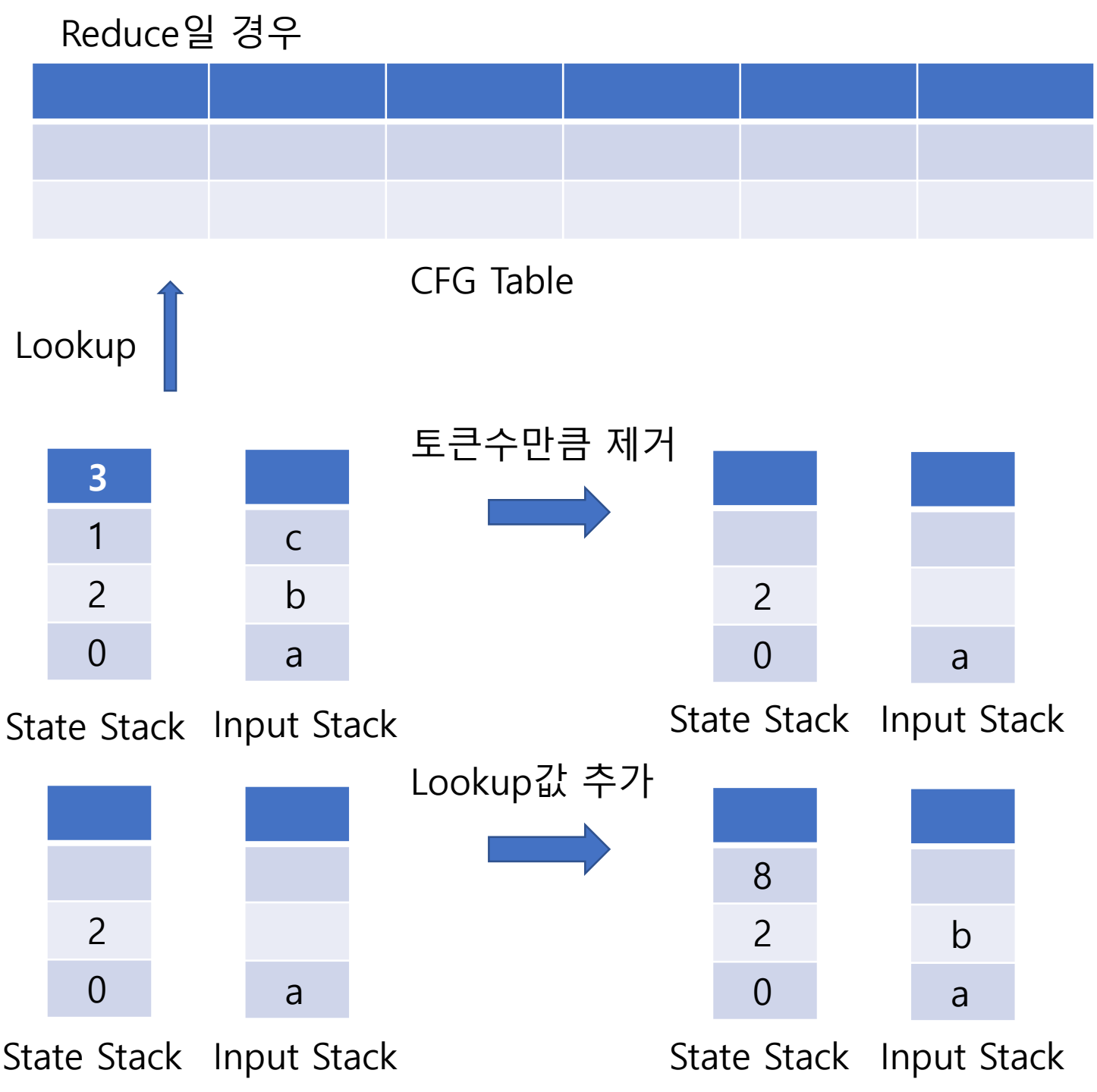
Shift일 경우 state 및 nextInput 추가

State Stack

Input Stack

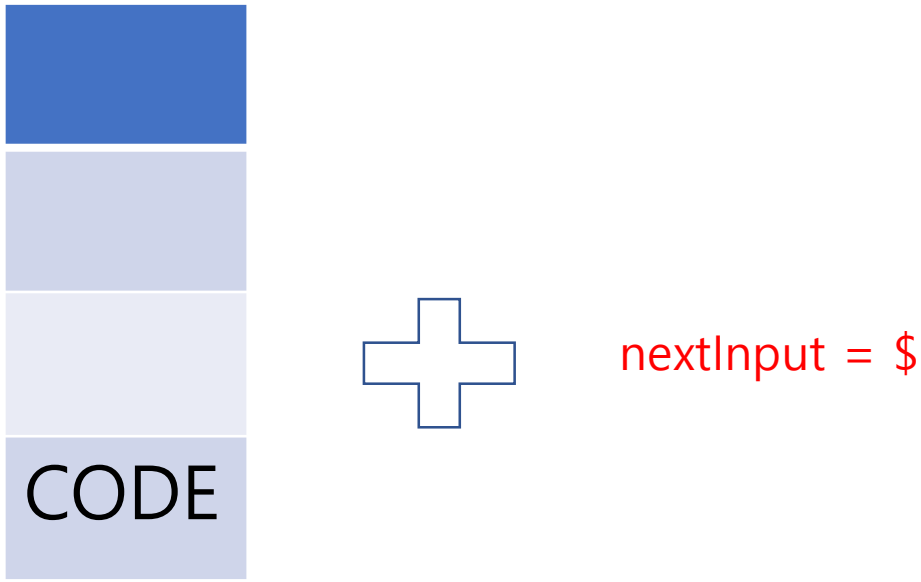
5. 구현
(파싱 알고리즘(3))

- 액션테이블이 reduce일 경우 그에 맞는 cfgTable을 확인하여 그 토큰수 만큼 스테이트스택과 인풋 스택에서 빼줍니다.
- 다시 goToTable을 lookUp하여 reduce된 값과 스테이트를 각각 스택에 추가해줍니다.
- 이 과정을 종료될때까지 반복합니다.



5. 구현
(종료조건(1))

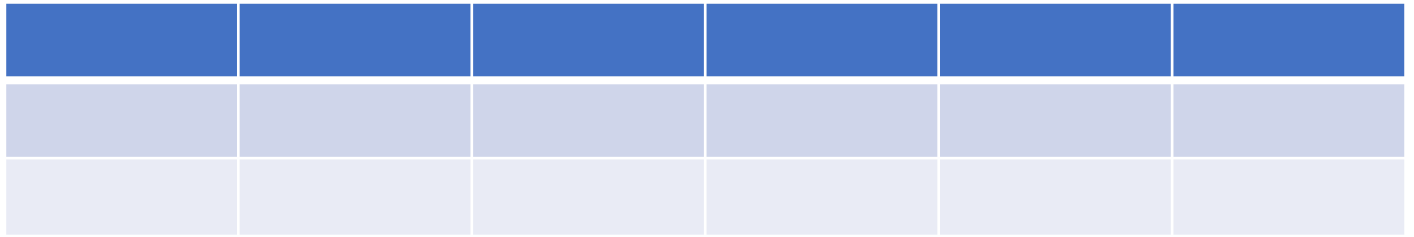
- 종료조건은 두가지가 있습니다.
- 하나는 모든 reducing이 적용되어 topInput이 CODE이고 또 nextInput 이 $ 이면 모든 토큰을 읽어 CODE로 돌아왔다는 의미이므로 올바른 코드입니다 하고 종료합니다.



모든 input처리했는데
cfg start state이면 종료

5. 구현
(종료조건(2))

- 두번째 종료조건은 현재 스테이트에서 nextInput으로 actionTable을 lookUP했는데 아무것도 존재하지 않다면, 이런 문법은 존재하지 않기 때문에, 현재스테이트와 다음 인풋을 알려주며 에러 원인을 알려줍니다.
- 또 에러가 일어난 라인넘버를 알려주어 , 어디서 문제가 일어났는지 알려줍니다.



Action Table

Look up 해도 아무것도 존재하지 않았을 때

문법이 존재하지 않으므로 에러 메시지 출력

5. 구현 (결과)

- 올바른 코드일경우

-> 올바른 코드일 경우 에러 없이 정상작동합니다

- 틀린 코드일 경우

-> 틀린 코드일 경우 에러의 원인인 현재 스테이트와 넥스트 인풋을 같이 보여주면서 어디서 발생했는지인 라인넘버도 같이 알려줍니다.

```
int foo(float x , float y){
    if(a>b){

    }else{
        c=a+b;
    }
    return 0;
}
int main(int a , int b){
    float s = 3.1;
    char c = "if else";
    while(a<b){
        a = a+1;
        b = b;
    }

    for(i=5 ; i<5 ; i=i+1){
        a=5;
    }

    int x = 3;
    return 0;
}
```

```
(base) useungjinuiMBP:syntax_analyzer wooseungjin$ python syntax_analyzer.py
옳은 코드입니다
```

```
int foo(float x , float y){
    if(a>b){

    }else{
        c=a+b
    }
    return 0;
}
```

```
(base) useungjinuiMBP:syntax_analyzer wooseungjin$ python syntax_analyzer.py
현재 스테이트: 23에서 다음 인풋: rbrace으로 가는 엑션테이블이 없습니다
6번째 라인에 에러가 있습니다
```