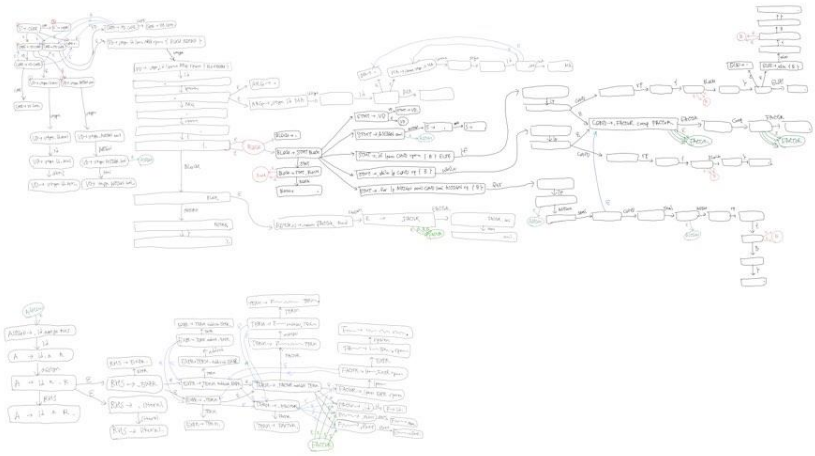
COMPILER PROJECT II: Syntax Analyzer

Documentation

20184690 장성현

1. NFA for recognizing viable prefixes



2. DFA Transition Table & SLR Parsing Table

DFA Transition Table & SLR Parsing Table은 워드에 입력하기에 공간이 부족해 엑셀 스프레드시트 파일을 함께 첨부했습니다.

3. How syntax analyzer works for validating tokens

SLR Parsing Table의 정보와 32개의 CFG 정보를 각각의 배열에 저장해놓고, 현재 state와 next input symbol이 배열의 정보와 일치하는 경우를 탐색하는 방식으로 구현하였습니다.

3-1. Data Structure

SLR Parsing Table은 Shift와 Reduce와 GOTO라는 문자열을 담는 2차원 배열 3개를 만들어 테이블의 정보를 담았습니다. shiftDecision() 함수와 reduceDecision() 함수는 이 배열들의 정보를 바탕으로 decision을 하게

됩니다.

State와 token은 모두 7글자 이하의 String 자료형을 가지고 있습니다.

State의 정보를 담는 데에는 Stack 구조를 사용하였고, 가장 마지막에 넣은 값인 Stack[top] 이 현재 state를 나타내게 됩니다.

CFG 정보의 구현은 다음과 같습니다.

CODE→VDECL CODE | FDECL CODE | ∈

VDECL→vtype id semi | vtype ASSIGN semi

ASSIGN→id assign RHS

FDECL→vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace

ARG→vtype id MOREARGS | ∈

MOREARGS→comma vtype id MOREARGS | ∈

BLOCK→STMT BLOCK | ∈

STMT→VDECL | ASSIGN semi

STMT→if lparen COND rparen lbrace BLOCK rbrace ELSE

STMT→while lparen COND rparen lbrace BLOCK rbrace

STMT→for lparen ASSIGN semi COND semi ASSIGN rparen lbrace BLOCK rbrace

ELSE→else lbrace BLOCK rbrace | ∈

RHS→EXPR | literal

EXPR→TERM addsub EXPR | TERM

TERM→FACTOR multdiv TERM | FACTOR

FACTOR→lparen EXPR rparen | id | num | float

COND→FACTOR comp FACTOR

RETURN→return FACTOR semi

위의 32개의 CFG에 순서대로 1부터 32까지 번호를 매겨서

좌변의 Non terminal은 배열 reduceLS에,

우변의 Non terminal, terminal 개수는 배열 reduceRS에 순서대로 저장했습니다.

(위에 써있지 않은 더미 cfg s'→ CODE 의 경우 0으로 번호를 매겼습니다)

3-2. Algorithms

Slr parsing table의 빈칸들까지 모두 배열에 저장했다면 decision을 검색하는데 걸리는 시간이 훨씬 줄어들었겠지만, 대신 빈칸들이 무수히 많기 때문에 메모리를 많이 차지합니다. 따라서 메모리를 적게 차지하는 편을 택했습니다.

1. **Shift를 하는 조건에 해당하는지 확인** (데이터 110개, 같은 state에 대해 최대 5개)
2. **Reduce를 하는 조건에 해당하는지 확인** (데이터 131개, 같은 state에 대해 최대 8개)
3. **2를 만족하면 GOTO는 어디로 해야 하는지 확인** (데이터 61개, 같은 state에 대해 최대 4개)

   1~3을 Token의 개수만큼 반복 (1을 만족하면 2~3 건너뜀)

이 알고리즘으로 토큰 하나와 state 하나에 대한 decision을 구할 때 최악의 경우를 어림해보면 (110 + 5) + (131 + 8) + (61 + 4) = 319번째로 검색해서 찾는 경우입니다. Token n개의 소스코드에서 최악의 검색 횟수를 어림해보면 319n이고, 이 알고리즘의 시간복잡도는 O(n) 입니다.

3-3. 기존의 Lexical Analyzer에서 변경한 부분

-모든 token이름을 대문자에서 소문자로 변경

-Integer를 num으로 변경

-real을 float로 변경

-Arithm을 addsub와 multdiv로 분리

-lbrkt, rbrkt를 lbrace, rbrace로 변경

3-4. 제시된 CFG와 다르게 코드된 부분

Non terminal인 MOREARGS의 길이가 너무 길어서 MOREA 로 표기하였습니다.

3-5. Working Examples

첫번째 과제에서는 입력 파일에 줄바꿈이 있어도 lexical analyzer가 token을 인식했는데, 컴파일 환경이 바뀌어서 그런지 '₩n' 인식에 문제가 생겨서 불가피하게 입력파일을 줄바꿈 없이 한 줄로 된 소스 코드로 테스트했습니다.

Test.c

bool foo(int a) { a = 1; if(a == 1) { a = 5; } else{a = a - 4;} return 0; }

Test.out (lexical analyzer 후)

```
vtype       bool
id      foo
lparen
vtype       int
id      a
rparen
lbrace
id      a
assign      =
num     1
semi
if
lparen
id      a
comp        ==
num     1
rparen
lbrace
id      a
assign      =
num     5
semi
rbrace
else
lbrace
id      a
assign      =
id      a
addsub      -
num     4
semi
rbrace
return
num     0
semi
rbrace

```

Test.out (syntax analyzer 후)

```
Accepted
```

Test2.c

int comp(int a, int b) { while (a != b) {a = b * 2; } return 0; }

Test2.out (lexical analyzer 후)

```
vtype       int
id      comp
lparen
vtype       int
id      a
comma
vtype       int
id      b
rparen
lbrace
while
lparen
id      a
comp        !=
id      b
rparen
lbrace
id      a
assign      =
id      b
multdiv     *
num     2
semi
rbrace
return
num     0
semi
rbrace
```

Test2.out (syntax analyzer 후)

```
Accepted
```

Test3.c

int main() { int i; for(i = 0; i < 5; i = i + 1) { a = 120 / i; } return 0; }

Test3.out (lexical analyzer 후)

```
vtype       int
id      main
lparen
rparen
lbrace
vtype       int
id      i
semi
for
lparen
id      i
assign      =
num     0
semi
id      i
comp        <
num     5
semi
id      i
assign      =
id      i
addsub      +
num     1
rparen
lbrace
id      a
assign      =
num     120
multdiv     /
id      i
semi
rbrace
return
num     0
semi
rbrace
```

Test3.out (syntax analyzer 후)

```
Accepted
```

error.out (syntax analyzer 전) - 에러 출력을 위해 임의로 만든 파일

```
vtype       bool
id      foo
lparen
vtype       int
id      a
rparen
lbrace
id      a
assign      =
num     1
semi
if
lparen
id      a
comp        ==
num     1
rbrace
```

error.out (syntax analyzer 후)

에러가 발생한 state와 next input symbol 출력.

```
Error occured in State:20, Next input symbol:rbrace
```