**COMPILER PROJECT 2023**

Bottom up parser Implementation

Single LR Table Using(non-ambiguous)

|  |  |
| --- | --- |
| Professor | 김효수 |
| Department | Software |
| TEAM | Team-project 51 |
| Student ID | 20210189 / 20210115 |
| NAME | 한동희 / 이은학 |

**1. 프로젝트 목적**

이번 과제는 Single LR Table 이용해 상향식 구문 분석기를 구현하는 것을 목표로 합니다.

실제로 우리가 사용하는 while if else는 물론이고 variable type 지정하고 (){} 와 같이

블록이나 Parameter를 묶어주는 역할을 하는 여러 terminal 문자를 입력 값으로 취하면

그에 따라 컴파일러가 Syntax 분석 과정에서 올바른 문법인지 판단하는 단계에서 사용하는

상향식 파서 구현을 목표로 합니다.

주어진 CFG G를 기반으로 구현해야 하는데 문제가 있습니다.

텍스트, 화이트보드, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 직사각형, 화이트보드이(가) 표시된 사진

자동 생성된 설명

이와 같이 Shift 후state를 바꿀지 27번 규칙으로 감축될지 모호성이 발생됩니다.

이를 해결하기 위해서 생성규칙을 분석한 바

1. EXPR → EXPR addsub EXPR | EXPR multdiv EXPR
2. COND → COND comp COND 의 생성 규칙에 모호성을 발견했습니다.

먼저 첫번째의 생성규칙에서는 addsub/multdiv 기준으로 좌우 대칭이기 때문에

명확한 하나의 Parse Tree를 형성하는 것이 아닌 하나의 input string에 대해

여러가지 Parse Tree가 형성되기 때문에 문제가 발생한다. 이는 add sub이 mult div보다

일반적으로 우선순위가 낮기 때문에 multdiv가 좌측우선 감축되기 때문에 왼쪽 오른쪽

논터미널을 두고 구분 지어서 구현하는 것을 구상!

>>

좌측 우선 탐색이라고 생각하고 좌측에서 먼저 multdiv가 포함된 substrig을 먼저 감축시킴.

EXPR -> HEXPR EXPR' // 높은 우선순위를 가지는 HEXPR Non-terminal 하나 만들고 구별(좌측)

EXPR' -> addsub HEXPR EXPR' | '' // 오른쪽 EXPR’(우측)에서 addsub을 다룰 Non-terminal

HEXPR -> F HEXPR' // 인수가 일단 먼저 나와야 하기 때문

HEXPR' -> multidiv F HEXPR' | '' // 좌측 인수에 multdiv 후 인자가 붙을 수 있게 만들고 우재귀

F -> num | id | lparen EXPR rparen // F 는 인자를 의미하고 3가지 경우로 구분

>> 이로서 우선순위를 multidiv 포함된 substrig이 가지게 모호성을 제거.

두번째의 경우는 간단하게 좌측결합 법칙으로 설정해서 모호성을 제거 하였음.

COND -> boolstr COND'

COND' -> comp COND | ''

>>

또한, 모호성을 문법 제거하기 위한 방법을 찾아봤는데,

모호성은 어떤 생성규칙으로 감축될지 모호한 reduce-reduce Conflict

다른 상태로 이동할지 shift-reduce Conflict 이 존재하는데 주어진 문제를

shift-reduce Conflict인데 여기서 규칙과 입력 기호의 우선순위를 비교하여

생성 규칙의 우선순위가 높으면 감축을 선택하고, 그렇지 않으면 이동을 선택

하게 하면 되는데 여기서 사실 이번 과제 해결 아이디어가 나온다.

우선순위가 높아서 감축을 선택한다는 말은 좌측결합법칙을 따르는 분석기

입장에서는 그것을 먼저 계산한다는 의미와 같다. 따라서 좌측에서 먼저

multdiv를 만들면 되겠다고 생각하게 되었다.

그런데 이렇게 모호한 문법이 주어지는 이유는 모호한 문법을

모호하지 않은 문법이 만들어졌을 때 생기는 단일 생성 규칙을 없앨 수 있기 때문인데

이는 생성규칙이 늘어남을 의미하고 이는 곧 파싱표가 더 커지게 되고 구문분석 시간이 늘어남을

의미하는데 좀 더 방대한 생성 규칙에 대해서는 더더욱 최적화 하는 것이 중요한 점에서 모호한 문법으로 먼저 파싱표를 만들고 하나하나 충돌을 제거해서 파싱표와 생성규칙을 결정하는 것이 좀더 구문분석기의 성능을 향상시키는 방법이 될 수 있음을 느꼈다.

마지막으로 dummy 생성규칙 SCODE → CODE 추가해 주었다.

**2. 구현 설명**

Txt 파일에서 testing file을 만들어 INPUT을 받으면 되겠다고 생각하게 되었고,

벡터로 string or int를 저장하고 stack에 이를 push, pop하면 되겠다 생각해서

#include <fstream>

#include <vector>

#include <stack>

했고, Table 을 먼저 Reduce, Shift, Error, Accept를 구분을 생각했고,

Error는 처음 초기화 단계에서 0으로 Reduce는 감축이기 때문에 – 즉 음수인 경우

Shift을 양수의 경우를 생각해서 int형 2차원 Acction Table 을 하나하나 Machine에서 입력해줌.

GOTO또한 0으로 우선 초기화 하고 각각을 입력해 주었다.( Accept는 999로 구분해서 넣어줌)

<<준비단계

처음에는 INPUT을 받을 파일이 open되었는지 판단하고 넘어간다.

Int 형 값을 저장하는 stack class 정의하고 string형 word 정의하고

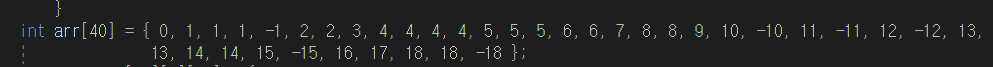
벡터에 각각의 INPUT에서 받는 Terminal을 받아줄 Vector class 정의한다.

(파일에서 string 발견하면 넣어준다.)

Non-terminal을 0~18까지 Terminal 을 19~40까지에 eof는 입실론을 표현하기 위해

새로 도입해서 1차원 string 배열에 넣어주었다.

또한, 생성규칙의 의 index를 찾기 위한 int 형 arr배열에 -를 붙여서 구분함.

>>

>>

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

INPUT 각각의 string을 rules string 1차원 배열에 정해 놓은 index의 일치 여부를 2중 for문을 이용해서 판별하고 해당 index를 새로운 int형 vector에 number을 넣어준다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

3차원 배열에 생성규칙 기준으로 20개 index주고 같은 Non-terminal에 감축될 수 있는 생성규칙이

최대 4개라서 그 다음 index에 4 마지막으로 생성규칙에서 생성되는 non-terminal & terminal의

개수가 최대 10개인 것을 확인하고 index에 10을 넣어 CFG 3차원 배열을 초기화 함.

>>본격적 코드 구현

Splitter의 위치를 알려주는 변수/ 변하는 state 넣어줄 state 변수를 초기화 해줌.

(처음의 상태는 항상 0에서부터 시작하므로 0으로 초기화)

또한, splitter의 바로 왼쪽을 탐색이 action table에서 Shift나 Reduce 찾기 위한

Spl, acttiontable에서 Shift한 후 변경되는 state를 저장할 x 정의함.

Stack의 처음 상태를 나타내기 위해 push(0) 해 주어 초기화 함.

Splitter의 위치를 업데이트를 위한 size , CFG tree 의 값을 받아오기 위해 정의한다.

스크린샷, 폰트, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명

처음 분기에서 acc을 나타내는 999가 acctiontable에서 나오고 word\_numd에서 CODE의 number

1이 나오면 SCODE -> CODE 유일하게 감축이 결정되므로 이때 INPUT ACCEPT ! 출력한다.

여기서 rules 배열의 해당 번호에 String을 출력하기 위한 for문을 넣었다.

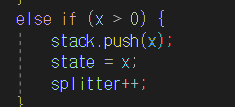
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

상태 값을 저장하는 x 값이 0보다 큰 shift 상황에서는 stack에 해당 상태 x를 push해주고

현재 상태를 x로 업데이트 한다. 그리고 shift 했기 때문에 splitter가 한 칸 이동했기 때문에

1 더해준다.



이제 음수인 상황에서는 그대로 모든 word\_num에 남아있는 number를 이용해 terminal 출력.

이제 입실론을 판단 시켜주는 arr[x]를 x에 업 데이트 해서 x를 이용해서 조건문 분기한다.

분기문에 음수 조건을 만족에서 들어가면 바로 x를 다시 양수로 변환해주고

Word\_num에 splitter 위치에 x 로 insert해주고 state는 stack top의 가장 최신 상태를

이용해 GOTO를 찾아 state 받고 stack에 업데이된 state를 push한다.

상태가 바뀌면서 shift 했기 때문에 splitter++ 해준다.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

이제 마지막 단계이다.

CFG 에서 x로 해당 생성 규칙 쪽으로 이동해서 2중 for 문으로

끝에서부터 

와 같이 만약 3(i)이면 {41}로 접근해서 0(i)({2,1})까지 탐색해가기 위해

CFG\_tree[x][i][0]의 값이 0이면 비어있는 생성규칙 set이기 때문에 continue로

건너뛰고 최대원소 개수가 10개 이므로 2중 for문에서 안쪽에서 끝에서 해당 terminal

이나 non-terminal의 number를 받아와서 k에 저장하고 만약 0이면 terminal/non-terminal

이 없다는 의미이기 때문에 continue로 건너뛰고, k 값 생성규칙의 terminal or non-terminal

의 번호가 word\_num의 한 칸 옆의 number와 같으면 size 값에 1더하고 다시 for문이 돌아가고

만약 같지 않으면 for문을 빠져나온다.

for문을 빠져나오기 전까지 돌면서 size가 알맞게 up date 되었으면 이것을 이용해

size가 0이면 처음부터 일치하지 못해 나온 것이기 때문에 if문에 들어가지 않고

만약 size가 1 이상이면 한 번 이상 일치 했기 때문에 if문 안에서 size의 index보 시작해

줄어가면서 word\_num의 값을 지운다. 지움과 동시에 splitter는 한 칸 뒤로 밀려나기 때문에

1 빼주고 stack에서도 상태 하나가 지워지게 된다. Pop을 이용해 빼낸다.

감축을 다 시키고 for문을 나오면 insert를 이용해 해당 생성 규칙을 번호를 word\_num에 넣어준다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

마지막으로 state를 GOTOtable을 이용해 stack top의 최신 상태의 생성규칙 x로

State를 업데이트 해 stack에 push해 준다. State가 업데이트 되었으므로 shift한 셈이기

때문에 splitter++ 해준다.

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

1. **INPUT Testing**

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명**

간단하게 출력해 보았다.

좀 더 복잡하게 구성해서 testing 해 보겠다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

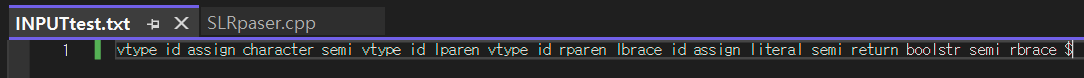
텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

좀 더 복잡하게 만들어서 넣어보았다.

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 평행, 번호이(가) 표시된 사진

자동 생성된 설명



Final Test

vtype id assign character semi vtype id lparen vtype id rparen lbrace id assign literal semi return boolstr semi rbrace vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace rbrace else lbrace rbrace return literal semi rbrace $

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진

자동 생성된 설명

아주 복잡한 INPUT도 올바르게 출력되는 것을 확인하였다.

이로서 올바르게 Testing 되는 것을 확인하였다. 구현단계에서 한방에 완벽할 수 없다는 걸 느꼈다.

일일이 찾아서 넣은 값이 잘못되거나 빠진 것도 찾아가면서 코드를 수정했다. 데이터가 좀 더 방대해지면 컴퓨터가 직접 데이터 관리 필요성이 절실하다 생각하게 되었다.

Syntax 분석기를 만드는 프로젝트를 해보았는데 재미있고 색달랐다.