NLP hw#2 : earley parser 제작

120150241 김상근

1. 개발 환경

C++ 언어와 표준 라이브러리 (Standard Template Library, STL) 을 이용하여 제작하였다. 코드는 unix 기반 컴퓨터에서 작성하였으나, 윈도우 환경의 Visual studio 2010 에서 잘 동작하는 것을 확인하였다. Unix 계열 컴퓨터에서는 make 명령어로 컴파일 할 수 있고, Visual studio 2010 에서는 새로운 프로젝트에 a.cpp, utils.h 와 grammer.txt, input.txt 를 추가하면 컴파일 할 수 있다.

2. 구조체 설명

```
struct Grammer {
    multimap<string,deque<string> > g;
    multimap<deque<string>,string> bg;
    Grammer() {}
    void insert(svs curGrammer) {
        g.insert(curGrammer);
        bg.insert(pair<deque<string>,
string>(curGrammer.second,curGrammer.first));
    }
    void printG() {
        for ( map<string,deque<string> >::iterator
it=g.begin();it!=g.end();it++ ) {
            fprintf(stdout,"%s -> ",it->first.c_str());
            for ( int i = 0 ; i < (int)it->second.size() ; i++ ) {
                fprintf(stdout,"%s ",it->second[i].c_str());
            }
            fprintf(stdout,"\n");
        }
    }
    void printBg() {
        for ( multimap<deque<string>,string>::iterator
it=bg.begin();it!=bg.end();it++ ) {
            for ( int i = 0 ; i < (int)it->first.size() ; i++ )
                fprintf(stdout,"%s ",it->first[i].c_str());
            fprintf(stdout,"-> ");
            fprintf(stdout,"%s\n",it->second.c_str());
        }
    }
    vector<deque<string> > findRhsUsingLhs(string lhs) {
        vector<deque<string> > ret;
        pair<multimap<string,deque<string>
>::iterator,multimap<string,deque<string> >::iterator> foundGrammer =
g.equal_range(lhs);
        for ( multimap<string,deque<string> >::iterator
it=foundGrammer.first;it!=foundGrammer.second;it++ )
            ret.push_back(it->second);
        return ret;
    }
    deque<string> findLhsUsingRhs(Words rhs) {
        deque<string> ret;
```

```
pair<multimap<deque<string>,string>::iterator,multimap<deque<string>,
string>::iterator> foundGrammer = bg.equal_range(rhs);
        for ( multimap<deque<string>,string>::iterator
it=foundGrammer.first;it!=foundGrammer.second;it++ )
            ret.push_back(it->second);
        return ret;
    }
};
```

Grammer 구조체는 multimap<string,vector<string>> 컨테이너 두 개를 갖고 있는데, 이는 문법이 lhs 일 때 rhs 리스트를 구할 수도 있어야하고, rhs 리스트일 때 lhs 품사도 구할 수 있어야 하기 때문이다. 또한 key 에 해당하는 품사가 여러개 존재할 수도 있기 때문에 map 이 아닌 다중 키를 제공하는 multimap 을 사용하였다. 편의성을 위하여 lhs 로 rhs 를 찾는 findRhsUsingLhs 함수와 rhs 로 lhs 를 찾는 findLhsUsingRhs 함수를 추가했다.

```
struct State {
    int start;
    int end;
    string constituent;
    Words found;
    Words next;
    vector<State*> child;
    State(){}
    State(State* state):
        start (state->start), end (state->end), constituent (state-
>constituent), found (state->found), next (state->next), child
(state->child) {}
    State(int _start,int _end,string _constituent,Words _found,Words
_next):
        start (_start), end (_end), constituent (_constituent), found
(_found), next (_next), child (vector<State*>()){}
    State(int _start,int _end,string _constituent,Words _found,Words
_next,vector<State *> _child):
        start (_start), end (_end), constituent (_constituent), found
(_found), next (_next), child (_child){}
    void printState(FILE* fp,Words words) {
        for ( int i = 0 ; i < (int)words.size() ; i++ ) {
            if ( i == end ) fprintf(fp,"+ ");
            fprintf(fp,"%s ",words[i].c_str());
        }
        fprintf(fp,"%s\n",end==(int)words.size()?"+":"");
        fprintf(fp,"[");
        fprintf(fp,"%d, %d, [%s], [",start,end,constituent.c_str());
        for ( int i = 0 ; i < (int)found.size() ; i++ )
            fprintf(fp,"%s%s",i==0?"":" ",found[i].c_str());
        fprintf(fp,"], [");
        for ( int i = 0 ; i < (int)next.size() ; i++ )
            fprintf(fp,"%s%s",i==0?"":" ",next[i].c_str());
        fprintf(fp,"] ");
        fprintf(fp,"]\n");
    }
};
```

State 구조체는 [start,end,constituent,[found],[next]] 그리고 found 들에 해당하는 child 로 구성되어있다. 각각의 found 는 어떤 state 에서 왔는지를 child 에 기록함으로써 백트랙킹을 통하여 최종 파스 트리를 출력할 수 있다.

3. 프로그램 플로우 설명

먼저 grammer.txt 에서 문법을 읽어온 후, 파싱하여 Grammer 구조체를 구성한다. 그 후 input.txt 에서 파싱할 문자열을 입력받는데, input.txt 에 여러개의 테스트 문자열이 있다고 가정하고 개행을 기준으로 구분하여 입력받는다. 그리고 파싱할 테스트 문자열마다 파싱하여 결과를 output.txt 에 출력한다.