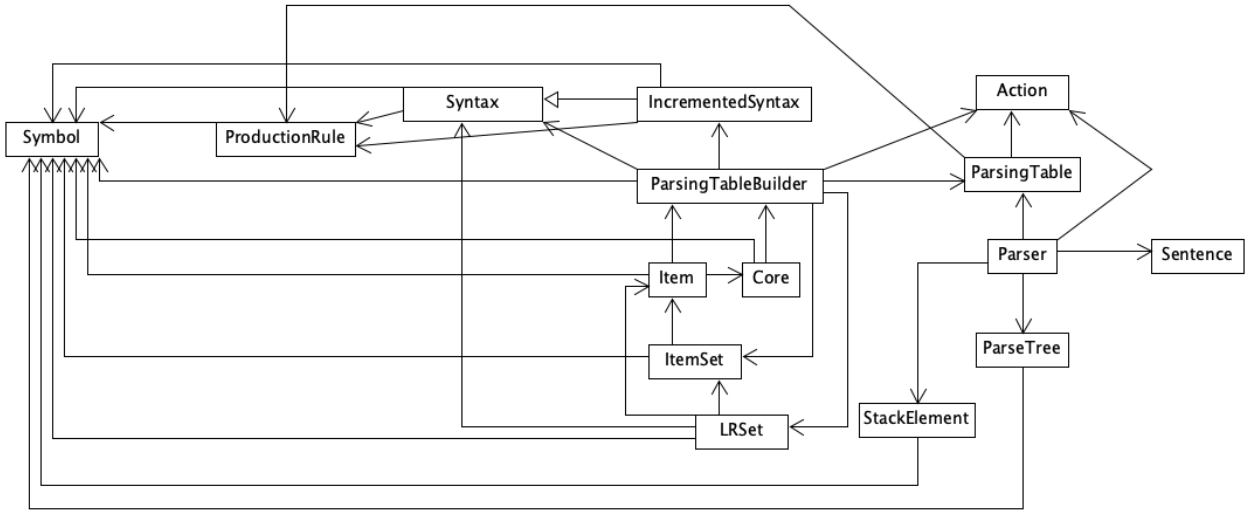
201946100 이종휘



각 클래스들에 대한 설명

Symbol
기본 심볼이다. 터미널 심볼, 논터미널 심볼 또는 엡실론이 될 수 있다.
ProductionRule
생성규칙이다. left hand side 와 right hand side 가 있다. 코드 내에서는 from 과 to 이다.
Syntax
문법이다. 심볼들과 생성규칙들을 담고있다. 이 클래스에서 문법파일 인식, first, follow 계산이 수행된다.
IncrementedSyntax
증가문법이다. Syntax 클래스로부터 상속된 클래스이다.
ParsingTableBuilder
parsing table 을 생성하는 클래스이다. LR(0,1)을 생성하고 그로부터 parsing table 을 생성한다.
Core
dot 이 포함된 생성규칙이다. dot 은 개념적으로만 존재하며, production rule 의 to 가 beforeDot, afterDot 으로 나뉜다.
Item
core 와 lookahead 를 포함하는 항목이다. LR(0)에서는 lookahead 가 사용되지 않는다.
ItemSet
closure, goto 에 의해 구해진 item set 이다. goto graph 정보를 담고 있다.
LRSet
정규항목집합이다. closure 와 goto 는 이 클래스에서 계산한다.
ParsingTable
파싱표이다. 파싱표의 파일 저장 및 불러오기는 이 클래스에 정의되어 있다.
Action
파싱표 안에 기록되는 action 이다.
Sentence
문장이다. 파일을 읽어들여 공백문자 단위로 token 들을 읽어들인다.
Parser
구문분석기이다. 주어진 파싱표에 따라 구문분석을 수행한다.
StackElement
Parser 에서 stack 에는 상태(int)와 심볼(Symbol)이 들어간다. 이 타입들을 묶어주기 위한 wrapper 이다.
ParseTree
구문분석에 의해 생성된 파스 트리이다.

구현 방법

우선 위에 보이는 클래스 다이어그램을 코드를 작성하기 전에 먼저 그렸다. 그 후에 그림에 따라 코드를 작성하였다. 맨 처음 주어진 문법을 읽어들이고 first 와 follow 를 계산하기 위해 Symbol, ProductionRule, Syntax 클래스를 작성했다. 그 후 파싱표를 생성하기 위하여 IncrementedSyntax, ParsingTableBuilder, Core, Item, ItemSet, LRSet, ParsingTable, Action 클래스를 작성했다. 마지막으로 파싱표에 따라 구문분석을 진행하기 위해 Parser, StackElement, ParseTree 클래스를 작성하였다.

프로그램 개요

make 로 컴파일을 하고 나면 실행 가능한 jar 파일은 jars 디렉토리에 4 개 생성된다.
java -jar jars/FitstFollowCalculator.jar <syntax file>
이 프로그램은 매개변수로 syntax 파일의 이름을 받아 그 파일의 symbol 들의 first, follow 들을 구하는 프로그램이다.
java -jar jars/ParsingTableBuilder.jar <SLR|CLR|LALR> <syntax file> [table file] [DEBUG]
이 프로그램은 주어진 syntax 파일에 대해 SLR 또는 CLR 또는 LALR 파싱표를 구하는 프로그램이다. 3 번째 매개변수를 저장할 table file 의 이름으로 하면 파싱표를 파일에 저장한다. 4 번째 매개변수에 DEBUG 를 입력하면 LR 항목도 같이 출력한다. LALR 은 구현에 실패하였다.
java -jar jars/ParseTreeReader.jar <table file>
이 프로그램은 주어진 table 파일을 보기 좋게 출력해준다.
java -jar jars/Parser.jar <table file> <sentence file> [DEBUG]
이 프로그램은 주어진 파싱표로 주어진 문장 파일의 내용을 구문분석하는 프로그램이다. 구문분석에 성공하면 파스트리를 출력한다. 3 번째 매개변수에 DEBUG 를 입력하면 각 단계별 stack, buffer 정보가 출력된다.

실행결과

아래 코드를 참조하여 report 디렉토리에서 결과를 확인할 수 있다.

```
java -jar jars/FirstFollowCalculator.jar data/example1.syntax > report/example1_first_follow.txt
java -jar jars/FirstFollowCalculator.jar data/example2.syntax > report/example2_first_follow.txt
java -jar jars/FirstFollowCalculator.jar data/example3.syntax > report/example3_first_follow.txt
java -jar jars/FirstFollowCalculator.jar data/example4.syntax > report/example4_first_follow.txt

java -jar jars/ParsingTableBuilder.jar SLR data/example1.syntax data/example1.table DEBUG >
report/example1_SLR_table.txt
java -jar jars/ParsingTableBuilder.jar CLR data/example1.syntax data/example1.table DEBUG >
report/example1_CLR_table.txt
java -jar jars/ParsingTableBuilder.jar CLR data/example2.syntax data/example2.table DEBUG >
report/example2_CLR_table.txt
java -jar jars/ParsingTableBuilder.jar SLR data/example3.syntax data/example3.table DEBUG >
report/example3_SLR_table.txt
java -jar jars/ParsingTableBuilder.jar CLR data/example3.syntax data/example3.table DEBUG >
report/example3_CLR_table.txt
java -jar jars/ParsingTableBuilder.jar SLR data/example4-2.syntax data/example4.table DEBUG >
report/example4_SLR_table.txt
java -jar jars/ParsingTableBuilder.jar CLR data/example4-2.syntax data/example4.table DEBUG >
report/example4_CLR_table.txt

java -jar jars/Parser.jar data/example1.table data/example1-1.sentence DEBUG > report/example1-1_parseTree.txt
java -jar jars/Parser.jar data/example1.table data/example1-2.sentence DEBUG > report/example1-2_parseTree.txt
java -jar jars/Parser.jar data/example1.table data/example1-3.sentence DEBUG > report/example1-3_parseTree.txt
```

```
java -jar jars/Parser.jar data/example1.table data/example1-4.sentence DEBUG > report/example1-4_parseTree.txt
java -jar jars/Parser.jar data/example1.table data/example1-5.sentence DEBUG > report/example1-5_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-1.sentence DEBUG > report/example2-1_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-2.sentence DEBUG > report/example2-2_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-3.sentence DEBUG > report/example2-3_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-4.sentence DEBUG > report/example2-4_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-5.sentence DEBUG > report/example2-5_parseTree.txt
java -jar jars/Parser.jar data/example2.table data/example2-6.sentence DEBUG > report/example2-6_parseTree.txt
java -jar jars/Parser.jar data/example3.table data/example3-1.sentence DEBUG > report/example3-1_parseTree.txt
java -jar jars/Parser.jar data/example3.table data/example3-2.sentence DEBUG > report/example3-2_parseTree.txt
java -jar jars/Parser.jar data/example3.table data/example3-3.sentence DEBUG > report/example3-3_parseTree.txt

java -jar jars/ParsingTableBuilder.jar CLR data/example4-4.syntax data/example4.table
java -jar jars/Parser.jar data/example4.table data/example4-1.sentence DEBUG > report/example4-1_parseTree.txt
java -jar jars/Parser.jar data/example4.table data/example4-2.sentence DEBUG > report/example4-2_parseTree.txt
```

참고 1: example2 의 경우 SLR 에서 충돌이 발생하여 파싱표가 생성되지 않는다.
참고 2: example4 의 경우 Stmt ==> if (Bool) Stmt || if (Bool) Stmt else Stmt 에서 first 의 충돌이 발생하여 LL 문법이 아니다. 즉, 파싱표가 생성되지 않는다.이에 따라 exmple4-2.syntax 파일을 새로이 생성하였다. 이 문법은 Stmt ==> if (Bool) Stmt else Stmt 규칙이 삭제된 문법이다.
참고 3: example4-2.syntax 의 경우 주어진 문장 example4-1.sentence 를 인식하지 못한다. { 다음의 int 를 인식하지 못한다. 이에 따라 example4-4.syntax 를 만들었다. 이는 while 문 안의 do-while 문을 인식하지 못한다. Stmt 블럭을 인식하는데 문제가 있다고 판단되어 example4-2.sentence 를 만들어 테스트 해보았다.