REPORT

Compiler Project 02

20133950

공찬형



1. 개요

프로그래밍 언어를 기계 언어 혹은, 그 하위 언어로 치환하거나, 그 언어를 이해하기 위하여, LR 파싱, LL 파싱과 같은 문법 해석 기법을 사용한다. 본 과제는 LR 파싱의 과정들 중에서, LR 파싱 테이블 구성을 위한 단계중 C0 Iteration을 구성하는 과정을 Java 언어를 통하여, 시각화 하였다.

1. 프로그램 동작 원리

LR 파싱에서, C0 Iteration을 구성하기 위하여, 다음과 같은 과정을 필요로 한다.

1. Grammar의 Production rule을 ‘.’ (Dot notation) 을 통하여, 현재까지 진행된 String을 표기할수 있게 만든다.
2. 파싱 과정의 모호함을 없애기 위하여, Augmented rule (S-> E : Initial Start symbol) 를 추가한다.
3. 추가된 Augmented rule에 dot notation을 추가하고, 그 dot notation 바로 뒤에 위치한 symbols, 즉 다음에 처리해야 할 String의 Closure를 구하여, 가능한 모든 경우의 수에 대응할수 있도록 한다.
4. 위의 과정을 I0 이라 정의하고, I0에서 다음에 처리할수 있는 symbol을 인식하고, I0에서 그 symbol을 만나면, dot notation을 한 칸 이동시키며, 그 dot notation의 closure를 포함하는 집합을 구성하고, 새로운 C0 Iterator를 구성한다.
5. 만약 4번의 과정에서, 구성된 C0 Iterator가 이미 구성되었다면, 그것을 생성하지 않는다.
6. 4번과 5번의 과정을 생성된 C0 Iterator를 순회하여 진행하고, 더이상 생성되지 않는다면, 종료한다.
7. Program Structure Overview
   1. Main.java

User input을 받기 위한 Basic main class.

* 1. Production\_rule.java

File에서 정의된 grammar의 production rule을 구성하는 structure class.

* 1. Dot\_rule.java

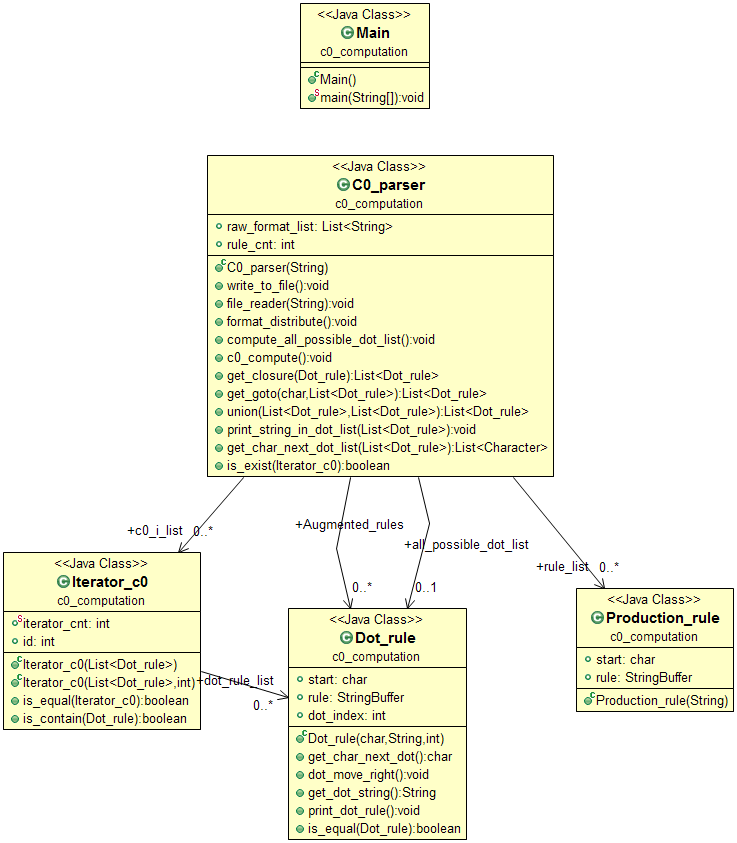
Iterator\_c0를 구성하는 dot notation이 포함된 production rule을 구성하는 structure class

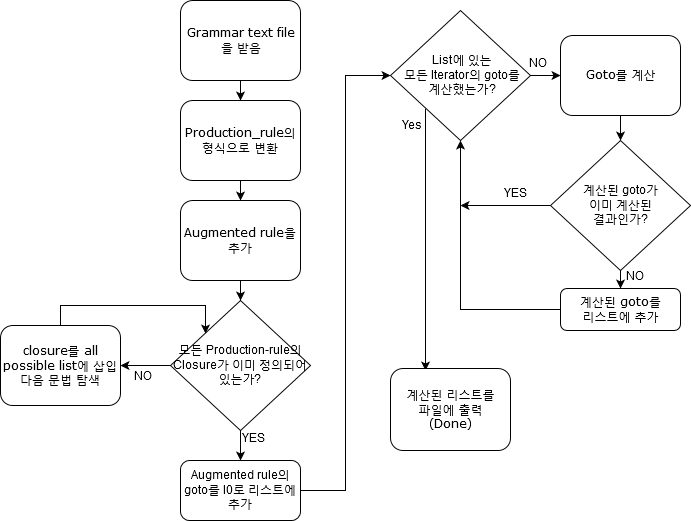
* 1. Iterator\_c0.java

C0를 구성하는 Iterator. dot\_rule의 list를 포함하는 structure class.

* 1. C0\_parser.java

C0을 구성하는 process를 담당하는 class. File을 Production\_rule로 처리하고, 그에 dot\_notation을 추가하여 C0의 모든 Iterator를 계산한다. 이후 write\_to\_file 프로시저를 통하여 Output을 출력한다.

1. Class diagram
2. Flow chart



1. Descriptions

*-Production\_rule.java*

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameter | Output | description |
| Production\_rule | String rule | - | 기본 생성자.  Input 문자열을, 시작문자와 오른쪽 규칙으로 분할하여 저장한다. |

*-Dot\_rule.java*

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameter | Output | description |
| Dot\_rule | char start, String rule,  int dot index | - | 기본 생성자.  시작 문자, 규칙, dot-notation의 위치를 참조하여, 객체를 생성한다. |
| get\_char\_next\_dot | - | 객체의 dot\_notation 바로 뒤에있는 symbol을 리턴. | Dot\_notation이 있는 dot\_rule의 closure를 계산하기 위하여 사용된다. |
| dot\_move\_right |  | - | Dot\_notation을 한칸 이동시켜, 현재 symbol을 처리한다. |
| get\_dot\_string |  | Dot notation이 포함된 규칙의 문자열을 리턴 (S->.E+T) | Dot\_notation이 포함된 현재 객체의 규칙 String을 리턴한다. 테스트, 혹은 추후 파일에 작성할 때 사용된다. |

*-Iterator\_c0*

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameter | Output | description |
| Iterator\_c0 | List<Dot\_rule> dot\_rule\_list | - | 기본 생성자.  Iterator가 포함할수 있는 dot\_rule의 리스트를 저장한다. |
| is\_equal | Iterator\_c0 operand | 같은 객체라면, true. | 리스트를 순회하면서, 새로운 Iterator가 만들어졌는지를 확인하기 위하여, 이전 생성여부를 확인하기 위하여 사용된다. 포함하는 규칙이 동일하다면, true를 리턴한다. |
| is\_contain | Dot\_rule rule | 현재 객체가 rule을 포함하고 있다면 true. | 현재 객체의 리스트가 parameter 규칙을 포함하고 있는지 판단한다. Iterator\_c0 단위 동치비교에 사용된다. |

*-C0\_parser*

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameter | Output | description |
| C0\_parser | String filepath | - | 기본 생성자.  Filepath를 기반으로, 멤버 함수를 이용하여 객체가 가지고있는 list의 내용을 채운다. |
| write\_file\_to\_list | String filename | - | 생성된 객체가 가지고있는 C0을 기반으로, input의 filename을 이름으로 하는 파일을 생성한다. |
| file\_reader | String filepath | - | 객체 생성시 받은 filepath를 기반으로, 텍스트파일의 내용을 파싱하여 시작 symbol과 오른쪽 규칙을 기반으로 하는 production\_rule 포맷의 리스트를 구성한다. 이후, Augmented rule을 추가한다. |
| compute\_all\_possible\_dot\_list | - | - | Goto와 clousre를 구하는 과정에서, 빈번하게 참조될 수 있는 모든 non-terminal로 시작하는 문법의 첫번째 index에 dot notation이 있는 리스트를 구성한다. 첫번째 rule은 (S->E) 이므로, closure(S->.E)를 미리 구해서 추후에 사용을 용이하게 하기 위함이다. |
| get\_closure | Dot\_rule rule | List<Dot\_rule>  rule에서, dot notation이후에 진행할 수 있는 closure를 구하여, list의 형태로 리턴 | 일반적인 closure연산과 같다. 단, rule에 끝 부분을 알기 위하여, 현재 문법에서 사용될 수 없는 문자 ‘>’ 를 사용하여 문자열의 끝을 표현하였다. |
| get\_goto | char symbol, List<Dot\_rule> list | List<Dot\_rule>  현재 Iterator의 Dot\_rule 리스트에서, symbol을 기반으로 하는 goto를 리턴 | Goto 연산을 위하여, goto(I0, s) 등으로 표현되는 연산을, 현재 Iterator가 포함하는 규칙에서 진행될 수 있는 symbol을 기반으로, 현재 Iterator에서 특정 문자로 인하여 진행되고, 그 진행된 결과의 closure를 모두 연산하여 List의 형태로 리턴한다. |
| c0\_compute | - | - | 최초에, Augmented rule의 goto를 구하여, I0로 지정한다. 이후 I0를 리스트에 추가하고, 리스트를 순회하여 현재 Iterator에서 연산 가능한 모든 goto를 구한다. 이때 생성된 goto, 즉 Iterator가 이전에 계산되어 리스트에 존재하지 않는다면, 그 Iterator를 리스트에 추가한다.  더 이상 리스트에 Iterator를 추가할수 없으면 종료한다. |
| union | List<Dot\_rule>, List<Dot\_rule> | List<Dot\_rule> | 두개의 Dot\_rule 리스트에서 중복인 규칙을 제외하고 합치기 위한 함수이다. |
| get\_char\_next\_dot\_list | List<Dot\_rule> | List<Character> | 현재 Dot\_rule의 리스트에서, dot notation이후에 존재하는 모든 문자의 리스트를 리턴한다. Goto연산시에 사용된다. |
| is\_exist | Iterator\_c0 | 이미 생성되었다면, true | Iterator 연산 중, 결과로 생성된 Iterator가 이전에 생성 되었었는지를 판단하고, loop를 종료하기 위하여 사용된다. |