2022 봄학기

KNU 4471.043 컴파일러 설계 과제 10

고상기

2022년 5월 22일 23시 59분

- 이번 과제는 6주차에 배운 내용을 기반으로 Decaf 언어의 LL 구문 분석기를 구현하고 분석 결과를 그래프로 출력하는 것입니다.
- 4주차 과제에서 보여드린 Decaf 언어의 EBNF 문법 중 ⟨Expr⟩로부터 생성될 수 있는 생성규칙들 중 일부를 구현하는 것이 목표입니다. 과제해서 구현해야 할 문법은 다음과 같습니다.

⟨Expr⟩ ::= `Constant` | `id` | `this` | `(` ⟨Expr⟩ `)` | ⟨Expr⟩ `+` ⟨Expr⟩ |
⟨Expr⟩ `-` ⟨Expr⟩ | ⟨Expr⟩ `*` ⟨Expr⟩ | ⟨Expr⟩ `/` ⟨Expr⟩ |
⟨Expr⟩ `%` ⟨Expr⟩ | `-` ⟨Expr⟩ | ⟨Expr⟩ `<` ⟨Expr⟩ | ⟨Expr⟩ `<=` ⟨Expr⟩ |
⟨Expr⟩ `>` ⟨Expr⟩ | ⟨Expr⟩ `>=` ⟨Expr⟩ | ⟨Expr⟩ `==` ⟨Expr⟩ |
⟨Expr⟩ `!=` ⟨Expr⟩ | ⟨Expr⟩ `&&` ⟨Expr⟩ | ⟨Expr⟩ `||` ⟨Expr⟩ | `!` ⟨Expr⟩ |
`ReadInteger` `(` `)` | `ReadLine` `(` `)` | `new` `id` | `NewArray` `(` ⟨Expr⟩ `,` `Type` `)`

- 다음은 위 EBNF 문법에서 사용된 표기법과 LL(1) 문법으로 전환하기 위한 과정들에 대한 추가적인 설명입니다.
  - ⟨Expr⟩: 해당 생성규칙에서 논터미널 기호는 ⟨Expr⟩ 하나뿐입니다. 나머지는 전부 터미널 기호로 간주합니다.
  - `Constant`, `id`, `type`: 기존 decaf 문법에선 `Constant`, `id`, `type`은 상수와 문자열, 식별자, 변수 타입등을 생성하는 논터미널 기호였습니다. 그러나 이번 과제에선 터미널 기호의 일종으로 취급합니다. 단, 그래프 출력 시 해당 터미널 기호가 어떤 value를 가지고 있는지도 같이 출력해야 합니다.
  - 문법의 모호성을 없애기 위해선 우선 연산자 우선순위를 고려해야 합니다. 연산자 우선순위를 고려해 문법을 변환하는 방법은 5주차 강의자료의 9–14 페이지를 참고해주세요.
  - 연산자 우선순위는 다음과 같습니다.

우선순위	연산자	연산자 유형
1	( )	괄호
2	! -	단항 연산
3	* / %	산술 연산
4	+ -	
5	< <= > >=	관계 연산
6	== !=	
7	&&	논리 연산
8	||	

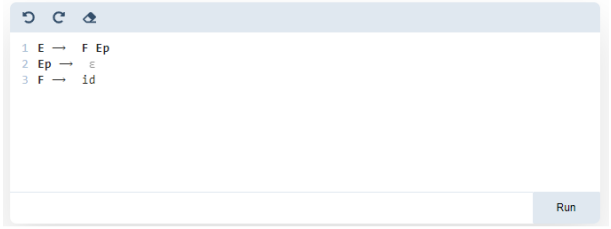
  - LL(1) 문법으로 변환하기 위해선 다음으로 좌재귀를 제거해야합니다. 해당 문법은 직접 좌재귀만 제거하면 추가적인 간접 좌재귀는 발생하지 않습니다.
  - 다음은 ⟨Expr⟩ → ⟨Expr⟩ `||` ⟨Expr⟩ 생성규칙에서 직접 좌재귀를 제거하는 예시입니다. ⟨Expr⟩은 편의상 E로 표기하겠습니다.

$$E \to FE'$$
$$E' \to \texttt{||}\ FE' \mid \epsilon$$

  - Hint : 다음은 우선순위를 고려해 문법을 변환하고 좌재귀를 제거한 문법의 형태입니다.

$$E \to FE'$$
$$E' \to \texttt{||}FE' \mid \epsilon$$
$$F \to GF'$$
$$\cdots$$
$$K \to \texttt{-}K \mid \texttt{!}K \mid (E) \mid \texttt{Constant} \mid \texttt{id} \mid \texttt{this} \mid \texttt{ReadInteger}() \mid \texttt{ReadLine}() \mid \texttt{new id} \mid \texttt{NewArray}(E, \texttt{Type})$$

- 생성규칙에 대한 FIRST와 FOLLOW 집합과 예측 파싱 테이블은 다음 URL에서 제공하는 도구로 간단히 구할 수 있습니다: `https://mikedevice.github.io/first-follow/`
- 해당 도구에서 생성 규칙의 왼쪽에 적힌 기호는 모두 논터미널 기호로 간주되고, 왼쪽에 적히지 않은 기호는 모두 터미널 기호로 간주됩니다. 또한 오른쪽 생성규칙에 아무것도 적지 않으면 ϵ으로 전이되는 규칙을 표현합니다. 각 기호간의 구분은 스페이스(space)로 합니다. FOLLOW 집합과 예측 파싱 테이블에서 문자열의 끝을 나타내는 기호는 ⊣입니다. 출력된 예측 파싱표에 따라 과제 코드를 작성하시면 됩니다.



First sets

	id	ε
E	+	-
Ep	-	+
F	+	-

Follow sets

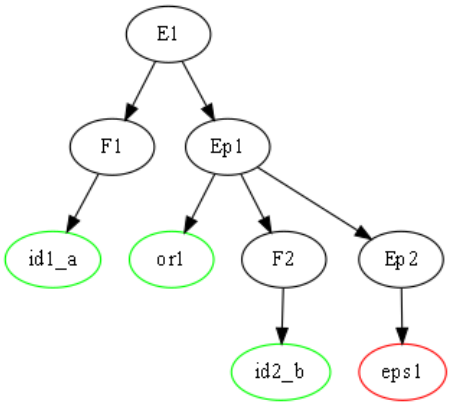
	id	⊣
E	-	+
Ep	-	+
F	-	+

Predict sets

	id	⊣
1	+	-
2	-	+
3	+	-

- 실습 파일로는 완성된 어휘 분석기와 완성되지 않은 구문 분석기의 스켈레톤 코드가 제공됩니다. 어휘 분석기는 프로그램 코드 파일을 입력으로 받아 토큰 리스트(`ArrayList` 객)를 반환합니다. 구문 분석기는 이 리스트를 입력으로 받아 파싱 결과를 출력합니다.
- 스켈레톤 코드 `LLParser.java`는 다음과 같은 간단한 생성규칙을 가진 언어를 파싱하고, $a \mid\mid b$ 라는 문자열에 대해 다음과 같은 그래프를 출력합니다.

$$E \rightarrow FE'$$
$$E' \rightarrow \mid\mid FE'$$
$$E' \rightarrow \epsilon$$
$$F \rightarrow \texttt{id}$$



- 정상적으로 파싱되는 문장에 대한 콘솔 출력 결과는 다음과 같습니다.

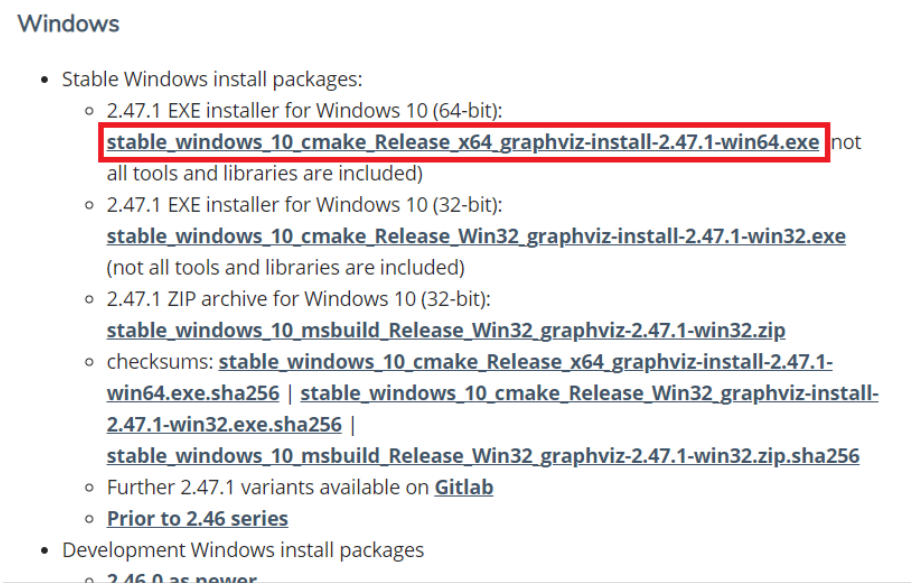
```
Accept
```

- 정상적으로 파싱되지 않는 문장에 대한 콘솔 출력 결과는 다음과 같습니다.

```
Invalid input. Expected $, but received ) at index 17.
```

- 프로젝트 폴더 내의 samples 폴더엔 입력 예시와 출력 예시가 포함되어 있으니 참고바랍니다.
- 다음은 GraphViz의 설치 방법부터 사용법에 대해 설명합니다. (Windows 10 64bit 기준)

1. GraphViz 다운로드: 본인 컴퓨터의 운영체제에 맞는 GraphViz 다운로드 및 설치(`https://graphviz.org/download/`).



실행파일 실행 시, 따로 체크하거나 수정하는 항목 없이 다음 버튼과 설치 버튼만 누르면 설치됩니다.

2. 프로젝트 파일 수정: `/src/Parser/config.properties` 파일 내용을 자신의 설치 경로에 맞게 수정하세요. 설치 시 설정을 바꾸지 않으셨다면 수정없이 그대로 사용할 수 있습니다.

```
############################################################
#                  Windows Configurations                  #
############################################################
# The dir. where temporary files will be created.
tempDirForWindows = C:/Temp
# Where is your dot program located? It will be called externally.
dotForWindows = C:/Program Files/Graphviz/bin/dot.exe
# 32bit 운영체제는 다음 경로에 설치됩니다. dotForWindows = C:/Program Files (x86)/Graphviz/bin/dot.exe
```

   - `tempDirForWindows` : 임시 파일 생성 폴더(임의로 변경 가능)
   - `dotForWindows` : 설치된 GraphViz의 실행파일인 `dot.exe` 파일의 경로를 입력

3. `addConnection` 메소드 사용법

```java
/**
 * 부모 노드의 번호, 부모노드 이름, 자식노드 이름을 입력해 그래프에 간선을 하나 추가하는 메소드
 * @param parentNum 부모노드의 번호
 * @param parentNode 부모노드의 이름
 * @param childNode 자식노드의 이름
 * @param childValue 자식노드가 id, Constant, Type인 경우 Value 입력, 그렇지 않을 땐 빈 문자열 "" 입력
 */
static void addConnection(int parentNum, String parentNode, String childNode, String childValue) {
    symbolMap.put(childNode, symbolMap.get(childNode) + 1);
    String valueNode = "";

    if (childValue.equals(""))
        valueNode = childNode + symbolMap.get(childNode);
    else
        valueNode = childNode + symbolMap.get(childNode) + "_" + childValue;

    dotFormat += parentNode + parentNum + "->" + valueNode + ";";
    valueList.add(valueNode);
}
```

- 그래프 객체를 선언하고 이미지 파일로 변환하는 코드, 그래프에 노드와 간선을 추가하는 코드는 모두 주어져 있습니다. 여러분이 그래프 생성을 위해 해야 할 일은 `addConnection` 메소드를 적절한 위치에서 적절한 방법으로 호출하는 것입니다.
- 부모 노드의 번호, 부모 노드의 이름, 자식 노드의 이름, 자식노드의 값을 인자로 넘겨주게 되면, 그래프에 부모노드와 자식노드를 잇는 간선이 추가됩니다. 부모 노드는 `parentNUm`을 따르고, 자식 노드는 추가될 때마다 `symbolMap`에 $+1$씩 더해지며 저장된 번호를 따릅니다.
- 자식노드에 값이 있는 경우는 `id`(변수이름), `Constant`(상수 및 문자열), `Type`(`int`, `double`과 같은 원시 자료형) 입니다. 따라서 이 세가지 경우 외에는 `addConnection` 호출시 빈 문자열 “” 을 넣어주면 되고, 세 가지 경우에 해당한다면

  $$\texttt{symbolist.get(index).getValue}$$

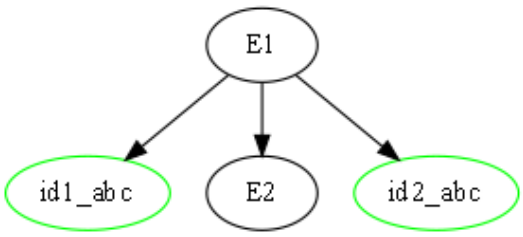
  를 호출해 해당 토큰의 value를 넘겨주면 됩니다.
- 이해를 돕기 위해 다음과 같은 생성 규칙이 있다고 가정합시다.

  $$E \rightarrow \texttt{id}\ \ E\ \ \texttt{id}$$

- 이때, 위의 생성규칙에 해당하는 그래프를 생성하는 명령은 다음과 같습니다.

```
addConnection(1, E, id, symbolist.get(index).getValue);
addConnection(1, E, E, “”);
addConnection(1, E, id, symbolist.get(index).getValue);
```

- 출력된 그래프 이미지는 다음과 같습니다.



  - 노드에 추가된 번호들은 오직 같은 기호 간의 구분을 위한 것일 뿐, 순서와는 상관이 없습니다. 따라서 `id2`가 왼쪽에, `id1`이 오른쪽에 와도 괜찮습니다. 그리고 id 노드에는 변수명인 abc가 뒤에 추가된 것을 볼 수 있습니다. 앞서 `id`는 터미널 기호로 취급하지만 value를 가진다고 설명했습니다. 이처럼 `Constant`, `id`, `Type`에 대해서는 그래프 출력시 value도 함께 출력해야 합니다.
    - `Constant`의 value는 정수 상수 또는 영문 알파벳 문자열 상수입니다.
    - `id`의 value는 변수의 식별자 이름입니다.
    - `Type`의 value는 `int`, `double`, `string`, `bool`입니다.
  - `samples` 폴더에 있는 예시 그래프들은 논터미널 기호의 이름들이 지워진 상태입니다. 만약 주어진 문법을 LL(1) 문법으로 변환한 후에 코드를 완성하고 실행시킨다면, 예시 이미지와 같거나 비슷한 그래프가 논터미널 기호도 제대로 표시되며 출력될 것입니다.
  - 예시 이미지들의 `eps`는 ϵ 기호를 뜻합니다.

4. GraphViz 활용시 주의사항
  - GraphViz의 노드 이름에는 특수문자를 사용할 수 없습니다. 영문 알파벳 문자와 숫자만으로 이루어진 노드 이름을 생성해주세요.
  - 논터미널 기호 노드는 앞 글자가 알파벳 대문자로 시작하도록 작성해주세요.
  - 터미널 기호 노드는 다음과 같은 알파벳 약자로 대체합니다. 다른 약자 말고 정해진 약자를 사용해주세요.

```
|| or / && and / == eq / != neq / < ll / <= le / > gg / >= ge /
+ sum / - sub / * mul / '/' div / % mod / ! not / ( lP / ) rP /
Constant con / this ths / ReadInteger rdI / ReadLine rdL / new new /
id id / newArray nwA / , com / Type typ / epsilon eps
```

  - 각 노드의 번호들은 예시 이미지와 같지 않아도 괜찮습니다.
  - 앞서 설명한 대로 `Constant`, `id`, `Type`의 value에는 특수문자가 포함되지 않습니다.

- 과제를 다 마친 후에는 `LLParser.java` 파일만 제출해주세요. 주요 채점 기준은 다음과 같습니다.
  - 문법적으로 옳은 코드에 대해 구문 분석 결과 출력 여부
  - 문법적으로 틀린 코드에 대해 오류 출력 여부
  - GraphViz를 활용하여 입력 코드에 대한 추상 구문 트리(abstract syntax tree) 출력
  - 출력된 추상 구문 트리에서의 연산자 우선순위 고려 여부
  - `Constant`, `id`, `Type` 노드에 대한 값(value) 제대로 출력