Homework 2
COSE312, Spring 2020

Hakjoo Oh

Due: 04/24, 23:59

Problem 1 이번 숙제에서는 수업시간에 다룬 하향식 파서(top-down parser)를 구현해 봅시다. OCaml 자료형으로 문맥 자유 문법(context-free grammar)을 다음과 같이 정의할 수 있습니다.

```
type symbol =
  | T of string  (* terminal symbol *)
  | N of string  (* nonterminal symbol *)
  | Epsilon      (* empty string *)
  | End          (* end marker *)
type production = (symbol * symbol list) list
type cfg = symbol list * symbol list * symbol * production
```

예를 들어, 다음의 문법

$$(\{E, E', T, T', F\}, \{+, *, (,), \mathbf{id}\}, E, \left\{ \begin{array}{lcl} E & \rightarrow & T\ E' \\ E' & \rightarrow & +\ T\ E' \mid \epsilon \\ T & \rightarrow & F\ T' \\ T' & \rightarrow & *\ F\ T' \mid \epsilon \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array} \right\})$$

은 아래와 같이 나타낼 수 있습니다:

```
let cfg1 = (
  [N "E"; N "E'"; N "T"; N "T'"; N "F"],
  [T "+"; T "*"; T "("; T ")"; T "id"],
  N "E",
  [
    (N "E", [N "T"; N "E'"]);
    (N "E'", [T "+"; N "T"; N "E'"]);
    (N "E'", []);
    (N "T", [N "F"; N "T'"]);
    (N "T'", [T "*"; N "F"; N "T'"]);
```

```
    (N "T'", []);
    (N "F", [T "("; N "E"; T ")"]);
    (N "F", [T "id"])
  ])
```

문자열은 기호(symbol)의 리스트로 나타냅니다. 예를 들어, 문자열 $\mathbf{id} + \mathbf{id} * \mathbf{id}$는 다음과 같이 표현합니다(문자열의 가장 마지막 기호는 `End`이어야 합니다):

```
let s1 = [T "id"; T "+"; T "id"; T "*"; T "id"; End]
```

1. 주어진 문법이 LL(1)에 속하는지 확인하는 함수 `check_LL1`을 작성해 봅시다.

   ```
   check_LL1 : cfg -> bool
   ```

   예를 들어, `check_LL1 cfg1`은 `true`를 반환해야 합니다. 아래와 같은 문법들에 대해서

   ```
   let cfg2 = (
     [N "S"; N "E"; N "E'"; N "T"; N "T'"; N "F"],
     [T "+"; T "-"; T "*"; T "/"; T "id"; T "num"; T "("; T ")"],
     N "S",
     [
         (N "S", [N "E"]);
         (N "E", [N "T"; N "E'"]);
         (N "E'", [T "+"; N "T"; N "E'"]);
         (N "E'", [T "-"; N "T"; N "E'"]);
         (N "E'", []);
         (N "T", [N "F"; N "T'"]);
         (N "T'", [T "*"; N "F"; N "T'"]);
         (N "T'", [T "/"; N "F"; N "T'"]);
         (N "T'", []);
         (N "F", [T "id"]);
         (N "F", [T "num"]);
         (N "F", [T "("; N "E" ;T ")"]);
     ]
   )

   let cfg3 = (
     [N "X"; N "Y"; N "Z"],
     [T "a"; T"c"; T"d"],
     N "X",
     [
   ```

```
    (N "X", [N "Y"]);
    (N "X", [T "a"]);
    (N "Y", [T "c"]);
    (N "Y", []);
    (N "Z", [T "d"]);
    (N "Z", [N "X"; N "Y"; N "Z"])
  ]
)

let cfg4 = (
  [N "S"; N "S'"; N "E"],
  [T "a"; T "b"; T "e"; T "i"; T "t"],
  N "S",
  [
   (N "S", [T "i"; N "E"; T "t"; N "S"; N "S'"]);
   (N "S", [T "a"]);
   (N "S'", [T "e"; N "S"]);
   (N "S'", []);
   (N "E", [T "b"])
  ]
)
```

`check_LL1 cfg2`는 `true`를, `check_LL1 cfg3`는 `false`를, `check_LL1 cfg2`는 `false`를 반환해야 합니다.

2. 문법과 문자열을 받아서 하향식 파싱을 수행하는 함수 `parse`를 작성해 봅시다.

```
parse : cfg -> symbol list -> bool
```

`parse`의 첫번째 입력으로 주어진 문법이 LL(1)이라 가정할 때, 두번째 입력으로 주어진 문자열이 해당 문법이 규정하는 언어에 속하면 `true`, 아니면 `false`를 반환합니다. 예를 들어, 아래 문자열들에 대해서

```
let s1 = [T "id"; T "+"; T "id"; T "*"; T "id"; End]
let s2 = [T "id"; T "/"; T "("; T "num"; T "+"; T "id"; T ")"; End]
```

`parse cfg1 s1`, `parse cfg1 s2`, `parse cfg2 s1`, `parse cfg2 s2`는 각각 `true`, `false`, `true`, `true`를 반환해야 합니다.