CS 421 --- LL Parsing Activity

Manager	Keeps team on track	
Recorder	Records decisions	
Reporter	Reports to class	
Reflector	Assesses team performance	

Purpose

There is a certain class of grammar for which it is very easy to write a parser without needing any special libraries or tools. Your objectives are to demonstrate how to write such a parser, how to identify a grammar that can use this approach, and how to fix common problems that prevent a grammar from being implemented as a recursive descent parser.

Problem 1 --- Recursive Descent Parsers

Consider these two grammars. Lower case letters will represent literal characters in the input. Upper case letters will represent nonterminal symbols. The character `c` will represent a random character.

Grammar 1

$$\begin{aligned} S &\rightarrow x\ S\ y \\ &\mid E \\ E &\rightarrow a\ E\ b \\ &\mid c \end{aligned}$$

Grammar 2

$$\begin{aligned} S &\rightarrow a\ S \\ &\mid E \\ &\mid F \\ E &\rightarrow b\ c \\ F &\rightarrow d\ c \end{aligned}$$

Consider now these HASKELL programs. Assume niceties like `deriving Show`, etc.

```haskell
-- Program 1
data S = S1 Char S Char
       | S2 E
data E = E1 Char E Char
       | E2 Char
parseS ('x':xs) =
   let (s,r1) = parseS xs
       ('y':r2) = r1
    in (S1 'x' s 'y', r2)
parseS xx = parseE xx
parseE ('a':xs) =
   let (e,r1) = parseE xs
       ('b':r2) = r1
    in (E1 'a' e 'b', r2)
parseE (x:xs) = (E2 x, xs)
```

```haskell
-- Program 2
data S = S1 Char S
       | S2 E
       | S3 F
data E = E1 Char Char
data F = F1 Char Char
parseS ('a':xs) =
   let (s,r1) = parseS xs
    in (S1 'a' s, r1)
parseS ('b':xs) = parseE ('b':xs)
parseS ('d':xs) = parseF ('d':xs)
parseE ('b':x:xs) = (E1 'b' x, xs)
parseF ('d':x:xs) = (F1 'd' x, xs)
```

Answer the questions on the next page.

Matching Grammar to Code Each of these programs is meant to implement the corresponding grammar. With your team, review the code and be able to explain to each other how it works. Then answer the following questions:
Problem 1) Why do each of the parse functions return a tuple?

Problem 2) How is parsing a non-terminal different than parsing a terminal?

Problem 3) The second grammar has these two line:

```
1 parseE ('b':x:xs) = (E1 'b' x, xs)
2 parseF ('d':x:xs) = (F1 'd' x, xs)
```

From this example, can you explain how the `parseS` function determines which of `parseE` or `parseF` to call?

Problem 2 --- What Could Possibly Go Wrong?

Grammar 3

$$\begin{aligned} S \rightarrow & \quad S\,y \\ \mid & \quad x \end{aligned}$$

Grammar 4

$$\begin{aligned} S \rightarrow & \quad a\,S \\ \mid & \quad a\,E \\ E \rightarrow & \quad x \end{aligned}$$

Consider now these HASKELL implementations. We omit the data declarations this time.

```haskell
-- Program 3
parseS ('x':xs) = (S2 'x', xs)
parseS xx =
   let (s,r1) = parseS xx
       ('y':r2) = r1
    in (S1 s 'y', r2)
```

```haskell
-- Program 4
parseS ('a':xs) =
   let (s,r1) = parseS xs
    in (S1 'a' s, r1)
parseS ('a':xs) =
   let (e,r1) = parseE xs
    in (S1 'a' e, r1)
parseE ('x':xs) = (E1 'x', xs)
```

Problem 4) The first program has a problem. What goes wrong? What feature of the grammar causes this problem to occur?

Problem 5) The second program also has a problem. What goes wrong? What feature of the grammar causes this problem to occur?

Problem 3 --- Fixing Left Recursion

Problem 6) Consider these two grammars:

Grammar 5

$$\begin{array}{rcl} S & \rightarrow & S\,a \\ & | & b \end{array}$$

Grammar 6

$$\begin{array}{rcl} S & \rightarrow & b\,S' \\ S' & \rightarrow & a\,S' \\ & | & \epsilon \end{array}$$

Draw two parse trees for the string baaa, one for each of the above grammars.

Problem 7) Consider these two grammars:

Grammar 7

$$\begin{array}{rcl} S & \rightarrow & S\,a \\ & | & S\,b \\ & | & c \\ & | & d\,e \end{array}$$

Grammar 8

$$\begin{array}{rcl} S & \rightarrow & c\,S' \\ & | & de\,S' \\ S' & \rightarrow & a\,S' \\ & | & b\,S' \\ & | & \epsilon \end{array}$$

Draw two parse trees for the string deba, one for each of the above grammars.

Problem 8) Describe a conversion procedure to fix a left-recursive grammar.

Given a grammar:

$$\begin{array}{rcl} S & \rightarrow & S\,\alpha \\ & | & \beta \end{array}$$

Show what the corresponding converted grammar looks like. The α and β here mean ``any arbitrary sequence of terminals and nonterminals.''

Problem 4 --- Fixing Common Prefixes

Problem 9) Consider these two grammars:

Grammar 9

$$\begin{aligned} S &\to a\ b \\ &\mid a\ E \\ E &\to x\ y \end{aligned}$$

Grammar 10

$$\begin{aligned} S &\to a\ S' \\ S' &\to b \\ &\mid E \\ E &\to x\ y \end{aligned}$$

Draw two parse trees for the string axy, one for each of the above grammars.

Problem 10) Consider these two grammars:

Grammar 11

$$\begin{aligned} S &\to a\ b \\ &\mid E \\ E &\to x\ y \\ &\mid a\ z \end{aligned}$$

Grammar 12

$$\begin{aligned} S &\to a\ S' \\ &\mid x\ y \\ S' &\to b \\ &\mid z \end{aligned}$$

Draw two parse trees for the string az, one for each of the above grammars. The second grammar is missing the E production entirely. Why is this necessary?

Problem 11) Describe a conversion procedure to fix a common-prefix rule in a grammar. Given this stylized grammar,

$$\begin{aligned} S &\to \alpha\ \beta \\ &\mid \alpha\ \gamma \\ &\mid \delta \end{aligned}$$

show what the corresponding converted grammar looks like.

Problem 5 --- Apply It!

Problem 12) This grammar is not LL. Convert it to an equivalent grammar that is LL.

Grammar 13

$$
\begin{array}{rcl}
S & \rightarrow & S\ x \\
 & | & a\ E \\
E & \rightarrow & y\ a\ y \\
 & | & y\ a\ z
\end{array}
$$

Problem 13) There is a third thing that can go wrong! Look at this grammar and describe what goes wrong. Note, it's not *just* that there is an ϵ production.

Grammar 14

$$
\begin{array}{rcl}
A & \rightarrow & B\ c \\
 & | & x \\
B & \rightarrow & c \\
 & | & \epsilon
\end{array}
$$

Problem 14) Were all these too easy? Try converting this one then.

Grammar 15

$$
\begin{array}{rcl}
A & \rightarrow & A\ x \mid B\ y \mid z \\
B & \rightarrow & A\ i \mid B\ j \mid k
\end{array}
$$

LL Parsing Activity --- Team's Assessment

Manager or Reflector: Consider the objectives of this activity and your team's experience with it, and then answer the following questions after consulting with your team.

1. What was a strength of this activity? List one aspect that helped it achieve its purpose.

2. What change could we make to this activity to make it more effective?

3. What insights did you have about the activity at the meta level? (I.e., we're not asking abou the content, but maybe how the activity was organized)

LL Parsing Activity--- Reflector's Report

Manager	Keeps team on track	
Recorder	Records decisions	
Reporter	Reports to Class	
Reflector	Assesses team performance	

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?