CD Moodle Assignment 2 📝

Jovial Joe Jayarson

11 June 2020

IES17CS016

1. Differentiate between top-down and bottom-up parsers. Discuss the term left recursion. Eliminate the left recursion in the following grammar.

$$\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{T}$$

$$\mathrm{T} \rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$$

$$\mathrm{F} \rightarrow (\mathrm{E}) \mid \mathrm{id}$$

Ans 1:

Top Down vs Bottom Up Approach

Top Down	Bottom Up
Parsing is done from the very top of the tree to leaf nodes.	Parsing is done in the reverse order or from bottom in upward direction
Uses left-most derivation	Uses right-most derivation in the reverse order
Contains Backtracking and Non-Backtracking Parsers	Aka. Shift-Reduce Parser; Contains Operator Precedence & LR Parsers

Left Recursion: A production of the form $A \rightarrow A\alpha|\beta$ is said called a left recursion because the non-terminal on the LHS is same as the first (left-most) in the RHS of a production. I must be eliminated by re-writing it as:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A'|\epsilon$$

Given Grammar:

$$\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{T}$$

$$\mathrm{T} \rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$$

$$\mathrm{F} \rightarrow (\mathrm{E}) \mid \mathrm{id}$$

It contains left recursion in the 1st and 2nd production.

- Removing left recursion in $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{T}$ here: α = $+ \mathrm{E}$ and β = T It can be re-written as:

$$\mathrm{E} \rightarrow \mathrm{T}\ \mathrm{E'}$$

$$\mathrm{E'} \rightarrow + \mathrm{T}\ \mathrm{E'} \mid \epsilon$$

- Removing left recursion in $\mathrm{T} \rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$ here α = $* \mathrm{F}$ and β = F It can be re-written as:

$$\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$$

$$\mathrm{T'} \rightarrow * \mathrm{F}\ \mathrm{T'} \mid \epsilon$$

∴ The final set of production rules or grammar is:

$$\mathrm{E} \rightarrow \mathrm{T}\ \mathrm{E'}$$

$$\mathrm{E'} \rightarrow + \ \mathrm{T}\ \mathrm{E'} \mid \epsilon$$

$$\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$$

$$\mathrm{T'} \rightarrow * \ \mathrm{F}\ \mathrm{T'} \mid \epsilon$$

$$\mathrm{F} \rightarrow (\ \mathrm{E}\ ) \mid \mathrm{id}$$

2. Find the `FIRST` and `FOLLOW` of the non-terminals in the grammar

$$\mathrm{E} \rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T}$$

$$\mathrm{T} \rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$$

$$\mathrm{F} \rightarrow (\mathrm{E}) \mid \mathrm{id}$$

Ans 2:

The given grammar contains left-recursion and so it must be removed first. Removing left-recursion from $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T}$

$$\mathrm{E} \rightarrow \mathrm{T}\ \mathrm{E'}$$

$$\mathrm{E'} \rightarrow + \mathrm{T}\ \mathrm{E'} \mid \epsilon$$

and removing left-recursion from $\mathrm{T} \rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$

$$\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$$

$$\mathrm{T} \rightarrow * \mathrm{F}\ \mathrm{T'} \mid \epsilon$$

Therefore the production rules are:

$$\mathrm{E} \rightarrow \mathrm{T}\ \mathrm{E'}$$

$$\mathrm{E'} \rightarrow + \mathrm{T}\ \mathrm{E'} \mid \epsilon$$

$$\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$$

$$\mathrm{T'} \rightarrow * \mathrm{F}\ \mathrm{T'} \mid \epsilon$$

$$\mathrm{F} \rightarrow (\mathrm{E}) \mid \mathrm{id}$$

Finding `FIRST(E, E', T, T', F)`:

> Rules
>
> - if α is a terminal a, then FIRST(α) = a.
> - if α is a non-terminal X and $\mathrm{X} \rightarrow$ a α, then FIRST(α) = $\{a\}$.
> - if α is a non-terminal X and $\mathrm{X} \rightarrow \epsilon$, then FIRST($\alpha$) = $\{\epsilon\}$.
> - If $\mathrm{X} \rightarrow Y_1, Y_2, ..., Y_k$:
>   - then FIRST(X) = FIRST(Y_1).
>   - If ϵ belongs to FIRST(Y_1):
>   - then FIRST(X) = FIRST(Y_2) and so on.
>   - If ϵ belongs to FIRST(Y_k) then ϵ is added to FIRST(X).

FIRST(E) = FIRST(T) = FIRST(F) = $\{\mathrm{c}, \mathrm{id}\}$ FIRST($\mathrm{E'}$) = $\{+, \epsilon\}$ FIRST(T) = FIRST(F) = $\{(, \mathrm{id}\}$ FIRST($\mathrm{T'}$) = $\{*, \epsilon\}$ FIRST(F) = $\{(, \mathrm{id}\}$ $\therefore$

Non-Terminals	`FIRST`
E	$\{c, id\}$
E'	$\{+, \epsilon\}$
T	$\{(, id\}$
T'	$\{*, \epsilon\}$
F	$\{(, id\}$

Finding `FOLLOW(E, E', T, T', F)`:

> Rules
>
> - If A is a start symbol the FOLLOW(A) = $.
> - If $A \rightarrow \alpha\ B\ \beta$ and $\beta \neq \epsilon$ then FOLLOW(B) = FIRST(β).
> - If $A \rightarrow \alpha\ B$ (i.e. $\beta = \epsilon$) or if $A \rightarrow \alpha\ B\ \beta$ and FIRST(β) contains ϵ then FOLLOW(B) = FOLLOW(A).

- E
  - FOLLOW(E) = $\{\$\}$ (being the first symbol)
  - FOLLOW(E) = FIRST( )) (production: $F \rightarrow (\ E\ ) \mid id$ here β =))

∴ FOLLOW(E) = $\{\$,)\}$

- E'
  - FOLLOW(E') = FOLLOW(E) = $\{\$,)\}$ (production: $E \rightarrow T\ E'$ here $\beta = \epsilon$)
  - FOLLOW(E') = FOLLOW(E') = $\{\$,)\}$ (production: $E \rightarrow +\ T\ E'$ here $\beta = \epsilon$)

∴ FOLLOW(E') = $\{\$,)\}$

- T
  - FOLLOW(T) = FIRST(E') = $\{+, \epsilon\}$ (production: $E \rightarrow T\ E'$ here $\beta = E'$) but FOLLOW cannot contain ϵ, therefore it (ϵ) is removed and FOLLOW(E) = $\{\$,)\}$ is added.
  - FOLLOW(T) = FIRST(E') = $\{+, \epsilon\}$ (production: $E' \rightarrow +\ E' \mid \epsilon$ here $\beta = E'$) but FOLLOW cannot contain ϵ, therefore it (ϵ) is removed and FOLLOW(E') = $\{\$,)\}$ is added.

∴ FOLLOW(T) = $\{+, \$,)\}$

- T'
  - FOLLOW(T') = FOLLOW(T) = $\{+, \$,)\}$ (production: $T \rightarrow F\ T'$ here $\beta = \epsilon$)
  - FOLLOW(T') = FOLLOW(T) = $\{+, \$,)\}$ (production: $T' \rightarrow *\ F\ T' \mid \epsilon$ here $\beta = \epsilon$)

∴ FOLLOW(T') = $\{+, \$,)\}$

- F

- FOLLOW(F) = FIRST($\mathrm{T'}$) = $\{*, \epsilon\}$ (production: $\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$ here β = $\mathrm{T'}$) but FOLLOW cannot contain ϵ, therefore it (ϵ) is removed and FOLLOW(T) = $\{+, \$,)\}$
- FOLLOW(F) = FIRST($\mathrm{T'}$) = $\{*, \epsilon\}$ (production: $\mathrm{T} \rightarrow * \ \mathrm{F}\ \mathrm{T'} \mid \epsilon$ here β = $\mathrm{T'}$) but FOLLOW cannot contain ϵ, therefore it (ϵ) is removed and FOLLOW(T) = $\{+, \$,)\}$

$\therefore$ FOLLOW(F) = $\{+, *, \$,)\}$

$\therefore$

Non-Terminals	`FIRST`	`FOLLOW`
E	$\{\mathrm{c}, \mathrm{id}\}$	$\{\$,)\}$
E'	$\{+, \epsilon\}$	$\{\$,)\}$
T	$\{(, \mathrm{id}\}$	$\{+, \$,)\}$
T'	$\{*, \epsilon\}$	$\{+, \$,)\}$
F	$\{(, \mathrm{id}\}$	$\{+, *, \$,)\}$

3. Explain about $\mathrm{LL}(1)$ grammar with an example?

Ans 3:

- LL(1) grammar:
  - An LL(1) grammar is a formal grammar that can be parsed by an LL(1) parser, which parses the input from Left to Right.
  - It constructs a Leftmost derivation of the sentence / production.
  - A language that has an LL grammar is known as an LL language.
  - 1 stands for using one input symbol at each step, i.e. position of `lookahead`.
- Identifying an LL(1) grammar:
  - Construct the LL(1) parsing table and check for any conflicts. These conflicts can be FIRST/FIRST conflicts, where *two different productions* would have to be predicted for a non-terminal/terminal pair.

Consider the grammar:

$$\mathrm{E} \rightarrow \mathrm{T}\ \mathrm{E'}$$

$$\mathrm{E'} \rightarrow +\ \mathrm{T}\ \mathrm{E'} \mid \epsilon$$

$$\mathrm{T} \rightarrow \mathrm{F}\ \mathrm{T'}$$

$$\mathrm{T'} \rightarrow *\ \mathrm{F}\ \mathrm{T'} \mid \epsilon$$

$$\mathrm{F} \rightarrow (\ \mathrm{E}\ ) \mid \mathrm{id}$$

The FIRST and FOLLOW will be:

Non-Terminals	`FIRST`	`FOLLOW`
E	$\{\mathrm{c}, \mathrm{id}\}$	$\{\$,)\}$
E'	$\{+, \epsilon\}$	$\{\$,)\}$
T	$\{(, \mathrm{id}\}$	$\{+, \$,)\}$
T'	$\{*, \epsilon\}$	$\{+, \$,)\}$
F	$\{(, \mathrm{id}\}$	$\{+, *, \$,)\}$

Construct the parse table:

> Rules
>
> - Create table with non-terminals are row headers and terminals $\cup \{\$\}$ as column headers.
> - For each non-terminal (NT), terminal pair(T); the table entry will be production NT → T if it (T) is found in FIRST(NT).

> - If ϵ is found in FIRST(NT) then fill the table corresponding to T found in FOLLOW(NT) with NT $\rightarrow \epsilon$
> - All the free cells are left as is and are called `ErrorState`
> - Then according to the statement if there are more than one productions / entries within a cell then the grammar is not LL(1).

	id	$+$	$*$	$($	$)$	$\$$
E	$\text{E} \rightarrow \text{T E'}$			$\text{E} \rightarrow \text{T E'}$		
E'		$\text{E'} \rightarrow + \text{T E'}$			$\text{E'} \rightarrow \epsilon$'	$\text{E'} \rightarrow \epsilon$'
T	$\text{T} \rightarrow \text{F T'}$			$\text{T} \rightarrow \text{F T'}$		
T'		$\text{T'} \rightarrow \epsilon$	$\text{T'} \rightarrow * \text{F T'}$		$\text{T'} \rightarrow \epsilon$	$\text{T'} \rightarrow \epsilon$
F	$\text{F} \rightarrow \text{id}$			$\text{F} \rightarrow (\text{E})$		

Now there are no double entries in the table.

$\therefore$ The given grammar is $\text{LL}(1)$.

4. Construct a recursive descent parser of the following grammar.

$$S \rightarrow cAd$$

$$A \rightarrow ab \mid b$$

Ans 4:

- A parser that uses a collection of recursive procedure for parsing the given input string is called a *recursive descent parser*.
- It is a kind of top-down parser built from a set of mutually recursive procedures.

> Rules
>
> - Remove left recursion if any.
> - For every non terminal write a function.
> - `l` is a lookahead pointer.
> - `match()` is a function which checks for equality of input with `l`. It also obtains the next character.

Given:

$$S \rightarrow cAd$$

$$A \rightarrow ab \mid b$$

This grammar has no left recursion, hence proceeding with left recursion.

```
void S()
{
    if(l == 'c')
    {
        match('c');
        A();
        match('d');
    }
}
void A()
{
    if(l == 'a')
    {
        match('a');
        match('b');
    }
    else
        match('a');
```

```
}
void match(char t)
{
    if(l == t)
    {
        l = getchar();
    }
    else
        printf("MatchError");
}
void main()
{
    S();
    if(l == '$')
        printf("Sucess")
    else
        printf("Failed")
}
```