Recursive descent top-down parsing

LL(1) parser can be written simply as a set of recursive functions:

```
void E() { M(); E1(); }

void E1() {
  if (next == '+')
    { skip('+'); E(); }
  else if (next == ')')
    { }
  else
    { syntaxerror(); }
}
```

$$E \to M\ E'$$
$$E' \to +\ E$$
$$E' \to$$
$$M \to F\ M'$$
$$M' \to *\ M$$
$$M' \to$$
$$F \to x$$
$$F \to y$$
$$F \to (\ E\ )$$

Recursive descent top-down parsing

LL(1) parser can be written simply as a set of recursive functions:

```
void E() { M(); E1(); }

void E1() {
  if (next == '+')
    { skip('+'); E(); }
  else if (next == ')')
    { }
  else
    { syntaxerror(); }
}
```

$$
\begin{aligned}
E &\to M\ E' \\
E' &\to +\ E \\
E' &\to \\
M &\to F\ M' \\
M' &\to *\ M \\
M' &\to \\
F &\to x \\
F &\to y \\
F &\to (\ E\ )
\end{aligned}
$$

Where do these symbols come from?

What if they overlap?

How does it work?

- Use current terminal symbol(s) to choose production
- Then skip matching terminal symbols and recursively parse nonterminal symbols in production
- How to use terminal symbol to decide which production to use?

Top-down parsing

How to construct a parser for any LL(1) grammar.

Top-down parsing: how?

- Use current terminal symbol(s) to choose production
- Then skip matching terminal symbols and parse nonterminal symbols in production
- How to map from terminal symbol to production?

  General method:
  Compute, for each nonterminal symbol *N*:

  - FIRST(*N*): set of terminal symbols that can begin strings derived from *N*.
  - FOLLOW(*N*): set of terminal symbols that can immediately follow strings derived from *N*.
  - nullable(*N*): whether empty string can be derived from *N*.

To compute FIRST and FOLLOW (simple)

Initialize: $\forall N$ FIRST(N)=$\emptyset$, FOLLOW(N)=$\emptyset$

For each terminal symbol T, FIRST(T)=$\{T\}$

Repeat (until FIRST and FOLLOW do not change):

- For each production $X \rightarrow Y_1 \ldots Y_k$:
  - Add FIRST(Y_1) to FIRST(X);
  - Add FOLLOW(X) to FOLLOW(Y_k);
  - For each i from 1 to k-1:
    - Add FIRST(Y_{i+1}) to FOLLOW(Y_i);

To compute FIRST, FOLLOW, nullable

Initialize: $\forall N$ FIRST(N)=$\emptyset$, FOLLOW(N)=$\emptyset$, nullable(N)=false

For each terminal symbol T, FIRST(T)=$\{T\}$

Repeat (until FIRST, FOLLOW, nullable do not change):

- For each production $X \to Y_1 \ldots Y_k$:
  - If k=0 or $Y_1 \ldots Y_k$ are all nullable then nullable(X) = true;
  - For each i from 1 to k:
    - If i=1 or $Y_1 \ldots Y_{i\text{-}1}$ are all nullable
      - then add FIRST(Y_i) to FIRST(X);
    - If i=k or $Y_{i+1} \ldots Y_k$ are all nullable
      - then add FOLLOW(X) to FOLLOW(Y_i);
    - For each j from i+1 to k:
      - If i+1=j or $Y_{i+1} \ldots Y_{j\text{-}1}$ are all nullable
        - then add FIRST(Y_j) to FOLLOW(Y_i);

Example

0	**nullable**	**FIRST**	**FOLLOW**
E	false	$\emptyset$	$\emptyset$
E'	false	$\emptyset$	$\emptyset$
M	false	$\emptyset$	$\emptyset$
M'	false	$\emptyset$	$\emptyset$
F	false	$\emptyset$	$\emptyset$

$$
\begin{aligned}
&E \rightarrow M\ E' \\
&E' \rightarrow +\ E \\
&E' \rightarrow \\
&M \rightarrow F\ M' \\
&M' \rightarrow *\ M \\
&M' \rightarrow \\
&F \rightarrow x \\
&F \rightarrow y \\
&F \rightarrow (\ E\ )
\end{aligned}
$$

Example

1	**nullable**	**FIRST**	**FOLLOW**
E	false	$\emptyset$	$\{)\}$
E'	true	$\{+\}$	$\emptyset$
M	false	$\emptyset$	$\emptyset$
M'	true	$\{*\}$	$\emptyset$
F	false	$\{x,y,(\}$	$\emptyset$

$$E \to M\ E'$$
$$E' \to +\ E$$
$$E' \to$$
$$M \to F\ M'$$
$$M' \to *\ M$$
$$M' \to$$
$$F \to x$$
$$F \to y$$
$$F \to (\ E\ )$$

Example

2	**nullable**	**FIRST**	**FOLLOW**
E	false	$\{x,y,(\}$	$\{)\}$
E'	true	$\{+\}$	$\{)\}$
M	false	$\{x,y,(\}$	$\{+,)\}$
M'	true	$\{*\}$	$\emptyset$
F	false	$\{x,y,(\}$	$\{*\}$

$$E \to M\ E'$$
$$E' \to +\ E$$
$$E' \to$$
$$M \to F\ M'$$
$$M' \to *\ M$$
$$M' \to$$
$$F \to x$$
$$F \to y$$
$$F \to (\ E\ )$$

Example

3	**nullable**	**FIRST**	**FOLLOW**
E	false	$\{x,y,(\}$	$\{)\}$
E'	true	$\{+\}$	$\{)\}$
M	false	$\{x,y,(\}$	$\{+,)\}$
M'	true	$\{*\}$	$\{+,)\}$
F	false	$\{x,y,(\}$	$\{*,+,)\}$

$$E \rightarrow M\ E'$$
$$E' \rightarrow +\ E$$
$$E' \rightarrow$$
$$M \rightarrow F\ M'$$
$$M' \rightarrow *\ M$$
$$M' \rightarrow$$
$$F \rightarrow x$$
$$F \rightarrow y$$
$$F \rightarrow (\ E\ )$$

Predictive parsing table (LL(1) table)

Row for each nonterminal, column for each terminal symbol.

Put production $X \rightarrow Y_1 \ldots Y_k$ in row X, column a if:

- $a \in$ FIRST($Y_1 \ldots Y_k$)
- $a \in$ FOLLOW(X) if $Y_1 \ldots Y_k$ are all nullable

FIRST($Y_1 \ldots Y_k$) = FIRST(Y_1) $\cup$ FIRST($Y_2 \ldots Y_k$) if Y_1 nullable

FIRST($Y_1 \ldots Y_k$) = FIRST(Y_1) otherwise

Predictive parsing table (LL(1) table)

Or, for production $X \rightarrow Y_1 \ldots Y_k$:

set = ∅;

for each *i* from 1 to *k* {

add FIRST(Y_i) to set;

if (Y_i is not nullable) return *set*;

}

add FOLLOW(X) to set;

return *set*;

Put production $X \rightarrow Y_1 \ldots Y_k$ in row X, column a for all a in *set*.

Example predictive parsing table

$$E \to M\ E'$$
$$E' \to +\ E$$
$$E' \to$$
$$M \to F\ M'$$
$$M' \to *\ M$$
$$M' \to$$
$$F \to x$$
$$F \to y$$
$$F \to (\ E\ )$$

3	**nullable**	**FIRST**	**FOLLOW**
E	false	$\{x,y,(\}$	$\{)\}$
E'	true	$\{+\}$	$\{)\}$
M	false	$\{x,y,(\}$	$\{+,)\}$
M'	true	$\{*\}$	$\{+,)\}$
F	false	$\{x,y,(\}$	$\{*,+,)\}$

	(	**)**	**+**	*****	***x***	***y***
E	$E \to M\ E'$				$E \to M\ E'$	$E \to M\ E'$
E'		$E' \to$	$E' \to +\ E$			
M	$M \to F\ M'$				$M \to F\ M'$	$M \to F\ M'$
M'		$M' \to$	$M' \to$	$M' \to *\ M$		
F	$F \to (\ E\ )$				$F \to x$	$F \to y$

Top-down parsing algorithm

Same as recursive descent but using stack (of symbols) instead of recursion.

```
Push start symbol on stack;
a = first terminal symbol from input;
X = top symbol on stack;
while (stack not empty) {
    if (X == a) {
        pop stack;
        a = next terminal symbol from input;
    }
    else if ( X is a terminal ) error();
    else if ( M[X,a] is empty ) error();
    else if ( M[X,a] = X → Y1 … Yk ) {
        output production X → Y1 … Yk ;
        pop stack;
        push Yk on stack; …; push Y1 on stack;
    }
    X = top symbol on stack;
}
```

Top-down parsing example

a	X	Output	Stack
(	E	$E \to M E'$	$M\ E'$
(	M	$M \to F M'$	$F\ M'\ E'$
(	F	$F \to (E)$	$(\ E\ )\ M'\ E'$
(	(		$E\ )\ M'\ E'$
x	E	$E \to M E'$	$M\ E'\ )\ M'\ E'$
x	M	$M \to F M'$	$F\ M'\ E'\ )\ M'\ E'$
x	F	$F \to x$	$x\ M'\ E'\ )\ M'\ E'$
x	x		$M'\ E'\ )\ M'\ E'$
)	M'	$M' \to$	$E'\ )\ M'\ E'$
)	E'	$E' \to$	$)\ M'\ E'$
)	)		$M'\ E'$
+	M'	$M' \to$	E'
+	E'	$E' \to + E$	$+\ E$
+	+		E
Y	E	$E \to M E'$	$M\ E'$
Y	M	$M \to F M'$	$F\ M'\ E'$
Y	F	$F \to y$	$y\ M'\ E'$
Y	y		$M'\ E'$
\$	M'	$M' \to$	E'
\$	E'	$E' \to$	

$$E \to M\ E'$$
$$E' \to +\ E$$
$$E' \to$$
$$M \to F\ M'$$
$$M' \to *\ M$$
$$M' \to$$
$$F \to x$$
$$F \to y$$
$$F \to (\ E\ )$$

Input string:
$(x)+y$

Test

```
div  : SD text ED
text : item text
text :
item : CHAR
item : div
```

Compute nullable, FIRST, FOLLOW for each of the nonterminals: div, text, item.

Answer

	nullable	FIRST	FOLLOW
`div`	false	`SD`	`CHAR SD ED`
`text`	true	`CHAR SD`	`ED`
`item`	false	`CHAR SD`	`CHAR SD ED`