Introduction to Language Theory and Compilation: Exercises

Session 6: *First* sets, *Follow* sets and LL(1) parsing

ULB

Prediction issues

- We've seen that parsers cannot easily make a choice when several conflicting possibilities arise.
  - Top-down parsing: choice between several Produce actions
  - Bottom-up parsing: choice between Shift and Reduce actions
- We had to rely on (horribly inefficient) backtracking techniques.
- We'll now attempt to make better use of the input to *predict* which action should be taken.
- The objective is to avoid backtracking altogether.

First sets

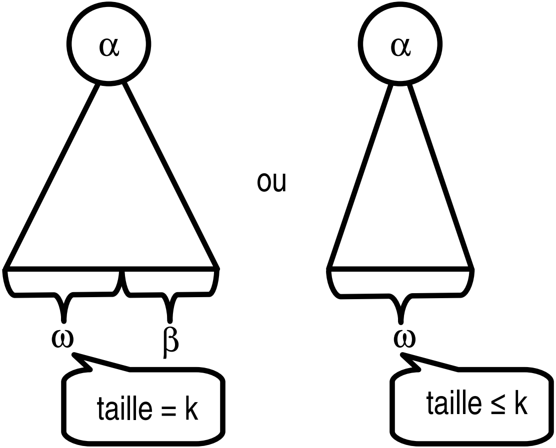
- Let $\alpha \in (V \cup T)^*$ be a sentential form. We define $\mathrm{First}^k(\alpha)$ as the set of strings of k first terminals that can be produced from α.
- Formally:

$$\forall \alpha \in (V \cup T)^* \ : \ \mathrm{First}^k(\alpha) =$$

$$\{\omega \in T^* | \alpha \overset{*}{\Rightarrow} \omega\beta$$

$$\wedge \ \big((|\omega| = k \wedge \beta \in T^*) \vee (|\omega| < k \wedge \beta = \varepsilon)\big)\}$$

ULB


α
α
ou
ω
β
ω
taille = k
taille ≤ k

First sets – construction algorithm

begin

    foreach $a \in T$ **do** $\mathrm{First}^k(a) \leftarrow \{a\}$

    foreach $A \in V$ **do** $\mathrm{First}^k(A) \leftarrow \emptyset$

    repeat

        foreach $A \in V$ **do**

            $\mathrm{First}^k(A) \leftarrow \mathrm{First}^k(A) \cup \{x \in T^* \mid A \rightarrow Y_1 Y_2 \ldots Y_n \wedge x \in \mathrm{First}^k(Y_1) \oplus^k \mathrm{First}^k(Y_2) \oplus^k \cdots \oplus^k \mathrm{First}^k(Y_n)\}$

    until *stability*

ULB

Follow sets

- Let A be a variable. We define $\mathrm{Follow}^k(A)$ as the set of k first terminals of the strings that can follow the productions of A.
- Computation:

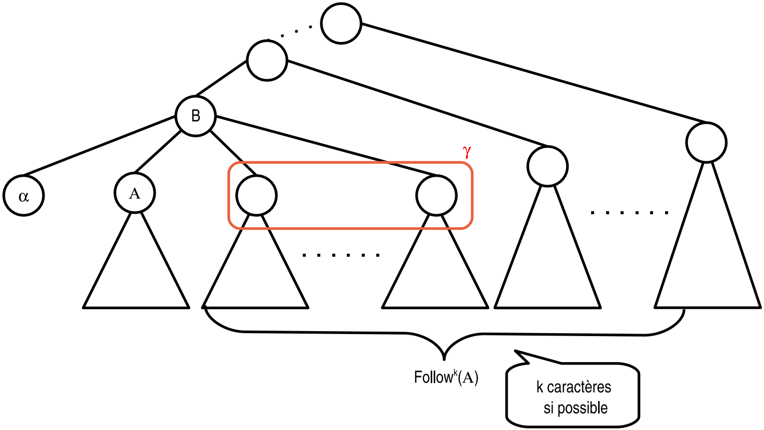
$$\forall A \in V \setminus \{S\} : \mathrm{Follow}^k(A) =$$

$$\bigcup_{B \to \alpha A \gamma} \{\mathrm{First}^k(\gamma) \oplus^k \mathrm{Follow}^k(B)\}$$

$$\bigcup (\mathit{if}\ S \stackrel{+}{\Rightarrow} \alpha A\ \mathit{then}\ \{\varepsilon\}\ \mathit{else}\ \emptyset)$$

ULB

Follow sets – intuitive illustration



Follow sets – construction algorithm

begin

  foreach $A \in V$ **do** $\mathrm{Follow}^k(A) \leftarrow \emptyset$;

  repeat

    if $B \to \alpha A \beta \in P$ **then**

      $\mathrm{Follow}^k(A) \leftarrow \mathrm{Follow}^k(A) \cup \{\mathrm{First}^k(\beta) \oplus^k \mathrm{Follow}^k(B)\}$;

  until *stability*;

Exercise 1

With regards to the grammar given on paper:

1. Give the $\mathrm{First}^1(A)$ and the $\mathrm{Follow}^1(A)$ sets for each $A \in V$.
2. Give the $\mathrm{First}^2(\text{<expression>})$ and the $\mathrm{Follow}^2(\text{<expression>})$ sets.

LL(1) grammars

- A grammar is said to be LL(1) if it can be recognized by a top-down parser with one lookahead symbol.
- Formally:

A grammar G is LL(1) if and only if:
For each production $A \to \alpha_1$ and $A \to \alpha_2$ $(\alpha_1 \neq \alpha_2)$ we have:
$\mathrm{First}^1(\alpha_1 \mathrm{Follow}^1(A)) \cap \mathrm{First}^1(\alpha_2 \mathrm{Follow}^1(A)) = \emptyset$

ULB

Action table

- We can easily construct an *action table* for an LL(1) grammar.
- The table tells us what action should be taken depending on the current symbol on top of the parser's stack and the current lookahead symbol.

$$
\begin{aligned}
S &\rightarrow aS \quad (1)\\
  &\rightarrow b \quad (2)
\end{aligned}
$$

	a	b	c	$\cdots$
S	P1	P2	×	$\cdots$
a	M	×	×	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Action table construction algorithm

begin
- $M \leftarrow \times$;
- **foreach** $A \rightarrow \alpha$ **do**
  - **foreach** $a \in \mathit{First}^1(\alpha)$ **do**
    - $M[A, a] \leftarrow M[A, a] \cup \mathrm{Produce}(A \rightarrow \alpha)$;
  - **if** $\varepsilon \in \mathit{First}^1(\alpha)$ **then**
    - **foreach** $a \in \mathit{Follow}^1(A)$ **do**
      - $M[A, a] \leftarrow M[A, a] \cup \mathrm{Produce}(A \rightarrow \alpha)$;
- **foreach** $a \in T$ **do** $M[a, a] \leftarrow \mathrm{Match}$;
- $M[\$, \varepsilon] \leftarrow \mathrm{Accept}$;

Exercise 2

Which of these grammars are LL(1)?

- 1

$$\begin{aligned} S &\rightarrow ABBA \\ A &\rightarrow a \mid \varepsilon \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

- 2

$$\begin{aligned} S &\rightarrow aSe \mid B \\ B &\rightarrow bBe \mid C \\ C &\rightarrow cCe \mid d \end{aligned}$$

- 3

$$\begin{aligned} S &\rightarrow ABc \\ A &\rightarrow a \mid \varepsilon \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

- 4

$$\begin{aligned} S &\rightarrow Ab \\ A &\rightarrow a \mid B \mid \varepsilon \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

Exercise 3

Give the action table for the following grammar:

(1)	<S>	$\rightarrow$	<expr> $
(2)	<expr>	$\rightarrow$	− <expr>
(3)	<expr>	$\rightarrow$	(<expr>)
(4)	<expr>	$\rightarrow$	<var> <expr-tail>
(5)	<expr-tail>	$\rightarrow$	− <expr>
(6)	<expr-tail>	$\rightarrow$	ε
(7)	<var>	$\rightarrow$	ID <var-tail>
(8)	<var-tail>	$\rightarrow$	(<expr>)
(9)	<var-tail>	$\rightarrow$	ε

ULB

Recursive descent parser

How can we implement a top-down parser?

- We create one function per variable.
- Each function calls a `next_token()` function to read the next token on the input.
- We use the latter as *lookahead symbol* and proceed to pick the right action.
  - Call(s) to `match(token)`
  - Call(s) to a function corresponding to another variable (i.e. Produce)
- We may also use a `syntax_error()` function to explicitly signal that parsing failed.

Recursive descent parser – example

<statement> → id := <expr> ; | read (<id list>) ; | write (<expr list>) ;

```
void statement()
{
   token tok = next_token();
   switch(tok)
   {
   case ID:
       match(ID); match(ASSIGNOP); expression(); match(SEMICOLON);
       break;
   case READ:
       match(READ); match(LPAREN); id_list(); match(RPAREN);
       match(SEMICOLON); break;
   case WRITE:
       match(WRITE); match(LPAREN); expr_list(); match(RPAREN);
       match(SEMICOLON); break;
   default:
       syntax_error(tok); break;
   }
}
```

Exercise 4

Using the grammar given on paper, program a recursive descent parser for rules (14) through (21).