Parsing Theory

Left and right associative operators

To remove the ambiguit and treat "-" as a left associate operator (as usual) we can rewrite the grammar to
$E \to E - T$
$E \to T$
$T \to N$
and to treat "-" as right associative we use
$E \to T - E$
$E \to T$
$T \to N$

Recusive-Descent Parsing

Translating EBNF to BNF

Replace optional construct $[S]$ by a new nonterminal $OptS$:
$OptS \to S|\epsilon$
For example, we can rewrite
$RelCondition \to Exp\ [RelOp\ Exp]$
as
$RelCondition \to Exp\ OptRelExp$
$OptRelExp \to RelOp\ Exp|\epsilon$
Replace grouping construct $(\ S\ )$ by a new nonterminal $GrpS$:
$GrpS \to S$
For example, we can rewrite
$RepF \to$
$(TIMES\ |\ DIVIDE)\ Factor\ RepF\ |\epsilon$
as
$RepF \to TermOp\ Factor\ RepF\ |\epsilon$
$TermOp \to TIMES\ |\ DIVIDE$

parsing example

First & Follow Sets

First Sets

The first set for a construct α records

- the set of terminal symbols α can start with, and
- if α is nullable, it contains the empty string ϵ to indicate that.

Calculating First Sets

Let

- α be a terminal symbok,
- $\alpha_1, \alpha_2, \ldots, \alpha_n$ be strings of (terminal and nonterminal) symbols, and
- A be a nonterminal symbol defined by a single production
$$A \to \alpha_1|\alpha_2|\ldots|\alpha_n,$$

then

$$\begin{aligned} Fst(\epsilon) &= \{\epsilon\} \\ Fst(\alpha) &= \{\alpha\} \\ Fst(\alpha_1|\alpha_2|\ldots|\alpha_n) &= Fst(\alpha_1) \cup Fst(\alpha_2) \\ &\quad \cup \cdots \cup Fst(\alpha_n) \\ Fst(A) &= Fst(\alpha_1|\alpha_2|\ldots|\alpha_n) \end{aligned}$$

Let $S_1, S_2, \ldots, S_n$ be (terminal or nonterminal) symbols, then

$$\begin{aligned} &Fst(S_1 S_2 \ldots S_n) = \\ &\quad Fst(S_1) - \{\epsilon\} \\ &\quad \cup Fst(S_2) - \{\epsilon\} \quad \text{if } S_1 \text{ is nullable} \\ &\quad \cup \ldots \\ &\quad \cup Fst(S_i) - \{\epsilon\} \quad \text{if } S_1\text{-}S_{i-1} \text{ is nullable} \\ &\quad \cup \ldots \\ &\quad \cup Fst(S_n) - \{\epsilon\} \quad \text{if } S_1\text{-}S_{n-1} \text{ is nullable} \\ &\quad \cup \{\epsilon\} \quad \text{if } S_1\text{-}S_n \text{ is nullbale} \end{aligned}$$

Calculating Algorithmically

Start with the first sets for all nonterminals being the empty set and note that the fisrt set for every terminal symbol, α, is the singleton set $\{\alpha\}$. We then make a pass over all production in a grammer considering all atlternatives and process as follows.

- If there is a production of the form $N \leftarrow \epsilon$, we add ϵ to the first set of N.
- If there is a production of the form $N \leftarrow S_1 S_2 \ldots S_n$, then for each $i \in 1..n$, if for all $j \in 1..i-1$, S_j is nullabe, we add the current first set for S_i minus ϵ to the first set for N.
- If every construct $S_1, ..., S_n$ is nullabe, we add ϵ to the first set for N.

We repeat the the passes until no set is modified in the pass, in which case we are finished.

Example

$$A \leftarrow B\,x|C \qquad (1)$$
$$B \leftarrow C\,y|D \qquad (2)$$
$$C \leftarrow D\,z|\epsilon \qquad (3)$$
$$D \leftarrow A\,w \qquad (4)$$

A	$\{\}$	$\{\epsilon\}$	$\{\epsilon, y\}$	$\{\epsilon, y, w\}$	$\{\epsilon, y, w\}$
B	$\{\}$	$\{y\}$	$\{y\}$	$\{y, w\}$	$\{y, w\}$
C	$\{\epsilon\}$	$\{\epsilon\}$	$\{\epsilon, w\}$	$\{\epsilon, w, y\}$	$\{\epsilon, w, y\}$
D	$\{\}$	$\{w\}$	$\{w, y\}$	$\{w, y\}$	$\{w, y\}$

Follow Sets

The follow set for a nonterminal, N, is the set of terminal symbols that may follow N in any context within the grammar. End-of-file is represented by the special terminal symbol \$ in which a follows N.
A nonterminal, N, is followed by a terminal symbol, a, if there is a derivation from $S\$$

Calculating Follow Sets

We compute the Follow set for a nonterminal, N, using two rules.

- If there is a production of the form
$$A \to \alpha N \beta$$
then any symbols that can start β can follow N, and hence $Follow(N)$ must include all the terminal symbols in $First(\beta)$. Note that ϵ is not included even if it appears in $First(\beta)$.
$$First(\beta) - \{\epsilon\} \subseteq Follow(N)$$
- If there is a production of the form
$$A \to \alpha N \beta$$
and β is nullable, then any token that can follow A can also follow N. Hence,
$$Follow(A) \subseteq Follow(N)$$
The case where β is nullable includes the case when β is empty and the production is of the form
$$A \to \alpha N$$

Calculating Algorithmically

Start with all nonterminal symbols having an empty follow set, $\{\}$, except for the start symbol, S, which has the follow set $\{\$\}$.
We make a pass through the grammar examining the right side of every production. For each occurence of a nonterminal within the right side of some production, we augment the Follow set for that nonterminal according to the following process. Assume we are processing an occurence of N and the production is of the form
$$A \to \alpha N \beta$$
we add $First(\beta) - \{\epsilon\}$ to the Follow set computer for N so far, and if β is nullable, we also add the current Follow set for A to the Follow set for N.
After making a complete pass, we repeat the process with the Follow sets computed so far until no Follow sets are modified, in which case we are done.

Example

$$S \to xAB \qquad Fst(S) = \{x\}$$
$$A \to y|zB \qquad Fst(A) = \{y, z\}$$
$$B \to \epsilon|Ax \qquad Fst(B) = \{\epsilon, y, z\}$$

S	$\{\$\}$	$\{\$\}$	$\{\$\}$	$\{\$\}$
A	$\{\}$	$\{y, z, \$, x\}$	$\{y, z, \$, x\}$	$\{y, z, \$, x\}$
B	$\{\}$	$\{\$, y, z\}$	$\{\$, y, z, x\}$	$\{\$, y, z, x\}$

LL(1) Grammar

A BNF grammer is LL(1) if for each nonterminal, N, wher $N \to \alpha_1|\alpha_2|\ldots|\alpha_n$,

- the First sets for each pair of alternatives for N are disjoint, and
- if N is nullable, $First(N)$ and $Follow(N)$ are disjoin.

Left Factoring & Left Recursion

Left Factoring Productions

Not all EBNF grammars are suitable for Recursive-Descent Parsing, however, sometimes we can rewrite them into a form that is suitable.

Left Factor Rewriting Rule

To remove the left factor from
$$A \to \alpha\ \beta\ |\ \alpha\ \gamma,$$
we can rewrite the production using an aditional nonterminal A' as
$$A \to \alpha\ A'$$
$$A' \to \beta\ |\ \gamma$$

Left Recursive Productions

A production of the form
$$E \to E\ +\ T\ |\ T$$
is not suitable for RDP because the left recursion in the grammar leads to an infinite recursion.

Immediate Left Recursion Rewriting Rule (Simple)

To remove the left recursion from
$$A \to A\ \alpha\ |\ \beta,$$
we can rewrite the production as
$$A \to \beta\ A'$$
$$A' \to \epsilon\ |\ \alpha\ A'$$

Immediate Left Recursion Rewriting Rule (General)

To remove the left recursion from the general case

$$A \to A\alpha_1|A\alpha_2|\dots|A\alpha_n|\beta_1|\beta_2|\dots|\beta_m,$$

we can use grouping to see the structure in the same form as the simple case

$$A \to A(\alpha_1|\alpha_2|\dots|\alpha_n)|(\beta_1|\beta_2|\dots|\beta_m)$$

which allows us to rewrite the production as follows,

$$A \to (\beta_1|\beta_2|\dots|\beta_m)A'$$
$$A' \to \epsilon|(\alpha_1|\alpha_2|\dots|\alpha_n)A'$$

Indirect Left Recursion Rewriting Rule

The following productions have indirect recursion from $A \to B \to C \to A$.

$$A \to B\ \alpha$$
$$B \to C\ \beta$$
$$C \to A\ \gamma_1 \mid \gamma_2$$

To remove the recursion we can first collapse B into A as follows.

$$A \to C\ \beta\ \alpha$$
$$C \to A\ \gamma_1 \mid \gamma_2$$

Then we can collapse C into A.

$$A \to (A\ \gamma_1 \mid \gamma_2)\ \beta\ \alpha$$
$$\downarrow$$
$$A \to A\ \gamma_1\ \beta\ \alpha \mid \gamma_2\ \beta\ \alpha$$

This now leaves a simple direct left recursion which we can remove as follows.

$$A \to \gamma_2\ \beta\ \alpha\ A'$$
$$A' \to \gamma_1\ \beta\ \alpha\ A'$$