13 FIRST and FOLLOW Sets

Wednesday, March 20, 2024 12:15 PM

Issue: Guiding the algorithm.

$A \to B \mid C \mid d$

parseA()

? which to call?

- if we know what terminals a B can begin with and what terminals a C can begin with then. compare the token with these sets of terminals to make a decision. (assuming these sets don't intersect)

- FIRST Sets

A set of terminals a derivation from a non-terminal can begin with,

$$FIRST(\alpha) = \{a \mid \alpha \overset{*}{\Rightarrow} a\beta\}$$

Example:

$S \to AB \mid BA$
$A \to aB \mid cS$
$B \to bc$

$FIRST(a) = \{a\}$
$FIRST(b) = \{b\}$
$FIRST(c) = \{c\}$

$FIRST(B) = \{b\}$ $\quad B \to bc$

$FIRST(A) = \{a, c\}$ $\quad A \to aB$, $A \to cS$

$FIRST(S) = \{a, b, c\}$

$S \to AB \to aBB$
$S \to BA \to bcA$

$S \to AB \to cSB$

Algorithm:

1) IF X is a terminal $FIRST(X) = \{X\}$

else Repeat:

2) IF $X \to \lambda$ THEN add λ to $FIRST(X)$

3) IF $X \to Y_1 Y_2 Y_3 \dots Y_k$ THEN

a) IF for some j, $\lambda \in FIRST(Y_j)$ for every $j < i$ and $a \in FIRST(Y_i)$ then add a to $FIRST(X)$

b) IF for all i, $1 \le i \le k$, $\lambda \in FIRST(Y_i)$

b) IF for all i, $1 \le i \le k$, $\lambda \in FIRST(Y_i)$
then add λ to $FIRST(X)$

E.G.

$S \to ABCD$
$D \to AB$
$A \to a \mid \lambda$
$B \to b \mid \lambda$
$C \to c$

$FIRST(A) = \{\lambda, a\}$
$FIRST(B) = \{\lambda, b\}$
$FIRST(C) = \{c\}$
$FIRST(D) = \{\lambda, a, b\}$

$FIRST(a) = \{a\}$
$FIRST(b) = \{b\}$
⋮

$FIRST(S) = \{c, a, b\}$

$D \to AB \to aB$
$B \to b$
$\to \lambda$

$S \to ABCD$
$Y_1\ Y_2\ Y_3\ Y_4$
$S \to ABCD \to aBCD$
$S \to ABCD$

E.G.

$E \to TE'$
$E' \to +TE' \mid \lambda$
$T \to FT'$
$T' \to *FT' \mid \lambda$
$F \to int \mid (E)$

	FIRST
E	int (
E'	+ λ
T	int (
T'	* λ
F	int (

- USING FIRST SETS

To guide our parser.

$A \to B \mid C \mid d$

```
FUNCTION parse_A()
  IF token in FIRST(B) THEN
    parse_B()
  ELSIF token in FIRST(C) THEN
    parse_C()
  ELSIF token = 'd' THEN
    get_token()
…
END.
```

Of course, we are assuming that

$FIRST(B) \cap FIRST(C) = \emptyset$
$FIRST(B) \cap FIRST(d) = \emptyset$
$FIRST(C) \cap FIRST(d) = \emptyset$

"pairwise disjoint"

If not the case, rewrite the grammar.

E.G.

$A \to dB \mid dC \mid d$
$B \to ax$
$C \to ay$

⇓ Left Factor

$A \to dA'$
$A' \to B \mid C \mid \lambda$

⇓ Factorization

$A \to dA'$
$A' \to aA'' \mid \lambda$
$A'' \to B' \mid C'$
$B' \to x$
$C' \to y$

- FOLLOW Sets

Intuition: For a non-terminal, are the terminals that can appear immediately to it's right in a derivation.

e.g. $S \xrightarrow{*} ABC \xrightarrow{*} AbcC$

so $b \in FOLLOW(A)$

Definition

$FOLLOW(A) = \{a \mid S \xrightarrow{*} \delta A a \Gamma\}$

Note:

$\$ \in FOLLOW(S)$ where S is the start symbol

E.g.

Example:

$S \to AB \mid BA$
$A \to aB \mid cS$
$B \to bc$

$FOLLOW(S) = \{\$, b\}$
$FOLLOW(A) = \{b, \$\}$
$FOLLOW(B) = \{a, c, \$, b\}$

$S \to AB \to Abc$
$S \to BA \to BaB$

$S \to BA \to BcS$ $\quad S \to AB \to Abc \to cSbc$

$S\$$ $\quad S \to BA\$$ $\quad S \to AB\$$ $\quad S \to AB \to Abc \to aBbc$

Algorithm:

// to compute FOLLOW sets

1) add \$ to FOLLOW(S), where S is the Start Symbol

REPEAT:

2) IF $A \to \alpha B \beta$ THEN add everything in FIRST(β) (except λ) to FOLLOW(B)

3) IF $A \to \alpha B$ or ($A \to \alpha B \Gamma$ and λ in FIRST(Γ)) THEN add everything in FOLLOW(A) to FOLLOW(B)

Note: λ is never in a FOLLOW set.

3) $A \to \alpha B$ $\quad S \to^* Ax \to \alpha Bx$

$A \to \alpha B \Gamma$, $\Gamma \to^* \lambda$ $\quad S \to^* Ax \to \alpha B \Gamma x \to^* \alpha Bx$

E.G.

$E \to TE'$
$E' \to +TE' \mid \lambda$
$T \to FT'$
$T' \to *FT' \mid \lambda$
$F \to int \mid (E)$

	FIRST	FOLLOW
E	int (	\$)
E'	+ λ	\$)
T	int (	+ \$)
T'	* λ	+ \$)
F	int (	* + \$)

USES:

- Error recovery on a recursive descent parser.

Intuition

Suppose: parse_A()

$A \to \sim$ $\quad$ FIRST(A) = { a b c }
FOLLOW(A) = { x y z }

Let token = p

Pseudocode.

- if token is not in the FIRST set
  - print error
  - read tokens until you find a token in the FIRST set.
- parse_A as usual

- if token is not in the FOLLOW set.
  - print error.
  - read tokens until you find a token in the FOLLOW set.

Example implementation:

A → a B b B → c

```
FUNCTION old_parse_A()
  IF token = 'a' THEN
    getToken()
    parse_B()
    IF token = 'b' THEN
      getToken()
```

```
FUNCTION new_parse_A()
  WHILE token not in FIRST(A) DO
    error(token)
    getToken()

  old_parse_A()

  WHILE token not in FOLLOW(A) DO
    error(token)
    getToken()
```

trace.

x x a c b y y

parse_A()
  parse_B()

```
// version 2 : handle a missing A
FUNCTION new_parse_A()
  WHILE token not in FIRST(A) Union FOLLOW(A) DO
    error(token)
    getToken()

  IF token in FIRST(A)
    old_parse_A()

    WHILE token not in FOLLOW(A) DO
      error(token)
      getToken()

  ELSE
    error("expecting A")
```

—o— EOF.