13 FIRST and FOLLOW Sets

Wednesday, April 2, 2025 12:35 PM

Motivation: Guide the Algorithm.

$A \to B \mid C \mid d$ (? between B and C)

idea: in parse_A() use the token to decide → parse_B() / parse_C()

If we know what terminals can B begin with
and what terminals can C begin with
then
compare token with these sets to make a decision.

- **FIRST Sets:**

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

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

E.g.

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

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

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

$S \to AB \to aBB$
$S \to BA \to bcA$
$A \to aB$
$B \to bc$
$S \to AB \to cSB$

- Algorithm:

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

Repeat:

2) if $X \to \lambda$ then add λ to $FIRST(X)$

3) if $X \to Y_1 Y_2 Y_3 Y_4 \ldots Y_k$ then:

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

and $a \in FIRST(Y_i)$
then add a to $FIRST(X)$

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

E.G.

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

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

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

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

$D \to AB \to B \to b$

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

Check
FIRST(B)
FIRST(C)
FIRST(d)
} should be mutually exclusive.

```
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
END.
```

So:
1.- Compute First Sets
2.- check for "pairwise disjointness"

fix by either reading more tokens or change the grammar.

E.G.

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

⇓ left factoring

$A \to dA'$
$A' \to B \mid C \mid \varepsilon$
$B \to ax$
$C \to ay$

⇓ factor common elements

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

- FOLLOW Sets:

Intuition: for a non-terminal X the follow set is the set of terminals that can appear immediately to the right of X in a derivation.

E.g.

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

$S \to AB \to Abc$ so $b \in FOLLOW(A)$

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

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

cont.

$FOLLOW(S) = \{\$, b\}$ — $S \to BA \to BaB$
$FOLLOW(A) = \{b, \$\}$ — $S \to BA \to Bc S$
$FOLLOW(B) = \{a, c, b, \$\}$ — $S \to AB \to Abc \to aBbc$

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

S → BA → Bcs
S → AB → Abc → aBbc
S → AB → Abc → cSbc

S$ → AB$ S$ → BA$

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) to FOLLOW(B)

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

Note: λ is never in a FOLLOW set.

e.g. rule 3)

$S \to Ax \to \alpha Bx$ $S \to Ax \to \alpha B \Gamma x \to \alpha Bx$

E.G.

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

	FIRST	FOLLOW
S	a b c	$
D	a b λ	$
A	a λ	b c $
B	b λ	c $
C	c	a b $

S → ABCD → ACD → AcD
S$ → ABCD$ → ABCAB$
→ ABCA$ → ABC$

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 (	* +) $

USE: -Error Recovery on recursive descent parsers

Intuition:

$A \rightarrow \alpha$ expect: token $\in$ FIRST(A)

parse_A() expect token $\in$ FOLLOW(A)

Algorithm:

If we need to call parse_A()

Consume tokens until token $\in$ (FIRST(A) $\cup$ FOLLOW(A))

If token $\in$ FIRST(A)

parse_A()

Consume tokens until token $\in$ FOLLOW(A)

Code:

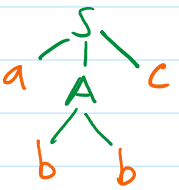
```
FUNCTION OldParse_A ( )
 …
 …
```

```
FUNCTION NewParse_A ( )
  WHILE token not in ( FIRST(A) union FOLLOW(A) ) DO
    error(token)
    gotToken()

  IF token in FIRST(A)
    OldParse_A()

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

Trace:

$S \rightarrow aAc$

$A \rightarrow bb$

	FIRST	FOLLOW
A	b	c

w = a * * bb yy c