

UNIVERSITY OF TWENTE.



# COMPILER CONSTRUCTION

## CC 2-3: FOLLOW, FIRST+

MODULE 8: PROGRAMMING PARADIGMS

1 MAY 2019





# FIRST-COMPUTATION

Y → X z

The idea is straightforward

- Function from symbols to sets of initial terminals in RHS
  - The *FIRST* of a terminal is that terminal itself  $FIRST(a) = \{a\}$
- Let the RHS start with  $X$  (terminal or nonterm.): copy *FIRST*( $X$ ) except  $\epsilon$ 
  - If  $\epsilon$  in *FIRST*( $X$ ), also look at *second* symbol of RHS
  - Continue with 3rd etc., as long as  $\epsilon$  is in *FIRST* of all previous symbols
- Include  $\epsilon$  if the RHS consists of only non-terminals with  $\epsilon$  in their *FIRST*
  - RHS equal to  $\epsilon$  is a special case of this
- Iterate the above until all *FIRST*-sets are stable
  - Iteration is necessary to cope with chain of initial non-terminals

$$\begin{aligned} A &\rightarrow a B \\ B &\rightarrow b C | \epsilon \\ C &\rightarrow c \end{aligned}$$

$$\begin{aligned} A &\rightarrow B a \\ B &\rightarrow C b | \epsilon \\ C &\rightarrow c \end{aligned}$$

$$\begin{aligned} A &\rightarrow B C | a \\ B &\rightarrow C b | \epsilon \\ C &\rightarrow c | \epsilon \end{aligned}$$

*FIRST*(.)

A	∅	{a}	{a}	∅	∅	{a}	{a, c}	∅	{a}	{a, c, ε}	{a, c, ε, b}
B	∅	{b, ε}	{b, c}	∅	{ε}	{ε, c}	{ε, c}	∅	{ε}	{ε, c, b}	{ε, c, b}
C	∅	{c}	{c}	∅	{c}	{c}	{c}	∅	{c, c}	{c, ε}	{c, ε}

↓

# FIRST-COMPUTATION

The idea is straightforward

1. Function from symbols to sets of initial terminals in RHS
  - The *FIRST* of a terminal is that terminal itself
2. Let the RHS start with  $X$  (terminal or nonterm.): copy *FIRST*( $X$ ) except  $\epsilon$ 
  - If  $\epsilon$  in *FIRST*( $X$ ), also look at *second* symbol of RHS
  - Continue with 3rd etc., as long as  $\epsilon$  is in *FIRST* of all previous symbols
3. Include  $\epsilon$  if the RHS consists of only non-terminals with  $\epsilon$  in their *FIRST*
  - RHS equal to  $\epsilon$  is a special case of this
4. Iterate the above until all *FIRST*-sets are stable
  - Iteration is necessary to cope with chain of initial non-terminals

$A \rightarrow a B$
$B \rightarrow b C \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B C \mid a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c \mid \epsilon$

Omitting  
first & last stage  
in other examples

	0	1	2
$A$	$\emptyset$	$a$	$a$
$B$	$\emptyset$	$b \epsilon$	$b \epsilon$
$C$	$\emptyset$	$c$	$c$

	1	2	3
$A$	$\emptyset$	$a$	$a c$
$B$	$\epsilon$	$\epsilon c$	$\epsilon c$
$C$	$c$	$c$	$c$

	1	2	3
$A$	$a$	$a c \epsilon$	$a c \epsilon b$
$B$	$\epsilon$	$\epsilon c b$	$\epsilon c b$
$C$	$c \epsilon$	$c \epsilon$	<u><math>c \epsilon</math></u>

# FIRST-ALGORITHM FROM THE BOOK

- First: initialisation, set all *FIRST*s to  $\emptyset$

- Then: `while (FIRST sets are still changing) do;` ← iterate until stable

*P* = set of rules

Collect *FIRST* for *A*

from left to right  
go through RHS  
as long as  $\epsilon$  remains

```

while (FIRST sets are still changing) do;
  for each  $p \in P$ , where  $p$  has the form  $A \rightarrow \beta$  do;
    if  $\beta$  is  $\beta_1 \beta_2 \dots \beta_k$ , where  $\beta_i \in T \cup NT$ , then begin;
      rhs ← FIRST( $\beta_1$ ) - { $\epsilon$ };
      i ← 1;
      while ( $\epsilon \in \text{FIRST}(\beta_i)$  and  $i \leq k-1$ ) do;
        rhs ← rhs  $\cup$  (FIRST( $\beta_{i+1}$ ) - { $\epsilon$ });
        i ← i + 1;
      end;
    end;
    if  $i = k$  and  $\epsilon \in \text{FIRST}(\beta_k)$ 
      then rhs ← rhs  $\cup$  { $\epsilon$ };
    FIRST( $A$ ) ← FIRST( $A$ )  $\cup$  rhs;
  end;
end;

```

← copy *FIRST* (except  $\epsilon$ )  
of initial symbol of RHS (=  $\beta_1$ )

← copy *FIRST* (except  $\epsilon$ )  
of next symbol of RHS

reached the end and  $\epsilon$  still there?  
OK, put  $\epsilon$  in *FIRST*(*A*)

# FOLLOW-COMPUTATION

Trickier than *FIRST*

1. Function from non-terminals *A* to sets of terminals that may follow *A*
  - To determine, look at occurrences of *A* in RHSs of other rules
2. Initialize *FOLLOW* with { eof } for start symbol,  $\emptyset$  for all others
3. Work through RHS from right to left (to cater for  $\epsilon$ )
  - *B* at the end of RHS of *A*: copy *FOLLOW*(*A*) to *FOLLOW*(*B*) *A* → .....*B*
  - Whatever may follow *A* may also follow *B*
  - *B* not at the end: copy *FIRST* of symbol to the right of *B* to *FOLLOW*(*B*)
4. Accumulate as long as  $\epsilon$  in *FIRST* of RHS symbols
  - If pattern *ACb* is in RHS and  $\epsilon$  is in *FIRST*(*C*), put *b* in *FOLLOW*(*A*)

$A \rightarrow a B$
$B \rightarrow b C \mid \epsilon$
$C \rightarrow c$

$A \rightarrow \underline{B} a$
$B \rightarrow \underline{C} b \mid \epsilon$
$C \rightarrow c$

$\underline{A} \rightarrow B C \mid a$
$B \rightarrow \underline{C} b \mid \epsilon$
$C \rightarrow c \mid \epsilon$

*FOLLOW*( $\rightarrow$ )

<i>A</i>	{ eof }	{ eof }	{ eof }	eof	eof	eof	eof eof
<i>B</i>	$\emptyset$	{ eof }	{ eof }	$\emptyset$	a	a	$\emptyset$ c eof
<i>C</i>	$\emptyset$	$\emptyset$	{ eof }	$\emptyset$	b	b	$\emptyset$ eof b

# FOLLOW-COMPUTATION

Trickier than *FIRST*

1. Function from non-terminals  $A$  to sets of terminals that may *follow*  $A$ 
  - To determine, look at occurrences of  $A$  in RHSs of *other* rules
2. Initialize *FOLLOW* with  $\{ \text{eof} \}$  for start symbol,  $\emptyset$  for all others
3. Work through RHS from right to left (to cater for  $\epsilon$ )
  - $B$  at the end of RHS of  $A$ : copy *FOLLOW*( $A$ ) to *FOLLOW*( $B$ )
    - Whatever may follow  $A$  may also follow  $B$
  - $B$  not at the end: copy *FIRST* of symbol to the right of  $B$  to *FOLLOW*( $B$ )
4. Accumulate as long as  $\epsilon$  in *FIRST* of RHS symbols
  - If pattern  $ACb$  is in RHS and  $\epsilon$  is in *FIRST*( $C$ ), put  $b$  in *FOLLOW*( $A$ )

$A \rightarrow a B$
$B \rightarrow b C \mid \epsilon$
$C \rightarrow c$

	0	1	2
$A$	eof	eof	eof
$B$	$\emptyset$	eof	eof
$C$	$\emptyset$	$\emptyset$	eof

$A \rightarrow B a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c$

	0	1
$A$	eof	eof
$B$	$\emptyset$	a
$C$	$\emptyset$	b

$A \rightarrow B C \mid a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c \mid \epsilon$

	0	1
$A$	eof	eof
$B$	$\emptyset$	eof c
$C$	$\emptyset$	eof b

# WHAT'S THIS ABOUT EOF?

- Take into account that file *should* end after start symbol
  - Fake terminal eof, conceptually insert at end of file
- Also conceptually insert at end of every RHS of start symbol
  - That can go wrong if start symbol appears in some other RHS

$\begin{array}{l} A \rightarrow a B \mid \epsilon \\ B \rightarrow b C \mid \epsilon \\ C \rightarrow A c \end{array}$	not equivalent to	$\begin{array}{l} A \rightarrow a B \text{ eof} \mid \text{eof} \\ B \rightarrow b C \mid \epsilon \\ C \rightarrow A c \end{array}$
--	-------------------	--

- Don't *really* insert in rule; this is the intuition about the treatment of eof

Try to not make the start symbol appear on any RHS.

# FOLLOW-ALGORITHM FROM THE BOOK

- First: initialisation, set  $FOLLOW(S)$  to eof, all others to  $\emptyset$

- Then: 

```
while (FOLLOW sets are still changing) do; ← iterate until stable
  for each  $p \in P$  of the form  $A \rightarrow \beta_1\beta_2 \dots \beta_k$  do;
```

$TRAILER$  will be copied to  $FOLLOW$  of next symbol to the left

go from right to left through RHS  $\beta_1\beta_2 \dots \beta_k$

```

TRAILER ← FOLLOW(A);
for i ← k down to 1 do;
  if  $\beta_i \in NT$  then begin;
    FOLLOW( $\beta_i$ ) ← FOLLOW( $\beta_i$ )  $\cup$  TRAILER;
    if  $\epsilon \in FIRST(\beta_i)$ 
      then TRAILER ← TRAILER  $\cup$  (FIRST( $\beta_i$ ) -  $\epsilon$ );
      else TRAILER ← FIRST( $\beta_i$ );
    end;
  else TRAILER ← FIRST( $\beta_i$ ); // is  $\{\beta_i\}$ 
  end;
end;
end;
end;

```

update  $FOLLOW$  using  $TRAILER$

compute new  $TRAILER$

accumulate if  $\epsilon$  in  $FIRST$

current symbol was terminal; compute new  $TRAILER$

# AND NOW... FIRST+!

- All this work was just to compute the *FIRST+*-set of a rule  $A \rightarrow \beta$ 
  - In principle  $FIRST+(A \rightarrow \beta)$  equals  $FIRST(\beta)$  (minus  $\epsilon$ )
    - What's  $FIRST(\beta)$ ? We only have  $FIRST$  for single symbols!

$$\beta = \beta_1 \beta_2 \dots \beta_k$$

$$FIRST(\beta) = \begin{cases} \epsilon & \text{if } \beta = \epsilon \\ FIRST(\beta_1) & \text{if } \beta = \beta_1 \beta' \text{ and } \epsilon \text{ is not in } FIRST(\beta_1) \\ FIRST(\beta_1) \setminus \{\epsilon\} \cup FIRST(\beta') & \text{otherwise} \end{cases}$$

- This is essentially the same computation we already did for *FIRST*
- If  $\epsilon$  is in  $FIRST(\beta)$ , we add  $FOLLOW(A)$  to  $FIRST+(A \rightarrow \beta)$
- All together now:

$$FIRST^+(A \rightarrow \beta) = \begin{cases} FIRST(\beta) & \text{if } \epsilon \notin FIRST(\beta) \\ FIRST(\beta) \cup FOLLOW(A) & \text{otherwise} \end{cases}$$

Note that  $\epsilon$  may end up in *FIRST+*, but it's quite useless and will be omitted in these slides

- Remember: LL(1)-criterion
  - $FIRST+(A \rightarrow \beta_1)$  and  $FIRST+(A \rightarrow \beta_2)$  may not overlap for the same  $A$ !

# EXAMPLES

- Grammar

$A \rightarrow a B$
$B \rightarrow b C \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B C \mid a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c \mid \epsilon$

- FIRST*

A	a
B	b $\epsilon$
C	c

A	a c
B	$\epsilon$ c
C	c

A	a c $\epsilon$ b
B	$\epsilon$ c b
C	c $\epsilon$

- FOLLOW*

A	eof
B	eof
C	eof

A	eof
B	a
C	b

A	eof
B	eof c
C	eof b

- FIRST+*

# EXAMPLES

▪ Grammar

$A \rightarrow a B$
$B \rightarrow b C \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c$

$A \rightarrow B C \mid a$
$B \rightarrow C b \mid \epsilon$
$C \rightarrow c \mid \epsilon$

▪ FIRST

A	a
B	b $\epsilon$
C	c

A	a c
B	$\epsilon$ c
C	c

A	a c $\epsilon$ b
B	$\epsilon$ c b
C	c $\epsilon$

▪ FOLLOW

A	eof
B	eof
C	eof

A	eof
B	a
C	b

A	eof
B	eof c
C	eof b

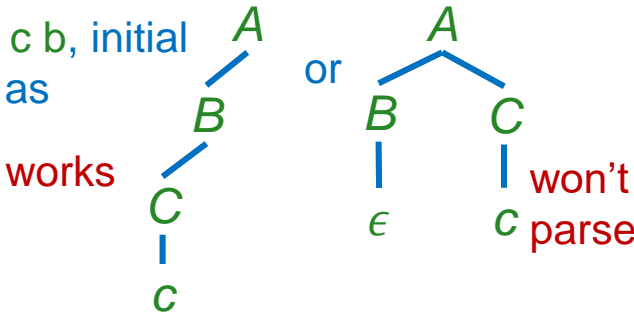
▪ FIRST+

$B \rightarrow b C$	b
$B \rightarrow \epsilon$	eof

$B \rightarrow C b$	c
$B \rightarrow \epsilon$	a

$A \rightarrow B C$	eof c b
$A \rightarrow a$	a
$B \rightarrow C b$	c b
$B \rightarrow \epsilon$	eof c
$C \rightarrow c$	c
$C \rightarrow \epsilon$	eof b

Not LL(1): in input string  $c b$ , initial symbol  $c$  can be parsed as



Overlap!