Programming Paradigms

Syntax (Part 5)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Specifying syntax**
  - Regular expressions
  - Context-free grammars
- **Scanning**
- **Parsing**
  - Top-down parsing
  - Bottom-up parsing

Generating a Top-Down Parser

- **To generate an LL(k) parser, need to predict which rule to apply**
- **Compute PREDICT sets for all productions, based on two helpers**
  - FIRST(N): What terminals come first when expanding non-terminal N?
  - FOLLOW(N): What terminals follow after non-terminal N?

FIRST Sets

FIRST(A): Set of all terminals that can begin a derivation starting with A

Example:

S $\rightarrow$ `simple` | `begin` **S** `end`

FIRST(S) = $\{$ `simple`, `begin` $\}$

Computing FIRST sets

$$\text{FIRST}(\varepsilon) = \{\varepsilon\}$$

$$\text{FIRST}(A\alpha) = \begin{cases} \{A\} & \text{if } A \text{ is terminal} \\ (\text{FIRST}(A) \setminus \{\varepsilon\}) \cup \text{FIRST}(\alpha) & \text{if } A \Rightarrow^* \varepsilon \\ \text{FIRST}(A) & \text{otherwise} \end{cases}$$

For a given grammar:

Apply above recursively until all FIRST sets remain constant

Example 1

$S \to a\,S\,e$

$S \to B$

$B \to b\,B\,e$

$B \to C$

$C \to c\,C\,c$

$C \to d$

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

$\text{FIRST}(B) = \{b, c, d\}$

$\text{FIRST}(C) = \{d, c\}$

Example 2

$P \rightarrow i \mid c \mid n\,T\,S$

$Q \rightarrow P \mid a\,S \mid d\,S\,c\,S\,T$

$R \rightarrow b \mid \varepsilon$

$S \rightarrow e \mid R\,n \mid \varepsilon$

$T \rightarrow R\,S\,q$

$FIRST(P) = \{ i, c, n \}$

$FIRST(Q) = \{ a, d, i, c, n \}$

$FIRST(R) = \{ b, \varepsilon \}$

$FIRST(S) = \{ e, \varepsilon, b, n \}$

$FIRST(T) = \{ b, \cancel{\varepsilon}, e, n, q \}$

corrected

Quiz: FIRST Sets

$\mathbf{S} \rightarrow \mathbf{a\ S\ e} \mid \mathbf{S\ T\ S}$

$\mathbf{T} \rightarrow \mathbf{R\ S\ e} \mid \mathbf{Q}$

$\mathbf{R} \rightarrow \mathbf{r\ S\ r} \mid \epsilon$

$\mathbf{Q} \rightarrow \mathbf{S\ T} \mid \epsilon$

Compute the FIRST sets of all non-terminals. What is the sum of the sizes of these sets?

Please vote via Ilias.

Quiz:

$FIRST(S) = \{a\}$

$FIRST(T) = \{r, a, \varepsilon\}$

$FIRST(R) = \{r, \varepsilon\}$

$FIRST(Q) = \{\varepsilon, a\}$

$\Sigma = 8$

FOLLOW Sets

FOLLOW(A): Set of all terminals that may follow A in some derivation

- Including special symbol EOF for "end of file"
- Never includes ϵ

Example:

S $\rightarrow$ **a B c**	**FOLLOW(S) =** $\{$ **EOF** $\}$
B $\rightarrow$ **d**	**FOLLOW(B) =** $\{$ **c** $\}$

Computing FOLLOW Sets

To compute FOLLOW(A), apply these rules until all FOLLOW sets constant

- If A is start symbol, put EOF in FOLLOW(A)
- Productions of the form B $\rightarrow \alpha$ A β:

  Add FIRST(β) - $\{ \epsilon \}$ to FOLLOW(A)
- Productions of the form

  B $\rightarrow \alpha$ A, or

  B $\rightarrow \alpha$ A β where $\beta \Rightarrow^* \epsilon$:

  Add FOLLOW(B) to FOLLOW(A)

Example 1

$S \to a\,S\,e \mid B$

$B \to b\,B\,C\,f \mid C$

$C \to c\,C\,g \mid d \mid \varepsilon$

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

$FIRST(B) = \{b, c, d, \varepsilon\}$

$FIRST(C) = \{c, d, \varepsilon\}$

$FOLLOW(S) = \{EOF, e\}$

$FOLLOW(B) = \{c, d, f, EOF, e\}$

$FOLLOW(C) = \{f, c, d, EOF, e, g\}$

Quiz: FOLLOW Sets

$\mathbf{S} \rightarrow \mathbf{a\ S\ e} \mid \mathbf{S\ T\ S}$

$\mathbf{T} \rightarrow \mathbf{R\ S\ e} \mid \mathbf{Q}$

$\mathbf{R} \rightarrow \mathbf{r\ S\ r} \mid \epsilon$

$\mathbf{Q} \rightarrow \mathbf{S\ T} \mid \epsilon$

Compute the FOLLOW sets of all non-terminals. What is the sum of the sizes of these sets?

Please vote via Ilias.

Quiz:

	FIRST	FOLLOW
S	a	EOF, e, r, a
T	r, a, ε	a
R	r, ε	a
Q	a, ε	a

$\Sigma = 7$

PREDICT Sets

PREDICT set for a rule: Which terminals to look for in LL(1) parser

- If next input token is in PREDICT of rule, apply the rule
- Computing the PREDICT set for rule A $\rightarrow \alpha$:
  - If ϵ in FIRST(α):

    PREDICT(A $\rightarrow \alpha$) = (FIRST(α) - $\{ \epsilon \}$) $\cup$ FOLLOW(A)
  - Otherwise:

    PREDICT(A $\rightarrow \alpha$) = FIRST(α)

Example

Grammar:

S $\rightarrow$ `a` **B**

S $\rightarrow$ `b` **C**

B $\rightarrow$ `b` `b` **C**

C $\rightarrow$ `c` `c`

	FIRST	FOLLOW
S	a, b	EOF
B	b	EOF
C	c	EOF

Example

Grammar:	PREDICT:
S → a B	{ a }
S → b C	{ b }
B → b b C	{ b }
C → c c	{ c }

	FIRST	FOLLOW
S	a, b	EOF
B	b	EOF
C	c	EOF

Example

Grammar:	PREDICT:
S → a B	{ a }
S → b C	{ b }
B → b b C	{ b }
C → c c	{ c }

	FIRST	FOLLOW
S	a, b	EOF
B	b	EOF
C	c	EOF

```
S() {
  if (inputToken == a)
    match(a); B();
  else if (inputToken == b)
    match(b); C();
  else error();
}
B() {
  if (inputToken == b)
    match(b); match(b); C();
  else error()
}
C() {
  if (inputToken == c)
    match(c); match(c);
  else error()
}
```

Example

Grammar:	PREDICT:
$S \rightarrow$ a B	{ a }
$S \rightarrow$ b C	{ b }
$B \rightarrow$ b b C	{ b }
$C \rightarrow$ c c	{ c }

	FIRST	FOLLOW
S	a, b	EOF
B	b	EOF
C	c	EOF

```
S() {
  if (inputToken == a)
    match(a); B();
  else if (inputToken == b)
    match(b); C();
  else error();
}
B() {
  if (inputToken == b)
    match(b); match(b); C();
  else error()
}
C() {
  if (inputToken == c)
    match(c); match(c);
  else error()
}
```

Example

Grammar:	PREDICT:
$S \rightarrow$ a B	{ a }
$S \rightarrow$ b C	{ b }
$B \rightarrow$ b b C	{ b }
$C \rightarrow$ c c	{ c }

	FIRST	FOLLOW
S	a, b	EOF
B	b	EOF
C	c	EOF

```
S() {
  if (inputToken == a)
    match(a); B();
  else if (inputToken == b)
    match(b); C();
  else error();
}
B() {
  if (inputToken == b)
    match(b); match(b); C();
  else error()
}
C() {
  if (inputToken == c)
    match(c); match(c);
  else error()
}
```

Computing the Parse Table

Computing an LL(1) parse table

- Given: PREDICT set of each rule
- Table is a mapping M:

  $N \times T \rightarrow$ Production rule or error

- For all productions A $\rightarrow \alpha$ do
  - For each terminal t in PREDICT(A $\rightarrow \alpha$):

    M[A][t] = A $\rightarrow \alpha$

  - Every undefined table entry is an error

Example: Parse table

Non-term. \ Term.	a	b	c
S	1	2	-
B	-	3	-
C	-	-	4

Table-based, Predictive Parsing

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

Table-based, Predictive Parsing

Parse stack: Prediction of what will be seen in the future

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

Table-based, Predictive Parsing

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

Read one token after another, always looking only one token ahead

Table-based, Predictive Parsing

Check if expected terminal is indeed the next token

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

Table-based, Predictive Parsing

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

Apply a production rule: Push right-hand side onto stack

Table-based, Predictive Parsing

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
until x is EOF
```

No entry in table:

Raise error

Example: Table-based parsing

Input: bcc

Stack	Remaining input	Steps
EOF, S	b, c, c, EOF	Pop S Use rule S → bC Push C, b
EOF, C, b	b, c, c, EOF	Pop b Read next token
EOF, C	c, c, EOF	Pop C Use rule C → cc Push c, c
EOF, c, c	c, c, EOF	Pop c Read next token
EOF, c	c, EOF	Pop c Read next token
EOF	EOF	Pop EOF

→ done