CS 536 Announcements for Wednesday, March 6, 2024

Last Time
• wrap up CYK
• classes of grammars
• top-down parsing

Today
• review grammar transformations
• building a predictive parser
• FIRST and FOLLOW sets

Next Time
• predictive parsing and syntax-directed translation

LL(1) Predictive Parser

Predict the parse tree top-down

Parser structure
• 1 token lookahead
• parse-selector table
• stack tracking current parse tree's frontier

Necessary conditions
• left-factored
• free of left-recursion
Review of LL(1) grammar transformations

Necessary (but not sufficient conditions) for LL(1) parsing

- free of left recursion – no left-recursive rules
- left-factored – no rules with a common prefix, for any nonterminal

Why left-recursion is a problem

Outside/high-level view

CFG snippet: `xlist → xlist X | X`
Current parse tree: `xlist` Current token: `X`

Inside/algorithmic-level view

CFG snippet: `xlist → xlist X | X`
Current parse tree: `xlist` Current token: `X`
Removing left-recursion (review)

Replace

\[ A \to A \alpha \mid \beta \]

with

\[ A \to \beta A' \]
\[ A' \to \alpha A' \mid \epsilon \]

where \( \beta \) does not start with \( A \) (or may be \( \epsilon \))

Preserves the language (as a list of \( \alpha \)'s, starting with a \( \beta \)), but uses right recursion

**Example**

\[ \text{xlist} \to \text{xlist X} \mid \epsilon \]
Left factoring (review)

Removing a common prefix from a grammar

Replace

\[ A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \ldots \mid \alpha \beta_m \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m \]

with

\[ A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m \]

\[ A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \]

where \( \beta_i \) and \( \gamma_i \) are sequence of symbols with no common prefix

Note: \( \gamma_i \) may not be present, and one of the \( \beta_i \) may be \( \epsilon \)

Idea: combine all "problematic" rules that start with \( \alpha \) into one rule \( \alpha A' \)

\( A' \) now represents the suffix of the problematic rules

**Example 1**

\[ \text{exp} \rightarrow \langle A \rangle \mid \langle B \rangle \mid \langle C \rangle \mid D \]

**Example 2**

\[ \text{stmt} \rightarrow \text{ID ASSIGN exp} \mid \text{ID ( elist )} \mid \text{return} \]

\[ \text{exp} \rightarrow \text{INTLIT} \mid \text{ID} \]

\[ \text{elist} \rightarrow \text{exp} \mid \text{exp COMMA elist} \]
Building the parse table

Goal: given production \( lhs \rightarrow rhs \), determine what terminals would lead us to choose that production

- what terminals could \( rhs \) possibly start with?
- What terminals could possibly come after \( lhs \)?

Idea: \( \text{FIRST}(rhs) = \) set of terminals that begin sequences derivable from \( rhs \)

Suppose top-of-stack symbol is nonterminal \( p \) and the current token is \( A \) and we have
  - Production 1: \( p \rightarrow \alpha \)
  - Production 2: \( p \rightarrow \beta \)

\( \text{FIRST} \) lets us disambiguate:
  - if \( A \in \text{FIRST}(\alpha) \), then
  - if \( A \in \text{FIRST}(\beta) \), then
  - if \( A \) is in just one of them, then

**FIRST sets**

\( \text{FIRST}(\alpha) \) is the set of terminals that begin the strings derivable from \( \alpha \), and also, if \( \alpha \) can derive \( \varepsilon \), then \( \varepsilon \) is in \( \text{FIRST}(\alpha) \).

Formally,

\[
\text{FIRST}(\alpha) = \\
\text{For a symbol } X \\
\text{if } X \text{ is terminal: } \text{FIRST}(X) = \{X\} \\
\text{if } X \text{ is } \varepsilon : \text{FIRST}(X) = \{\varepsilon\} \\
\text{if } X \text{ is nonterminal : for each production } X \rightarrow Y_1Y_2Y_3..Y_n \\
\text{  put FIRST}(Y_1) - \varepsilon \text{ into FIRST}(X) \\
\text{  if } \varepsilon \text{ is in FIRST}(Y_1), \text{ put FIRST}(Y_2) - \varepsilon \text{ into FIRST}(X) \\
\text{  if } \varepsilon \text{ is in FIRST}(Y_2), \text{ put FIRST}(Y_3) - \varepsilon \text{ into FIRST}(X) \\
\text{  ...} \\
\text{  if } \varepsilon \text{ is in FIRST}(Y_i) \text{ for all } i, \text{ put } \varepsilon \text{ into FIRST}(X)
\]
Example

Original CFG

expr → expr + term
   | term

term → term * factor
   | factor

factor → exponent ^ factor
   | exponent

exponent → INTLIT
   | ( expr )

Transformed CFG

FIRST FOLLOW

expr

expr'

term

term'

factor

factor'

exponent

FIRST

expr → term expr'

expr' → + term expr'

expr' → ε

term → factor term'

term' → * factor term'

term' → ε

factor → exponent factor'

factor' → ^ factor

factor' → ε

exponent → INTLIT

exponent → ( expr )
Computing FIRST(α) (continued)

Extend FIRST to strings of symbols α

Let α = Y₁Y₂Y₃..Yₙ

- put FIRST(Y₁) – ε into FIRST(α)
  - if ε is in FIRST(Y₁), put FIRST(Y₂) – ε into FIRST(α)
  - if ε is in FIRST(Y₂), put FIRST(Y₃) – ε into FIRST(α)
  - ...
  - if ε is in FIRST(Yᵢ) for all i, put ε into FIRST(α)

Given two productions for nonterminal p

- Production 1: p → α
- Production 2: p → β

FOLLOW sets

For single nonterminal a, FOLLOW(a) is the set of terminals that can appear immediately to the right of a

Formally,

FOLLOW(a) =
Computing FOLLOW sets

To build FOLLOW(a)

• if a is the start non-term, put EOF in FOLLOW(a)

• for each production \( x \rightarrow \alpha a \beta \)
  • put FIRST(\( \beta \)) – \( \epsilon \) into FOLLOW(a)
  • if \( \epsilon \) is in FIRST(\( \beta \)), put FOLLOW(x) into FOLLOW(a)

• for each production \( x \rightarrow \alpha a \)
  • put FOLLOW(x) into FOLLOW(a)

Building the parse table

for each production \( x \rightarrow \alpha \) {
    for each terminal T in FIRST(\( \alpha \)) {
        put \( \alpha \) in table[\( x \)][T]
    }
    if \( \epsilon \) is in FIRST(\( \alpha \)) {
        for each terminal T in FOLLOW(x) {
            put \( \alpha \) in table[\( x \)][T]
        }
    }
}