Section 4: CUP & LL

Jack Eggleston, Nate Yazdani & Aaron Johnston
CSE 401/M501 – Compilers
Autumn 2018
Administrivia

• Homework 2 is due tonight!
  – You have late days if you need them
• Parser is due one week from today
• Scanner feedback by next week
  – Be sure to check when debugging parser 😊
Agenda

• CUP tips, tricks, and demo
• LL parsing
  – See Sec. 3.3 of Cooper & Torczon for more
• A worksheet all about LL
The CUP parser generator

- Uses LALR(1)
  - Weaker but faster variant of LR(1)
- LALR is more sensitive to ambiguity than LR
Language Hierarchies

Diagram showing language hierarchies with unambiguous grammars on the left and ambiguous grammars on the right. The hierarchy includes LL(k), LR(k), LL(1), LR(1), LALR(1), SLR, LR(0), and LL(0) grammars.
The CUP parser generator

• Uses LALR(1)
  – Weaker but faster variant of LR(1)

• LALR is more sensitive to ambiguity than LR

• CUP can resolve some ambiguities itself
  – Precedence for reduce/reduce conflicts
  – Associativity for shift/reduce conflicts

• If you use those features, read the docs carefully
The CUP parser generator

*Demo*: testing and debugging a CUP parser
LL(k) parsing

• LL(k) scans left-to-right, builds leftmost derivation, and looks ahead $k$ symbols
• Typically $k = 1$, just like LR

• The LL condition enable the parser to choose productions correctly with 1 symbol of look-ahead
• We can transform a grammar to satisfy them
LL Condition

For each nonterminal in the grammar:

- Its *productions* must have disjoint FIRST sets

\[
\begin{align*}
A &::= x \mid B \\
B &::= x
\end{align*}
\]  
\[
\begin{align*}
A &::= x \mid B \\
B &::= y
\end{align*}
\]

- If it is *nullable*, the FIRST sets of its productions must be disjoint from its FOLLOW set

\[
\begin{align*}
S &::= A \ x \\
A &::= \epsilon \mid x \\
S &::= A \ y \\
A &::= \epsilon \mid x
\end{align*}
\]
Factoring out common prefixes

When multiple productions of a nonterminal share a common prefix, turn the different suffixes (“trails”) into a new nonterminal.

\[
Greeting ::= "hello, world" \mid "hello, friend" \mid "hello, " Name
\]
\[
Name ::= "Sarah" \mid "John" \mid ...
\]

\[
Greeting ::= "hello, " Address
\]
\[
Address ::= "world" \mid "friend" \mid Name
\]
\[
Name ::= "Sarah" \mid "John" \mid ...
\]
Removing direct left recursion

When a nonterminal has left-recursive productions, turn the different suffixes ("trails") into a new nonterminal, appended to the remaining productions.

```
Sum ::= Sum "+" Sum | Sum "-" Sum | Constant
Constant ::= "1" | "2" | "3" | ...

Sum ::= Constant SumTrail
SumTrail ::= "+" Sum | "-" Sum | $\varepsilon$
Constant ::= "1" | "2" | "3" | ...
```
Removing indirect left recursion

- Pseudocode from Cooper & Torczon:

  ```plaintext
  impose an order on the nonterminals, A_1, A_2, ..., A_n
  for i ← 1 to n do:
    for j ← 1 to i - 1 do:
      if ∃ a production A_i → A_jγ
          then replace A_i → A_jγ with one or more
              productions that expand A_j
    end;
  end;
  rewrite the productions to eliminate
  any direct left recursion on A_i
  end;
  ```

- FIGURE 3.6 Removal of Indirect Left Recursion.

- Rather conservative: no need to push A_j into A_i if you know that A_j \not\Rightarrow \alpha A_i \beta for any \alpha, \beta
Removing indirect left recursion

When a nonterminal has another nonterminal (B) on the left of a production, rewrite that production to use all possible expansions of B. Repeat until the left side of every production is a terminal or direct left recursion. (Must choose an order to process nonterminals)

\[
Expr ::= \text{Ternary} \mid \text{Addition}
\]
\[
\text{Ternary} ::= Expr \text{"?"} Expr \text{":"} Stmt
\]
\[
\text{Addition} ::= Expr \text{"+"} Expr
\]
\[
Expr ::= Expr \text{"?"} Expr \text{":"} Stmt \mid Expr \text{"+"} Expr
\]
Worksheet

• Discuss and work in small groups!

• Reminders:
  – FIRST(\(\alpha\)) is the set of terminal symbols that can begin a string derived from \(\alpha\)
  – FOLLOW(A) is the set of terminal symbols that may immediately follow A in a derived string
  – nullable(A) is whether A can derive \(\varepsilon\)
Computing FIRST, FOLLOW, and nullable

repeat
    for each production $X := Y_1 Y_2 \ldots Y_k$
        if $Y_1 \ldots Y_k$ are all nullable (or if $k = 0$)
            set nullable[$X$] = true
        for each $i$ from 1 to $k$ and each $j$ from $i+1$ to $k$
            if $Y_1 \ldots Y_{i-1}$ are all nullable (or if $i = 1$)
                add FIRST[$Y_i$] to FIRST[$X$]
            if $Y_{i+1} \ldots Y_k$ are all nullable (or if $i = k$)
                add FOLLOW[$X$] to FOLLOW[$Y_i$]
            if $Y_{i+1} \ldots Y_{j-1}$ are all nullable (or if $i+1=j$)
                add FIRST[$Y_j$] to FOLLOW[$Y_i$]
    Until FIRST, FOLLOW, and nullable do not change