CSE 401/M501 – Compilers

Section 4: CUP and LR parsing
Nate Yazdani & Aaron Johnston
Spring 2018
Administrivia

• Homework 2 is due tonight!
  – You have late days if you need them
• Parser is due one week from today
• Scanner feedback before the weekend
  – 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 fun worksheet!
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
• LL(1) requires for every nonterminal $A$...
  – $\bigcap_{\alpha \in \text{RHS}(A)} \text{FIRST}(\alpha) = \emptyset$
  – nullable($A$) $\Rightarrow$ FIRST($A$) $\cap$ FOLLOW($A$) $= \emptyset$
• Those restrictions enable the parser to choose productions correctly with 1 symbol of look-ahead
• We can transform a grammar to satisfy them
Factoring out common prefixes

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

\[
\text{Greeting} ::= \text{"hello, world"} \mid \text{"hello, friend"} \mid \text{"hello, \ Name}\]
\[
\text{Name} ::= \text{"Sarah"} \mid \text{"John"} \mid \ldots
\]

\[
\text{Greeting} ::= \text{"hello," Name}
\text{Address} ::= \text{"world"} \mid \text{"friend"} \mid \text{Name}
\text{Name} ::= \text{"Sarah"} \mid \text{"John"} \mid \ldots
\]
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.

\[
Sum ::= Sum \text{"+"} \text{Sum} \mid Sum \text{"-"} \text{Sum} \mid \text{Constant}
\]

\[
Constant ::= \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \ldots
\]

\[
Sum ::= \text{Constant} \text{SumTrail}
\]

\[
SumTrail ::= \text{"+"} \text{Sum} \mid \text{"-"} \text{Sum} \mid \varepsilon
\]

\[
Constant ::= \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \ldots
\]
Removing indirect left recursion

- Pseudocode from Cooper & Torczon:

\[
\text{impose an order on the nonterminals, } A_1, A_2, \ldots, A_n \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do; } \\
\quad \text{for } j \leftarrow 1 \text{ to } i - 1 \text{ do; } \\
\qquad \text{if } \exists \text{ a production } A_i \rightarrow A_j \gamma \\
\qquad \quad \text{then replace } A_i \rightarrow A_j \gamma \text{ with one or more } \\
\qquad \quad \text{productions that expand } A_j \\
\qquad \text{end; } \\
\text{rewrite the productions to eliminate } \\
\text{any direct left recursion on } A_i \\
\text{end; }
\]

- 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 \( \alpha, \beta \)
Worksheet time

• Feel free to discuss and work in small groups

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