Compilers

Foundations of Formal Languages

Regular Language: A language that can be expressed using regular expressions or recognized by finite automata (DFA or NFA). Regular languages are closed under union, concatenation, and Kleene star operations. Regular languages are less expressive than context-free languages. **Regular Expression**: A pattern describing a regular language, using symbols like '*' (zero or more), '+' (one or more), '|' (union), and '()' (grouping). **Context-Free Language (CFL)**: Languages defined by context-free grammars (CFGs) and recognized by pushdown automata (PDAs). **Context-Free Grammar (CFG)**: A grammar consisting of rules of the form $A \to \alpha$, where A is a nonterminal, and α is a string of terminals and/or nonterminals. CFGs generate context-free languages, which can be recognized by pushdown automata. **Strong Grammar**: A grammar is strongly LL(k) if, for every nonterminal A and every lookahead string of k tokens w, there is at most one production rule $A \to \alpha$ that can be applied, determined solely by the first k tokens of the input (the lookahead). This eliminates any need for backtracking or additional context beyond the k-token lookahead.

First and Follow Sets

The $\mathbf{FIRST^k}(\alpha)$ set for a string α (terminal, non-terminal, or sequence) contains all sequences of up to k terminals that can appear as the first k symbols in any derivation of α. If α derives a string shorter than k, the entire derived string is included.

Steps to Compute $FIRST^k(\alpha)$:

1. If α is a terminal a: Add the string a (of length 1) to $FIRST^k(\alpha)$. 2. If α is a sequence $\alpha_1\alpha_2\dots\alpha_n$: - Initialize $FIRST^k(\alpha) = \emptyset$. - For $i = 1$ to n: - Add to $FIRST^k(\alpha)$ all strings of length up to k from $FIRST^k(\alpha_i)$ if $\alpha_1,\dots,\alpha_{i-1}$ can all derive the empty string ϵ. - For strings shorter than k, concatenate with $FIRST^{k-|s|}(\alpha_{i+1}\dots\alpha_n)$ (where $|s|$ is the length of the string). - If α can derive ϵ, include ϵ in $FIRST^k(\alpha)$. 3. If α is a non-terminal A: - For each production $A \to \beta$, compute $FIRST^k(\beta)$ and add its strings to $FIRST^k(A)$. - Repeat until $FIRST^k(A)$ stabilizes (no new strings are added). 4. Handle k-length lookahead: Truncate strings longer than k to their first k symbols.

The $\mathbf{FOLLOW^k(A)}$ set for a non-terminal A contains all sequences of up to k terminals that can appear immediately after A in some derivation from the start symbol. If A appears at the end of a derivation, include the end-of-input marker (e.g., \$).

Steps to Compute $FOLLOW^k(A)$:

1. Initialize: - For the start symbol S, add \$ (or a k-length end marker) to $FOLLOW^k(S)$. - Set $FOLLOW^k(A) = \emptyset$ for all other non-terminals A. 2. For each production $B \to \alpha A\beta$ (where A is a non-terminal): - Compute $FIRST^k(\beta)$ and add its strings to $FOLLOW^k(A)$. - If β can derive ϵ, add $FOLLOW^k(B)$ to $FOLLOW^k(A)$. 3. Iterate: - Repeat step 2 across all productions until no new strings are added to any $FOLLOW^k(A)$. 4. Handle k-length lookahead: Ensure all strings in $FOLLOW^k(A)$ are of length up to k, truncating longer strings to their first k symbols.

LR Items

An $\mathbf{LR(k)}$ item is a production with a dot ($\bullet$) marking a position in the right-hand side, along with a k-symbol lookahead string. The dot indicates how much of the production has been recognized so far. An item $[A \to \alpha \bullet \beta, w]$ means we have seen α and expect to see β, with lookahead w.

Types of LR Items: **Shift item**: $[A \to \alpha \bullet a\beta, w]$ where a is a terminal **Reduce item**: $[A \to \alpha\bullet, w]$ where the dot is at the end **Goto item**: $[A \to \alpha \bullet B\beta, w]$ where B is a non-terminal

Steps to Compute $\mathbf{CLOSURE(I)}$ for a set of items I: 1. Initialize $CLOSURE(I) = I$. 2. For each item $[A \to \alpha \bullet B\beta, w]$ in $CLOSURE(I)$ where B is a non-terminal: - For each production $B \to \gamma$: - Compute $FIRST^k(\beta w)$ (concatenate β and lookahead w, then take first k symbols). - For each string $u \in FIRST^k(\beta w)$: - Add item $[B \to \bullet\gamma, u]$ to $CLOSURE(I)$ if not already present. 3. Repeat step 2 until no new items are added to $CLOSURE(I)$.

Steps to Compute $\mathbf{GOTO(I, X)}$ for item set I and symbol X: 1. Initialize $J = \emptyset$. 2. For each item $[A \to \alpha \bullet X\beta, w]$ in I: - Add item $[A \to \alpha X \bullet \beta, w]$ to J. 3. Return $CLOSURE(J)$.

Steps to Construct the $\mathbf{LR(k)}$ Automaton: 1. Create the initial state $I_0 = CLOSURE(\{[S' \to \bullet S, \$^k]\})$ where S' is the augmented start symbol. 2. Initialize the set of states $\mathcal{C} = \{I_0\}$ and a worklist $W = \{I_0\}$. 3. While $W \neq \emptyset$: - Remove a state I from W. - For each symbol X (terminal or non-terminal) such that $GOTO(I, X) \neq \emptyset$: - Let $J = GOTO(I, X)$. - If $J \notin \mathcal{C}$: - Add J to $\mathcal{C}$ and W. - Add transition $I \xrightarrow{X} J$ to the automaton. 4. The resulting automaton defines the LR parsing table.

Parsing Conflicts

Shift-Reduce Conflict A shift-reduce conflict occurs when the parser cannot decide whether to: 1) **Shift**: Move the next input symbol onto the stack 2) **Reduce**: Apply a production rule to reduce symbols on the stack

In terms of LR items, this happens when a state contains both: 1) A shift item: $[A \to \alpha \bullet a\beta, w]$ (expecting terminal a) 2) A reduce item: $[B \to \gamma\bullet, a]$ (ready to reduce with lookahead a)

The conflict manifests in the parsing table as both a shift action and a reduce action for entry $[state, a]$. **Reduce-Reduce Conflict** A reduce-reduce conflict occurs when the parser cannot decide which production to use for reduction. This happens when a state contains multiple reduce items with overlapping lookaheads: 1) $[A \to \alpha\bullet, w]$ 2) $[B \to \beta\bullet, w]$

Both items indicate that reduction is possible with the same lookahead w, but the parser cannot determine which production rule to apply.

The conflict manifests in the parsing table as multiple reduce actions for the same entry $[state, w]$.

Resolution These conflicts indicate that the grammar is not in the respective LR class (LR(k), LALR(k), or SLR(k)). Resolution strategies include: 1) Increasing the lookahead length k 2) Grammar transformation (left-factoring, eliminating ambiguity) 3) Using precedence and associativity rules