Compilers

Calculate First and Follow

First $^k(\alpha)$

1. Initialization: For every terminal a, set $First^k(a) = a$ (since a terminal generates only itself). Initialize $First^k(A)$ to an empty set for every variable A.
2. Iterative Update: Repeat until no changes occur: For each variable A with a production rule $A ß X_1 X_2 ... X_n$: Compute $First^k(X_1)$, $First^k(X_2)$, ..., $First^k(X_n)$. Concatenate elements from $First^k(X_1)$, $First^k(X_2)$, ..., $First^k(X_n)$ and truncate the results to length k. Update $First^k(A)$ with these truncated elements if they are not already present.
3. Completion: The process stops when no $First^k$ sets are updated in an iteration, meaning all sets have stabilized.

Follow $^k(\alpha)$

1. Initialization: Initialize $Follow^k(X)$ to an empty set for each variable X.
2. Iterative Update: Repeat until no changes occur: For each rule $A ß B$: Compute $First^k()$ (where is the part of the right-hand side following B). Add $First^k()$ elements to $Follow^k(B)$, completing them with $Follow^k(A)$ if needed to ensure they reach length k. Update $Follow^k(B)$ with these completed elements if they are not already present.
3. Completion: The process stops when no $Follow^k$ sets are updated in an iteration, meaning all sets have stabilized.

Grammars

LL(k)

1. Identify Production Rules for Each Non-Terminal
1a. Initialization: For each non-terminal A, list all production rules of the form $A \to \alpha_1$ and $A \to \alpha_2$, where α_1 and α_2 are sequences of symbols (terminals and/or non-terminals). 1b. Compare Productions: For each pair of distinct production rules $A \to \alpha_1$ and $A \to \alpha_2$: Note that the possible right-hand sides are α_1 and α_2.
2. Analyze Derivations with Lookahead
2a. Derive Sentential Forms: For each pair $(A \to \alpha_1, A \to \alpha_2)$, consider the derivations of the form: $S \Rightarrow^* wA\gamma \Rightarrow w\alpha_1\gamma \Rightarrow^* wx_1$ $S \Rightarrow^* wA\gamma \Rightarrow w\alpha_2\gamma \Rightarrow^* wx_2$ Here, w is a prefix of terminals, and γ is a sequence of symbols. 2b. Compute Lookahead Sets: Determine the lookahead sets $\text{First}_k(x_1)$ and $\text{First}_k(x_2)$ where x_1 and x_2 are terminal strings derived from $\alpha_1\gamma$ and $\alpha_2\gamma$, respectively.
3. Check for Disambiguation
3a. Lookahead Comparison: For each pair $(A \to \alpha_1, A \to \alpha_2)$, compare the lookahead sets: Check if $\text{First}_k(x_1)$ equals $\text{First}_k(x_2)$.
3b. Verify Consistency: If $\text{First}_k(x_1) = \text{First}_k(x_2)$, ensure that α_1 and α_2 are identical: If $\alpha_1 \neq \alpha_2$ when the lookahead sets are equal, the grammar is not LL(k).
4. Conclude LL(k) Status
If all pairs of distinct production rules for every non-terminal either have different lookahead sets or identical right-hand sides, the grammar is LL(k). If any pair of rules has the same lookahead sets but different right-hand sides, the grammar is not LL(k).

Strong LL(k)

A $CFGG = \langle V, T, P, S \rangle$ is strong LL(k) iff, for all pairs of rules $A ß \alpha_1$ and $A ß \alpha_2$ in P(with $\alpha_1 = \alpha_2$):
$First^k(\alpha_1 Follow^k(A)) \cap First^k(\alpha_2 Followk(A)) = \varnothing$

LR(k)

Time Complexities

DFA: $\mathcal{O}(|w|)$
CFG: polynomial in $|w|$
Grammar (not necessarily context free): undecidable problem

Undecidability of Type Checking and Reachability Analysis

The exact versions of both type checking and reachability analysis are undecidable because they would require solving the halting problem, which is proven to be undecidable. For type checking, determining the exact type of an expression in all possible cases would involve analyzing all potential program executions, which is equivalent to solving whether a program halts with certain inputs. Similarly, exact reachability analysis involves determining whether a specific program state can be reached, which also boils down to predicting the behavior of all possible executions, again leading to the halting problem. Since the halting problem is undecidable, both exact type checking and exact reachability analysis are also undecidable.

Trivia

Q:Give an advantage of hand-built scanners over automata-based scanners? A:They can go beyond regular languages
Q: How can we make sure the sequence 'true' is always deamed a contant and not a string? A: Constants like 'true' should be recognized as a single token

Example Languages

LR(0) language that is not LL(1)
$L = \{a^n b^n | n \geq 0\}$
LR(0) language that is not LL(k) for any k
$L = \{a^n b^n c^m | n, m \geq 1\}$
LL(2) language that is not LL(1)
$L = \{a^n bc^n d \mid n \geq 1\} \cup \{a^n cb^n d \mid n \geq 1\}$
CFL that in not LR(1)
English language

Example Grammars

LL(1) grammar that is not strongly LL(1)
$S \to A|B$
$A \to aA|a$
$B \to bB|b$
LL(2) grammar that is not strongly LL(2)
$S \to AB$
$A \to aAa|a$
$B \to bBb|b$
LALR(1) grammar that is not SLR(1)
$S \to Aa$
$A \to Bb|Cc$
$B \to b$
$C \to c$
CFG that in not LR(1)
$S \to aB|aDc$
$B \to bBc|c$
$D \to bc|c$