Compilers

First and Follow

First

The $FIRST$ set of a non-terminal symbol A in a grammar is the set of terminal symbols that appear at the beginning of any string derived from A.
How to Compute: For a non-terminal symbol A: If A can derive a string that starts with a terminal symbol, include that terminal in $FIRST(A)$. If A can derive ϵ (the empty string), include ϵ in $FIRST(A)$. For a production $A \to B_1 B_2 \dots B_n$: Include the $FIRST$ set of B_1 in $FIRST(A)$. If B_1 can derive ϵ, then include $FIRST(B_2)$ in $FIRST(A)$, and so on for subsequent B_i until a non-ϵ deriving symbol is found or all B_i derive ϵ.

Follow

The FOLLOW set of a non-terminal symbol A is the set of terminal symbols that can appear immediately to the right of A in some sentential form derived from the start symbol of the grammar.
How to Compute: 1. Initialization: Add \$ (the end-of-input marker) to FOLLOW set of the start symbol. 2. For Each Production $A \to \alpha B \beta$: Add $FIRST(\beta)$ (excluding ϵ) to $FOLLOW(B)$. If β can derive ϵ (or if β is empty), add $FOLLOW(A)$ to $FOLLOW(B)$. 3. Repeat Until No Changes: Continue updating the FOLLOW sets until no more symbols can be added.

ItemFollows

The $FIRST$ set of a string of symbols is the set of terminals that can appear as the first symbol of a string derived from the given string. The $FOLLOW$ set of a string of symbols is the set of terminals that can appear immediately to the right of the string in some sentential form derived from the start symbol of the grammar.

Grammars

LL(k)

1. No Left Recursion: No non-terminal should be able to derive itself through a sequence of productions that starts with the same non-terminal. For example, a rule like $A \to A\alpha$ (where α is any sequence of terminals and/or non-terminals) would be left-recursive.
2. Left Factoring: For any non-terminal A and productions $A \to \alpha\beta_1$ and $A \to \alpha\beta_2$ where α is a common prefix, the grammar should be refactored to remove the common prefix. The left-factored form would be: $A \to \alpha A'$
$A' \to \beta_1 \mid \beta_2$
3. First/Follow Set Conditions: For each non-terminal A and productions $A \to \alpha_1$ and $A \to \alpha_2$: The sets of terminals that can appear as the first token of the strings derived from α_1 and α_2 (i.e., the FIRST sets) must be disjoint. If α_i can derive the empty string ϵ, then the FIRST set of α_i should not intersect with the FOLLOW set of A. The FOLLOW set of A is the set of terminals that can appear immediately to the right of A in some sentential form.

Languages

Regular langauges

Regular languages are those that can be recognized by a finite automaton or expressed by a regular expression.

Context-free languages

Context-free languages are those that can be recognized by a pushdown automaton or expressed by a context-free grammar.

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:Which one of the following is 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)
$L = \{a^n b c^n d \mid n \geq 1\} \cup \{a^n c b^n d \mid n \geq 1\}$
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)
$S \to aB|aDc$
$B \to bBc|c$
$D \to bc|c$