Compilers

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

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

Example Languages

LR(0) language that is not LL(1)
$S \to aSb|a$
LR(0) language that is not LL(k)
$S \to aSb|ab$
LL(2) language that is not LL(1)
$S \to aAB|bBA$
$A \to c$
$B \to d$
LL(3) language that is not LL(2)
$S \to aaa|aab|aac$
CFG that in not LR(1)
$S \to aB|aDc$
$B \to bBc|c$
$D \to bc|c$

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)
$S \to Aa$
$A \to Bb|Cc$
$B \to b$
$C \to c$