dOvs Eksamens Noter

Hugh Benjamin Zachariae

January 2020

Contents

1 Compiler intro

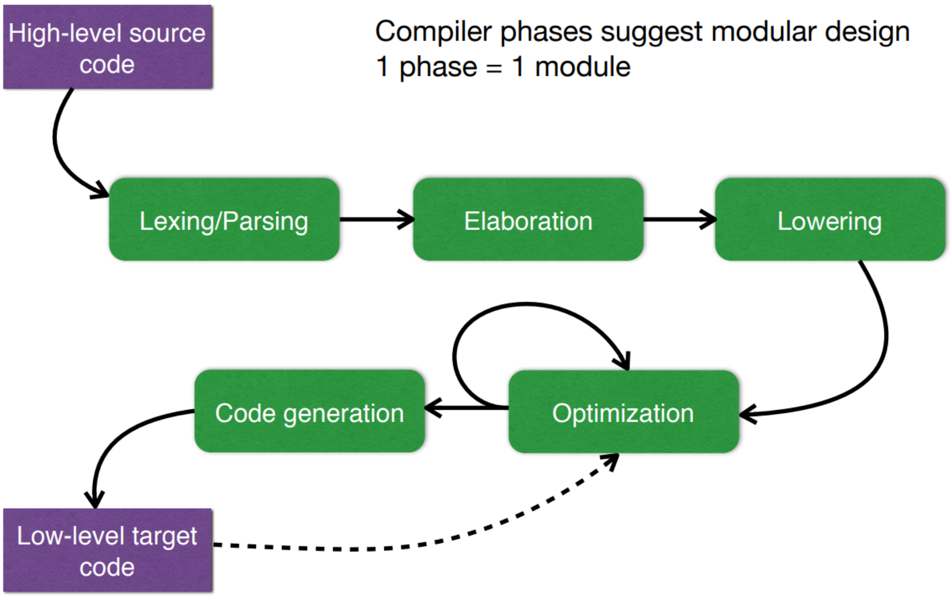

Figure 1: Compiler modular phases.

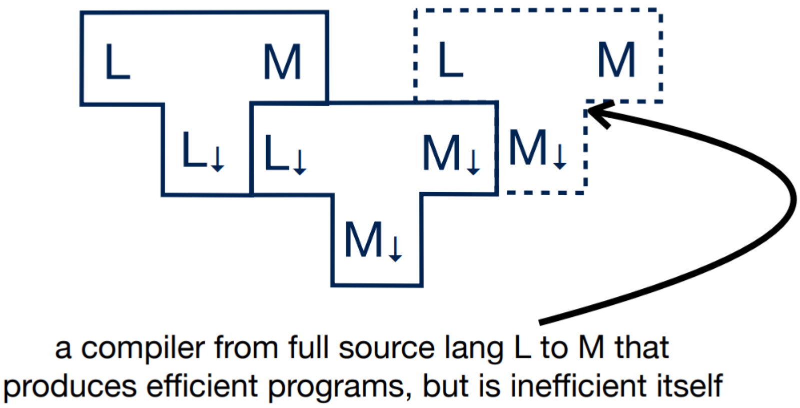

Figure 2: Bootstrap compiling

- **Lexing/Parsing**: String $\rightarrow_{lexing}$ Tokens $\rightarrow_{parsing}$ Abstract Syntax Tree (AST)
- **Elaboration**: *Resolving scope* and *Type checking*. Most errors found here.
- **Lowering**: High-level features to target-language like constructs (e.g. assembly-like). *Intermediate representation*, LLVM.
- **Optimization**: Detect and rewrite expensive operations. Lifting invariants out of loops, parallelization.
- **Code generation**: fx LLVM to X86 (registers, instruction etc.)

- **Bootstrapping compilers**: Compile your language in your own language.

2 Lexical

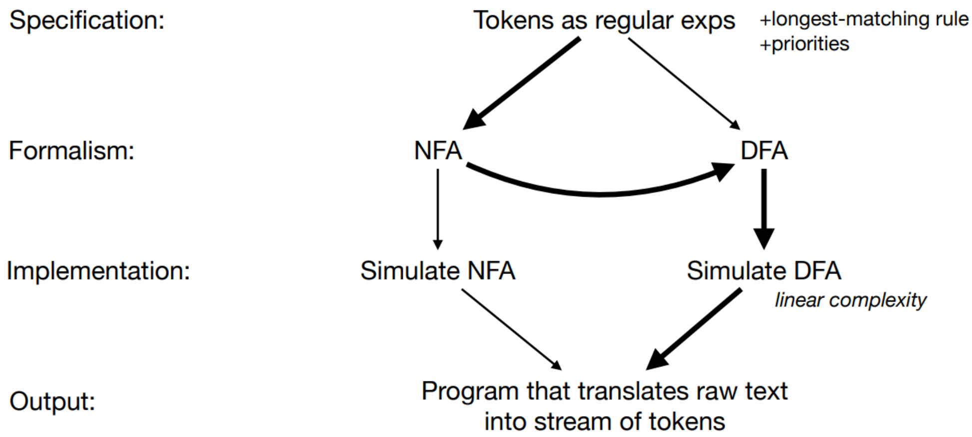

Figure 3: REG to NFA to DFA

- Tokens: E.g. ID("a"), INT, IF etc. Some tokens include metadata like names in ID.
- Non-tokens: comments, whitespace etc.
- REG $\rightarrow$ NFA $\rightarrow$ (closures) DFA $\rightarrow$ Minimized DFA (more effective)
- REG: Handle priorities and longest matching string token wins.
- Ocamllex: Lexer generator

3 Parsing

A context-free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$

- V is a finite set of *nonterminal* symbols
- Σ is an alphabet of *terminal* symbols and $V \cap \Sigma = \varnothing$
- $S \in V$ is a *start* symbol
- P is a finite set of *productions* of the form $A \rightarrow \alpha$, where
  - $A \in V$, i.e., A is a nonterminal, and
  - $\alpha \in (V \cup \Sigma)^*$, i.e., α is possibly empty string of nonterminals or terminals

Figure 4: CFG Definition

S → if E then S else S
S → begin S L
S → print E
L → end
L → ; S L
E → num = num

- FIRST (α) : set of terminals that begin strings derived from α
- FOLLOW(X): set of terminals a that can appear immediately to the right of X in some derivable string, e.g., S ⇒* αXaβ
- Let nullable(X) be true when X can derive empty string ε

Nonterminal	Nullable?	First set	Follow set
S		if, begin, print	else, end, ;, $
L		end, ;	else, end, ;, $
E		num	then, else, end, ; $

Figure 5: Top-down parsing table. You do not want more than one possibility in a cell.

- Abstract Syntax Tree (AST):
- Context-Free Grammars (CFG):
  - *Terminals* → *production rules*
  - Terminals are leafs in the tree (e.g. x, y).
  - Non-Terminals are links in the tree (e.g. BinExp)
  - Definition see figure 4.
  - Ambiguity: You don't want ambiguity, you want determinism. *Associativity* (right/left) and *precedence* (e.g. times before plus).
- Top-down/Bottom-up parsing:
  - Top-down is predictive parsing:
    * leftmost derivation
    * "see whats coming"
    * Breaks down at for example: $S \to S + x \mid S - x \mid x$. Here you don't know what to do when you see an $x \ldots$
    * See figure 5 for parsing table.
  - Bottom-up: **LR parsing** is rightmost reduction.