
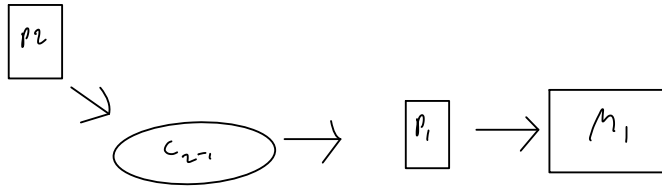


Ch 11 

Interpretation - run time

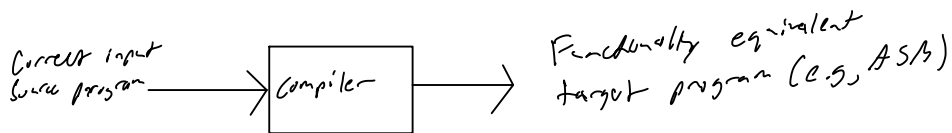
- model of interpretation
 - ↳ start with some host-to-program machine, M_1
 - ↳ write a single program for M_1 that mimics the behavior of some easier machine, M_2
 - ↳ results in a "virtual" M_2
- layers of abstraction
 - ↳ several layers of interpretation are used
- useful for one-off tasks

Compilation - compile time



- * Both interpretation and compilation:
- ↳ allow changes in source program
 - ↳ afford programming apps in platform ind. systems
 - ↳ widely used in computing

Basic Compiler Structure



Compilation Strategies

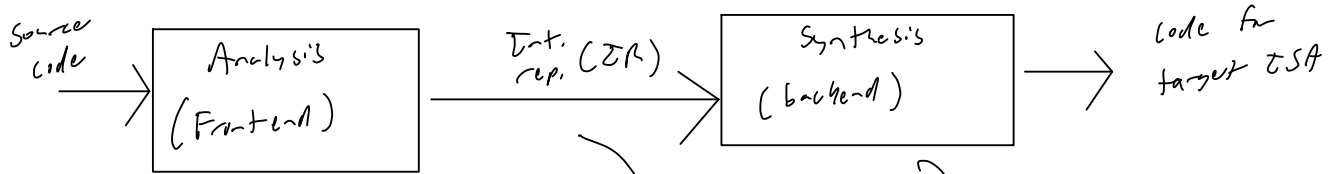
— compile_statement(statement)

- ↳ unconditional: $\text{expr};$
- ↳ compound: $\{ \text{statement}; \text{statement}; \dots \}$
- ↳ conditional: $\text{if } (\text{expr}) \text{ statement}; \text{ else statement};$
- ↳ iteration: $\text{while } (\text{expr}) \text{ statement};$

— compile_expr(expr)

- ↳ constants: 1234
- ↳ variables: $a, b[\text{expr}]$
- ↳ Assignment: $a = \text{expr}$
- ↳ Operations: $\text{expr}_1 + \text{expr}_2$
- ↳ Procedure calls: $\text{proc}(\text{expr}, \dots)$

Anatomy of a Modern Compiler



Frontend Stages

- ① Lexical analysis (scanning)
↳ Source → tokens
- ② Syntactic analysis (parsing)
↳ Tokens → syntax tree
- ③ Semantic analysis (mostly type checking)

Code generation

- register allocation
- translate each assignment & instructions
- exit each block: label, ass, branch
- lay out basic blocks, remove useless jumps
- ISA and CPU-specific optimizations

IR

- ↳ internal compile language that is:
 - ↳ language-ind.
 - ↳ machine-ind.
 - ↳ easy to optimize
- ↳ useful for optimizing source code
 - ↳ performs a set of passes over CFG (control flow graph)

Summary: Modern Compilers

