classmate
Date 15.8.2019
Page 1

OVERVIEW OF COMPILATION

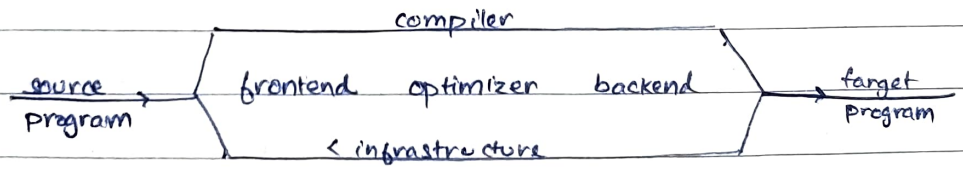
PROGRAMMING LANGUAGE

dfn the programming language is a formal language with mathematical properties and well-defined meanings, as opposed to a natural language with evolved properties and ambiguities.

Specify computations

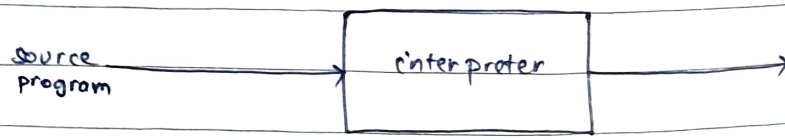
expressiveness
conciseness
clarity

dfn compiler:
a specialized program which translates a source program into a set of operations that are defined on the target computer.



dfn compilers that programming languages rather than the instruction set of a computer are often called source-to-source translators.

dfn interpreter:
an interpreter takes as an input an executable specification and produces as output the result of executing the specification



a good compiler contains a microcosm of computer science.
it makes practical applications of:
- greedy algorithms (register allocation)
- heuristic search techniques (list scheduling)
- graph algorithms (dead code elimination),
- dynamic programming (instruction selection),
- finite automata & push down automata (scanning & parsing),
- fixed-point algorithms (data-flow analysis).

it deals with problems, such as:
- dynamic allocation
- synchronization
- naming
- locality
- memory hierarchy management
- pipeline scheduling

formal language theory has led to tools that automate the production of scanners and parsers.

ad hoc methods - approximate - efficiency.

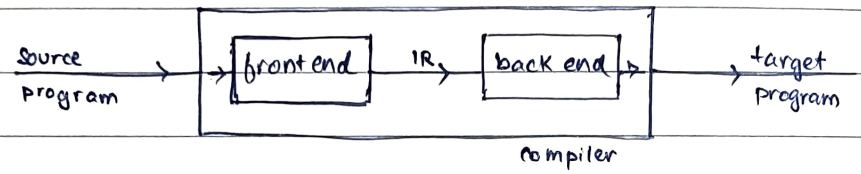
THE FUNDAMENTAL PRINCIPLES OF COMPILATION

the first principle that a compiler must observe is inviolable:

★ the compiler must preserve the meaning of the program being compiled. (social contract)

★ the compiler must improve the input program in some discernable way.

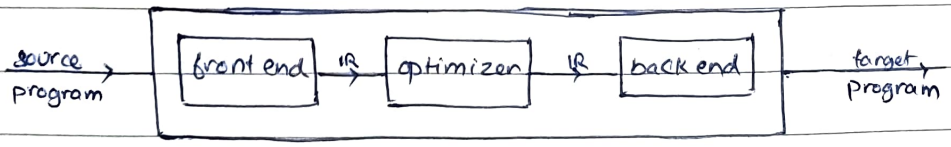
COMPILER STRUCTURE



two-phase compiler

the compiler can make multiple passes over the IR form of the code before emitting the target program.

two-phase structure : may simplify the process of retargeting. introducing IR makes it possible to add more phases to compilation.



three-phase compiler

optimizer:

- faster target prog.
- smaller target prog.
- fewer page faults
- less power.

front end:	optimizer:	backend:
- scan	- opt 1	- select
- parse	- opt 2	- schedule
- CSA	- opt n	- allocate
	~ analysis	

infrastructure:

- symbol tables
- trees
- graphs
- sets
- grammars
- ...

HIGH LEVEL VIEW OF TRANSLATION

form / syntax — meaning / semantics

defn the source language is a set, usually infinite, of strings defined by some finite set of rules, called a grammar.

sentence -> subject verb object endmark

dfn. the process of discovering words in a string of characters and classifying them according to their parts of speech is called scanning.

dfn. discovering whether a stream of classified words has a derivation in some set of grammatical rules is called parsing.

semantic analysis / context sensitive analysis.

a compiler implements the abstractions defined by the source language.

e.g. symbolic names.
procedures
parameters
lexical scopes
control-flow ops.

dfn the process of analyzing code to discover facts from context and using that knowledge to improve the code is often called code optimization.

dfn data flow analysis involves reasoning, at compile-time, about the flow of values at runtime.
system of simultaneous eqns.

dfn dependence analysis uses number theoretic tests to reason about the values that can be assumed by subscript expressions
used to disambiguate reference to array elements

loop-invariant computation,

varp → activation record pointer.

the problem of allocating an arbitrary set of values to a bounded set of machine registers in a way that minimizes loads and stores is NP-complete.

instruction scheduling is a hard-problem (NP-complete).

desirable properties of a compiler:
- speed
- space
- feedback (correct every syntax error)
- debugging
- compile-time efficiency

original FORTRAN : multipass system
* scanner parser register allocator optimizations

challenges for compilers:
- multiple functional units - long memory latencies
- parallel code generation.