



Compilers

Optimization Overview

- Optimization is our last compiler phase
- Most complexity in modern compilers is in the optimizer
 - Also by far the largest phase

- When should we perform optimizations?
 - On AST
 - **Pro**: Machine independent
 - **Con**: Too high level
 - On assembly language
 - **Pro**: Exposes optimization opportunities
 - **Con**: Machine dependent
 - **Con**: Must reimplement optimizations when retargeting
 - On an intermediate language
 - **Pro**: Machine independent
 - **Pro**: Exposes optimization opportunities

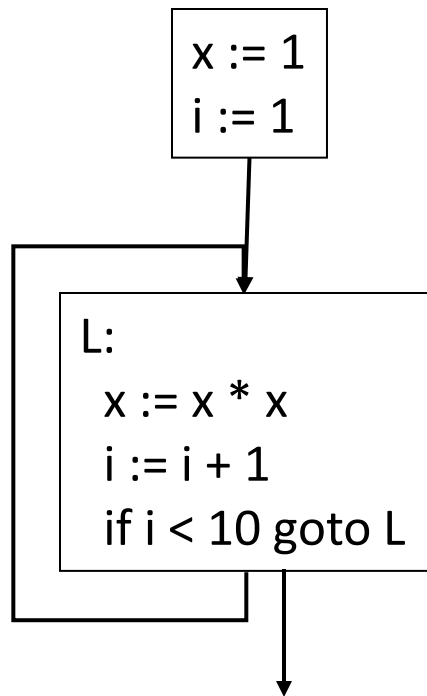
$P \rightarrow S P \mid S$
 $S \rightarrow \text{id} := \text{id op id}$
 $\mid \text{id} := \text{op id}$
 $\mid \text{id} := \text{id}$
 $\mid \text{push id}$
 $\mid \text{id} := \text{pop}$
 $\mid \text{if id relop id goto L}$
 $\mid \text{L:}$
 $\mid \text{jump L}$

- Id's are register names
- Constants can replace id's
- Typical operators: +, -, *

- A basic block is a maximal sequence of instructions with:
 - no labels (except at the first instruction), and
 - no jumps (except in the last instruction)
- Idea:
 - Cannot jump into a basic block (except at beginning)
 - Cannot jump out of a basic block (except at end)
 - A basic block is a single-entry, single-exit, straight-line code segment

- Consider the basic block
 1. L:
 2. $t := 2 * x$
 3. $w := t + x$
 4. if $w > 0$ goto L'
- (3) executes only after (2)
 - We can change (3) to $w := 3 * x$
 - Can we eliminate (2) as well?

- A control-flow graph is a directed graph with
 - Basic blocks as nodes
 - An edge from block A to block B if the execution can pass from the last instruction in A to the first instruction in B
 - E.g., the last instruction in A is **jump L_B**
 - E.g., execution can fall-through from block A to block B



- The body of a method (or procedure) can be represented as a control-flow graph
- There is one initial node
- All “return” nodes are terminal

- Optimization seeks to improve a program's resource utilization
 - Execution time (most often)
 - Code size
 - Network messages sent, etc.
- Optimization should not alter what the program computes
 - The answer must still be the same

- For languages like C and Cool there are three granularities of optimizations
 1. Local optimizations
 - Apply to a basic block in isolation
 2. Global optimizations
 - Apply to a control-flow graph (method body) in isolation
 3. Inter-procedural optimizations
 - Apply across method boundaries
- Most compilers do (1), many do (2), few do (3)

- In practice, often a conscious decision is made not to implement the fanciest optimization known
- Why?
 - Some optimizations are hard to implement
 - Some optimizations are costly in compilation time
 - Some optimizations have low payoff
 - Many fancy optimizations are all three!
- Goal: Maximum benefit for minimum cost