

Midterm Review

Today

- Overview of what we have learned so far
- Lattice terminology review
- Dominance frontier example
- Pessimistic global value numbering
- Induction variable identification
- PRE example

Big Picture: Traditional View of Compilers

Compiling down

- Translate high-level language to machine code

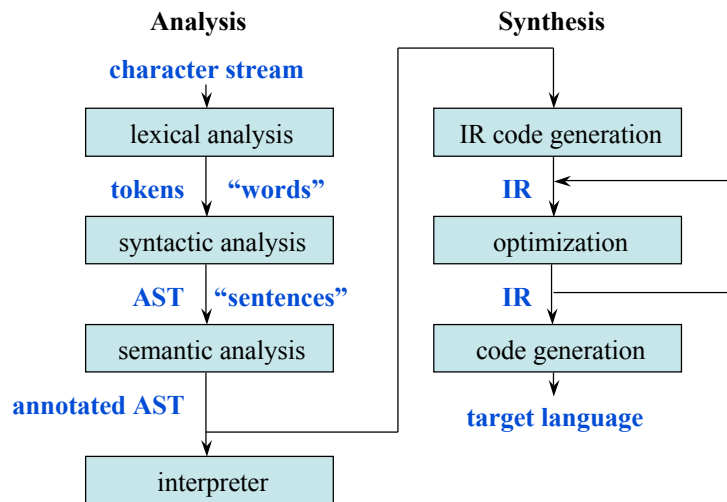
High-level programming languages

- Increase programmer productivity
- Improve program maintenance
- Improve portability

Low-level architectural details

- Instruction set
- Addressing modes
- Pipelines
- Registers, cache, and the rest of the memory hierarchy
- Instruction-level parallelism

Structure of a Typical Compiler



Topics

I. The Basics

- Scanning and parsing
 - bison and flex as parser generation and lexical analysis generation tools
- Dataflow analysis
 - Examples: reaching definitions, liveness, reaching constants, ...
 - Theoretic framework built on lattices
- Control flow analysis: control-flow graphs, dominators, dominance frontiers, irreducibility

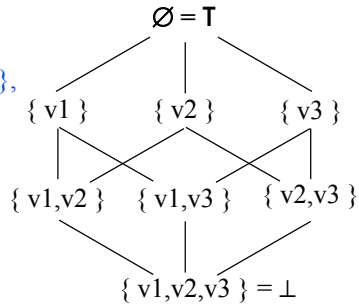
II. Analyses and Representations

- SSA Form: types of data dependencies, how to translate to minimal SSA
- Program optimizations
 - dead-code elimination, constant propagation (simple constants)
 - CSE, loop-invariant code motion, PRE, copy propagation
 - induction variable elimination, strength reduction, global value numbering
- Aliases
 - what induces aliases?
 - what is the difference between alias pairs and the points-to representation?
 - how do we characterize alias analysis algorithms?

Visualizing DFA Frameworks as Lattices

Example: Liveness analysis with 3 variables
 $S = \{v1, v2, v3\}$

- V: $2^S = \{\{v1, v2, v3\}, \{v1, v2\}, \{v1, v3\}, \{v2, v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset\}$
- Meet (\sqcap): \cup
 - \sqsubseteq : \supseteq
 - Top(T): \emptyset
 - Bottom (\perp): V
- F: $\{f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n\}$



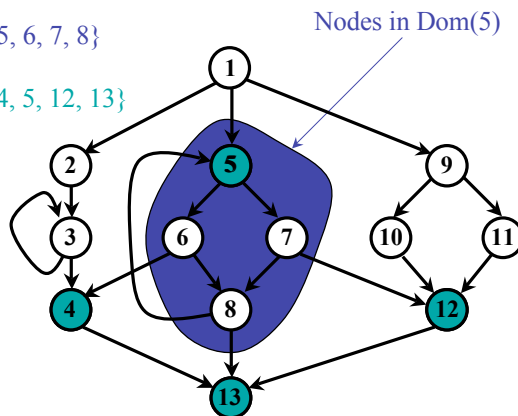
Inferior solutions are lower on the lattice
 More conservative solutions are lower on the lattice

Dominance Frontier Example

$DF(d) = \{n \mid \exists p \in \text{pred}(n), d \text{ dom } p \text{ and } d \not\text{dom } n\}$

$\text{Dom}(5) = \{5, 6, 7, 8\}$

$DF(5) = \{4, 5, 12, 13\}$



What's significant about the Dominance Frontier?

In SSA form, definitions must dominate uses

Pessimistic Global Value Numbering

Idea

- Initially each variable is in its own congruence class
- Consider each assignment statement s (reverse postorder in CFG)
 - Update LHS value number with hash of RHS
- Identical value number \Rightarrow congruence

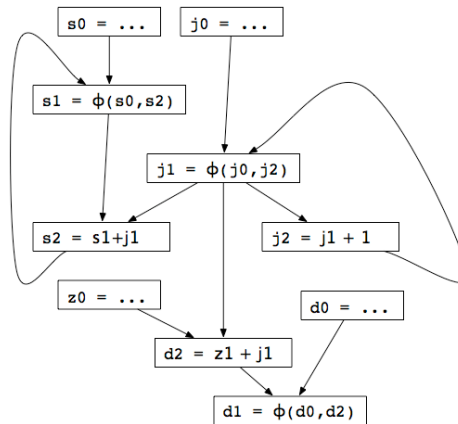
x		#1
$x+1$		#2
i		#2
k		#2
$i*2$	*(#2,2)	#3
d		#3
$k*2$	*(#2,2)	#3
y		#3

$i_0 = x_0 + 1$
$k_0 = x_0 + 1$
$d_0 = i_0 * 2$
$y_0 = k_0 * 2$

Induction Variable Identification

$s = \dots$
 $j = 0$
 $z = \dots$
 $d = \dots$

loop:
 $s = s + j$
 $d = z + j$
 $j = j + 1$
if ($j \leq 10$) **goto** loop



Derived: function of a basic as well as a loop invariant or itself
Results: j is a basic induction variable, d and s are derived

Global Possible Placement

The placement will create a redundancy on every edge out of the block

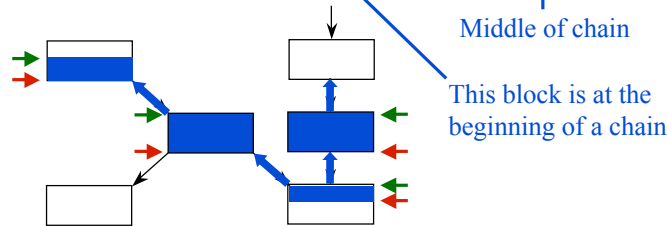
Flow Functions

Will turn partial redundancy into full redundancy

$$ppout[n] = \bigcap_{s \in succ[n]} ppin[s]$$

$$ppin[n] = anticipated_in[n] \cap partially_available_in[n] \cap$$

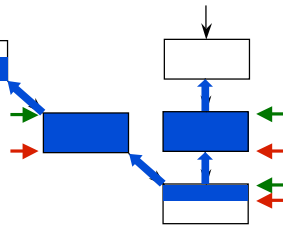
$$(locally_anticipated[n] \cup (ppout[n] \cap transparent[n]))$$



Updating Blocks

Intuition

- Perform insertions at top of the chain
- Perform deletion at the bottom of the chain



Functions

$$\text{delete}[n] = ppin[n] \cap locally_anticipated[n]$$



$$\text{insert}[n] = ppout[n]$$

$$\bigcap (\neg ppin[n] \cup \neg transparent[n])$$



$$\bigcap \neg available_out[n]$$

Don't insert it where it's fully redundant

Performing Insertions and Deletions

Overview

- find each delete expression and replace it with a temporary variable, keep track of expression mapped to variable
- find each statement $(x = \text{expr})$ where expr is an insert expression
- insert $\text{tempvar} = x$ after the found statement

OR

- find each delete expression and replace it with a temporary variable, keep track of expression mapped to variable
- for each delete block, put $(\text{tempvar} = \text{expr})$ before first computation of the expression (have to be careful it really is the same expression)
- find each statement $(x = \text{expr})$ where expr is the insert expression and replace it with $(x = \text{tempvar})$