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
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

\[ S = \{v1, v2, v3\} \]

- \( V: \quad 2^S = \{\{v1,v2,v3\}, \{v1,v2\}, \{v1,v3\}, \{v2,v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset\} \)
  - Meet (\( \cap \)): \( \cup \)
  - \( \sqsubseteq \):
  - Top(T): \( \emptyset \)
  - Bottom (\( \bot \)): \( V \)

- \( F: \quad f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n \)

\[ \emptyset = T \]

\[ \{ v1 \} \quad \{ v2 \} \quad \{ v3 \} \]

\[ \{ v1,v2 \} \quad \{ v1,v3 \} \quad \{ v2,v3 \} \]

\[ \{ v1,v2,v3 \} = \bot \]

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

---

**Dominance Frontier Example**

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

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

\[ \text{DF}(5) = \{4, 5, 12, 13\} \]

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

$$
\begin{align*}
  x & \quad \#1 \\
  x + 1 & \quad \#2 \\
  i & \quad \#2 \\
  k & \quad \#2 \\
  i^2 & \quad *(\#2, 2) \quad \#3 \\
  d & \quad \#3 \\
  k^2 & \quad *(\#2, 2) \quad \#3 \\
  y & \quad \#3
\end{align*}
$$

Induction Variable Identification

$$
\begin{align*}
  s &= \ldots \\
  j &= 0 \\
  z &= \ldots \\
  d &= \ldots \\
\end{align*}
$$

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

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

\[ ppin[n] = \text{anticipated_in}[n] \bigcap \text{partially_available_in}[n] \bigcap \left( \text{locally_anticipated}[n] \bigcup \left( ppout[n] \bigcap \text{transparent}[n] \right) \right) \]

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

Will turn partial redundancy into full redundancy.

Updating Blocks

Intuition

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

Functions

\[ \text{delete}[n] = ppin[n] \bigcap \text{locally_anticipated}[n] \]

\[ \text{insert}[n] = ppout[n] \bigcap \left( \neg ppin[n] \bigcup \neg \text{transparent}[n] \right) \bigcap \neg \text{available_out}[n] \]

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 \(\text{expr}\) 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 \(\text{expr}\) is the insert expression and replace it with \((x = \text{tempvar})\)