“Classic” constant propagation

- Idea: propagate and fold integer constants in one pass

\[
x = 1; \quad x = 1;
\]
\[
y = 5 + x; \quad y = 6;
\]
\[
z = y \times y; \quad z = 36;
\]

- Information about a single variable:
  i. Variable never defined
  ii. Variable has single constant value
  iii. Variable has multiple values

One-variable Const. Prop.

- never defined
- constant
- constant \( c_1 \)
- constant \( c_2 \)
- multiple

Full lattice:

Rest of definition

- Flow function for \( x = x \) \( \text{OP} \) \( c_1 \):
  \[
  F_n(\top) = \top \\
  F_n(\bot) = \bot \\
  F_n(c_2) = c_2 \text{OP} c_1
  \]
- Flow function is monotonic, distributive: iterative solution works, gives MOP
- What about multiple variables \( x_1 \ldots x_n \)?
  Want tuple \( (v_1 \ldots v_n) \),
**Multiple vars**

- Dataflow value is a tuple \((v_1, \ldots, v_n)\), each \(v_i\) in lattice \(L = \ldots -2 -1 0 1 2 3 \ldots\)

- Set of tuples \((v_1, \ldots, v_n)\) is also a lattice under component-wise ordering:
  \[ (v_1, \ldots, v_n) \subseteq (v'_1, \ldots, v'_n) \iff \forall i, v_i \subseteq v'_i \]
  \[ (v_1, \ldots, v_n) \sqcap (v'_1, \ldots, v'_n) = (v_1 \sqcap v'_1, \ldots, v_n \sqcap v'_n) \]

- For any two lattices \(L_1, L_2\), have product lattice \(L_1 \times L_2\):
  \( (v_1, v_2) \subseteq (v'_1, v'_2) \iff v_1 \subseteq v'_1 \quad \text{and} \quad v_2 \subseteq v'_2 \)

- Tuple dataflow values are in \(L_1 \times \ldots \times L_n = L^n\)

**Flow functions**

- Consider \(x_1 = x_2 \text{ OP } x_3\)
  \[ F(x_1, \top, x_3) = (\top, \top, x_3) \]
  \[ F(x_1, x_2, \top) = (\top, x_2, \top) \]
  \[ F(x_1, x_2, \bot) = (\bot, x_2, \bot) \]
  \[ F(x_1, c_2, c_3) = (c_2 \text{ OP } c_3, c_2, c_3) \]

- Monotonic? Distributes over \(\sqcap\)?

**Loops**

- Most execution time in most programs is spent in loops: 90/10 is typical
- Most important targets of optimization: loops
- Loop optimizations:
  - loop-invariant code motion
  - loop unrolling
  - loop peeling
  - strength reduction of expressions containing induction variables
  - removal of bounds checks
  - loop tiling
- When to apply loop optimizations?
High-level optimization?

- Loops may be hard to recognize in IR or quadruple form -- should we apply loop optimizations to source code or high-level IR?
  - Many kinds of loops: while, do/while, continue
  - Loop optimizations benefit from other IR-level optimizations and vice-versa -- want to be able to interleave

- Problem: identifying loops in flowgraph

Definition of a loop

- A loop is a set of nodes in the control flow graph, with one distinguished node called the header (entry point)
- Every node is reachable from header, header reachable from every node: strongly-connected component
- No entering edges from outside except to header
- Nodes with outgoing edges: loop exit nodes

Nested loops

- Control-flow graph may contain many loops, and loops may contain each other

- Control-flow analysis: identify the loops and nesting structure:

Dominators

- CFA based on idea of dominators
- Node A dominates node B if the only way to reach B from start node is through A
- Edge in flowgraph is a back edge if destination dominates source
- A loop contains at least one back edge
### Dominator tree

- Domination is transitive; if A dominates B and B dominates C, then A dominates C.
- Domination is anti-symmetric.
- Every flowgraph has dominator tree (Hasse diagram of domination relation).

### Dominator dataflow analysis

- Forward analysis; out[n] is set of nodes dominating n.
- "A node B is dominated by another node A if A dominates all of the predecessors of B."
  
  \[
  \text{in}[n] = \cap_{n' \in \text{pred}[n]} \text{out}[n']
  \]
  
  "Every node dominates itself."
  
  \[
  \text{out}[n] = \text{in}[n] \cup \{n\}
  \]

- Formally: \( L = \) sets of nodes ordered by \( \subseteq \), flow functions \( F_n(x) = x \cup \{n\}, \cap = \cap, \top = \{\text{all } n\} \)

\[\Rightarrow\] Standard iterative analysis gives best soln.

### Completing control-flow analysis

- Dominator analysis gives all back edges.
- Each back edge \( n \rightarrow h \) has an associated natural loop with \( h \) as its header: all nodes reachable from \( h \) that reach \( n \) without going through \( h \).
- For each back edge, find natural loop.
- Nest loops based on subset relationship between natural loops.
- Exception: natural loops may share same header; merge them into larger loop.
- Control tree built using nesting relationship.