Control dependence

Must ensure side-effects occur in proper order
Must ensure side-effects occur only under right conditions

CFG represents control dependence explicitly
  – but overspecifies control dependence requirements

Control dependence graph

Program dependence graph (PDG):
  data dependence graph + control dependence graph (CDG)
  [Ferrante, Ottenstein, & Warren, TOPLAS 87]

Idea: represent controlling conditions directly
  • complements data dependence representation

A node (basic block) \( Y \) is control-dependent on another \( X \) iff
  \( X \) determines whether \( Y \) executes, i.e.
  • there exists a path from \( X \) to \( Y \) s.t. every node in the path
    other than \( X \) & \( Y \) is post-dominated by \( Y \)
  • \( X \) is not post-dominated by \( Y \)

Control dependence graph:
  \( Y \) proper descendant of \( X \) iff \( Y \) control-dependent on \( X \)
  • label each child edge with required branch condition
  • group all children with same condition under region node

Two sibling nodes execute under same control conditions \( \Rightarrow \)
  can be reordered or parallelized, as data dependences allow

Challenging to “sequentialize” back into CFG form

Example

1. \( y := p + q \)
2. \( x > 0? \)
3. \( a := x \times y \)
4. \( a := y - 2 \)
5. \( w := y / q \)
6. \( x > 0? \)
7. \( b := 1 << w \)
8. \( x := a \& b \)

An example with a loop
Value dependence graphs

[Weise, Crew, Ernst, & Steensgaard, POPL 94]

Idea: represent all dependences, including control dependences, as data dependences
+ simple, direct dataflow-based representation of all “interesting” relationships
  • analyses become easier to describe & reason about
  – harder to sequentialize into CFG

Control dependences as data dependences:
  • control dependence on order of side-effects
    ⇒ data dependence on reading & writing to global Store
  • optimizations to break up accesses to single Store into separate independent chunks
    (e.g. a single variable, a single data structure)
  • control dependence on outcome of branch
    ⇒ a select node, taking test, then, and else inputs
    ⇒ demand-driven evaluation model

Loops implemented as tail-recursive calls to local procedures

Apply CSE, folding, etc. as nodes are built/updated

VDG for example, after store splitting

\[
\begin{align*}
y & := p + q \\
\text{if } x > 0 \text{ then } a & := x * y \text{ else } a := y - 2 \\
w & := y / q \\
\text{if } x > 0 \text{ then } b & := 1 << w \\
r & := a \% b
\end{align*}
\]