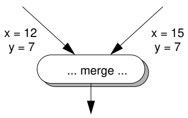
Introduction

Primary goal: Mark RTL values as static or dynamic

- “Static” in this context means “known at dynamic compile time”
- “Dynamic” - not known at dynamic compile time (*aka* stitch time)

Static values can be computed at dynamic compile time; code will be emitted to compute dynamic values at run time

Problem: What to do at merges?

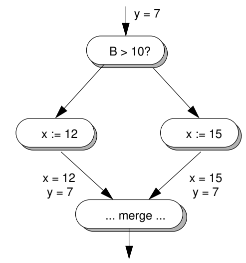


Is `x` a dynamic constant after the merge? Is `y`?

Merges

Merges are tagged as either:

- “Real”: multiple incoming paths might actually exist at stitch time
  - Variable is dynamic constant iff it has the same value on all incoming paths
- “Phantom”: only a single path will actually exist at stitch time
  - Variable is dynamic constant iff it is a dynamic constant on all incoming paths

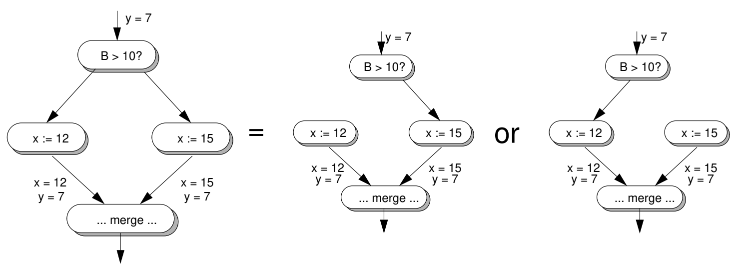


Family of Graphs

Values of static variables ⇒ different graphs

- Want to analyze family of graphs, determined by values of static variables

If B is static, could get either graph:



For each edge, compute conditions under which that edge is reachable

- Merge is phantom if only one path enters merge for every graph in family

Identifying Phantom Merges

Can use special merge rule when only one of incoming paths will exist at dynamic compile-time

Possible approaches:

Rules for single-entry, single-exit regions of CFG

+ Lots of previous work (Johnson & Pingali PLDI’93, Johnson’s thesis)
- Doesn’t work for unstructured control-flow (such regions aren’t SESE)

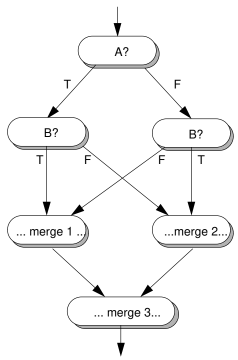
Control-dependence (Ferrante *et al* TOPLAS’87)

- Doesn’t work for unstructured control-flow

Custom analysis, based on tracking potential paths at program points

+ Works for unstructured control-flow ==> can give better results
+ New research
- New research

Unstructured control-flow example



Reachability-Tracking

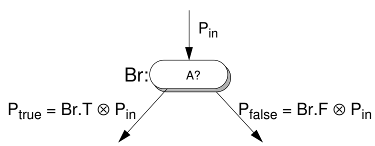
Want to compute the family of graphs for which each edge can be reached

A member of the family is specified by an assignment of values to the static variables

Compute the set of static variable assignments for each edge for which that edge will exist

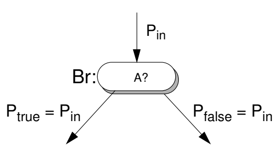
Handling Branches

If branch *Br* is static (i.e. *A* is a static variable):

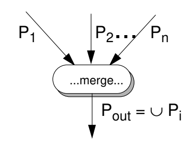


$A.x \otimes P$ - All paths in P with the additional constraint that branch node A went in direction x

If branch is dynamic:

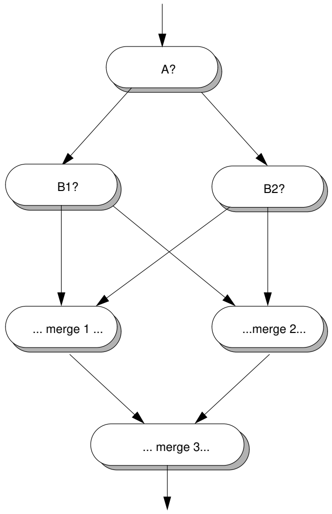


Handling Merges



A merge is “phantom” if the intersection of each pair of sets entering the merge is disjoint

- This implies that only one of the paths will exist in any member of the family of graphs



Fringe Benefits

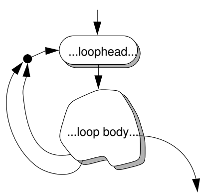
Template boundaries are made explicit by the labelling of edges

- templates are formally labelled by the conditions under which they execute
- template is region of code that exists or does not exist in particular instances of the family of graphs

Loop Unrolling

Loops that are to be completely unrolled need special rules:

- View as family of graphs where the loop is unrolled, but by an unknown number of times
- Treat back edges as static merges



Becomes