À la recherche du temps perdu

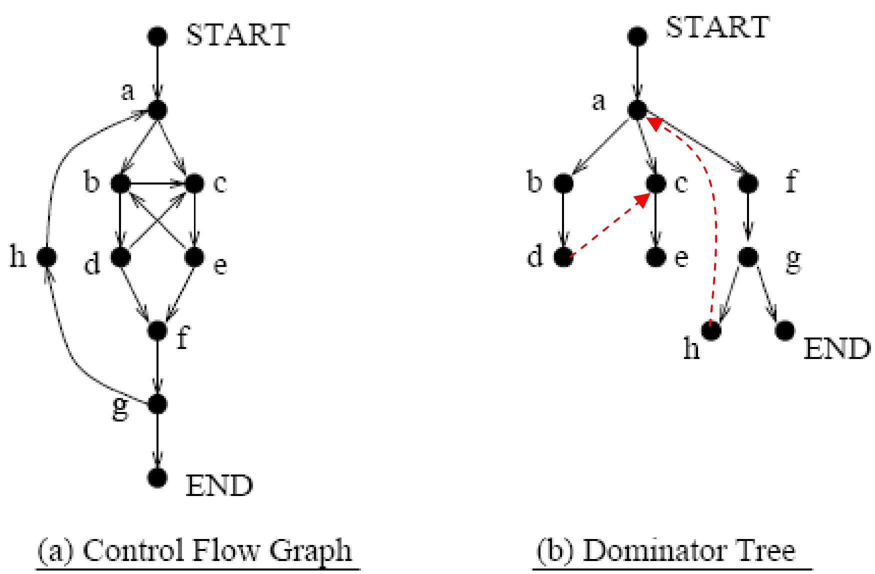
Keshav Pingali

University of Texas at Austin

Background

- Paper:
  - “Algorithms for Computing the Static Single Assignment Form”, Gianfranco Bilardi and Keshav Pingali, Journal of the ACM, 50(3), May 2003.
- Results discussed in talk:
  - Merge relation: useful relation for ϕ-placement algorithms
  - Structure of Merge relation
  - Three classes of ϕ-placement algorithms
    - Two-phase (Cytron et al)
    - Lock-step
    - Lazy (Sreedhar & Gao)
  - Optimal algorithms for single-variable ϕ-placement
    - Lock-step and lazy algorithms
  - Optimal algorithm for multiple-variable ϕ-placement
    - For structured programs
    - Two-phase algorithm

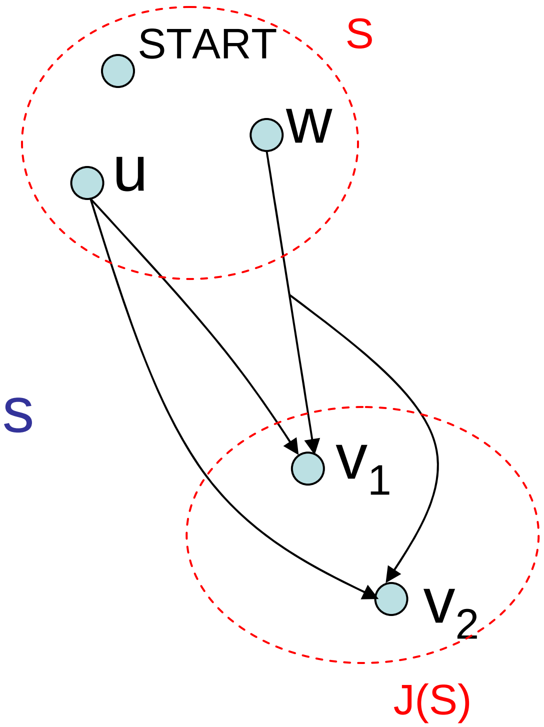
Dominators and CFG edges



- Dominance: relation on nodes ($\subseteq V \times V$)
  - u dominates v if u occurs on all paths START $\rightarrow$+ v
    - Example: dominators of f are START, a, f
  - dominance is transitive and transitive reduction is tree-structured
  - dominator tree can be built in O(|E|+|V|) time (Buchsbaum et al)
- CFG edges (u $\rightarrow$ v) can be classified into
  - dominator tree edges: if u dominates v (example: a $\rightarrow$ b)
  - up-edges: idom(v) dominates u (examples: h $\rightarrow$ a, d $\rightarrow$ c)

Join relation

- CFG: G = (V,E)
- Set S: nodes that have assignments to given variable
  - $\{START\} \subseteq S \subseteq V$
- Where should the corresponding ϕ-functions be placed?
- Join relation J gives answer [Cytron et al]
- J: $\mathscr{P}(V) \to \mathscr{P}(V)$
  - $v \in J(S)$ if $\exists\, u,w \in S$ such that there are paths
    - $u \to^+ v$
    - $w \to^+ v$
    - intersecting only at v

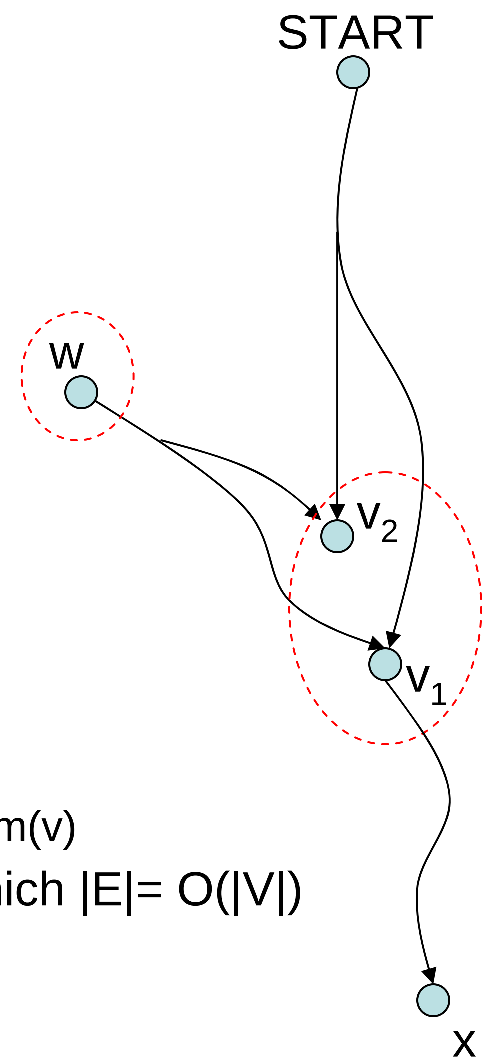


Optimal ϕ-placement algorithms

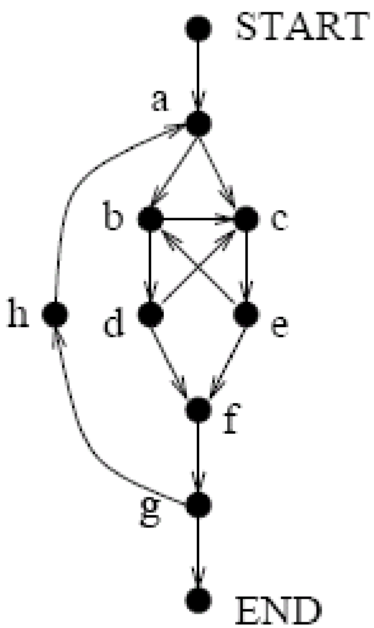
- Optimal algorithm for a single variable
  - Computes J(S) for one set S in O(|V| + |E|)
    - need at least this much time to read CFG
- Optimal algorithm for several variables
  - May be desirable to preprocess CFG to obtain data structure that facilitates computation of J(S) for any S
  - Preprocessing time = O(|V|+|E|)
    - need at least this much time to read CFG
  - Query time(S) = O(|S|+|J(S)|)
    - need at least this much time to read S and output J(S)

Merge relation

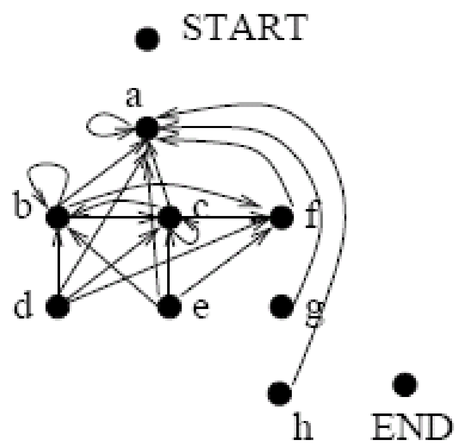
- Computing J efficiently
  - J: $\mathcal{P}(V) \rightarrow \mathcal{P}(V)$
  - Might be easier to compute function $V \rightarrow \mathcal{P}(V)$
- Merge relation M: $\subseteq V \times V$
  - $v \in M(w)$ if $v \in J(\{START,w\})$
    - variable assigned only at START and w
    - ϕ-node needed at v
- Properties of M relation
  1. $J(S) = \cup_{w \in S} M(w)$ (superposition)
  2. (M-paths)
     - M-path: $P = w \rightarrow^+ v$ that does not contain idom(v)
     - $v \in M(w)$ iff $\exists$ path $P = w \rightarrow^+ v$ that does not contain idom(v)
  3. size of M relation can be $\Omega(|V|^2)$ even for graphs for which $|E|= O(|V|)$
  4. M is a transitive relation
     - M-paths are closed under concatenation



Example



(a) Control Flow Graph

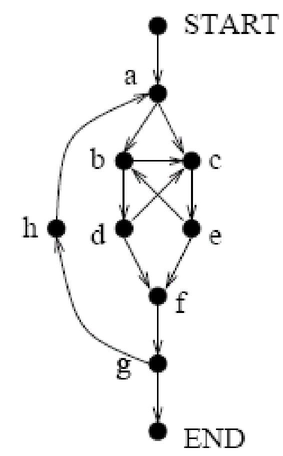


(c) M graph

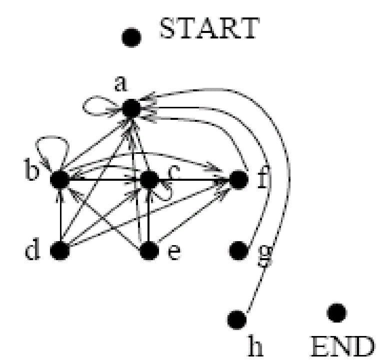
- J({START,d,a}) = M(d) U M(a) = {b,a,c,f} U {a} = {b,a,c,f}
- J({START,d,h}) = M(d) U M(h) = {b,a,c,f} U {a} = {b,a,c,f}

Structure of M relation

- M graph has non-trivial cycle iff CFG is irreducible
- ω–ordering of CFG nodes
  - Reducible programs
    - M graph is acyclic (ignoring self-loops)
    - Topological sort of the nodes of the M graph can be found in O(|V|+|E|) time (without building the M graph!)
  - Irreducible programs
    - Strongly-connected components and topological sort of acylic condensate can be found in O(|V|+|E|) time
    - Example: d,e,bc,f,g,h,a



(a) Control Flow Graph

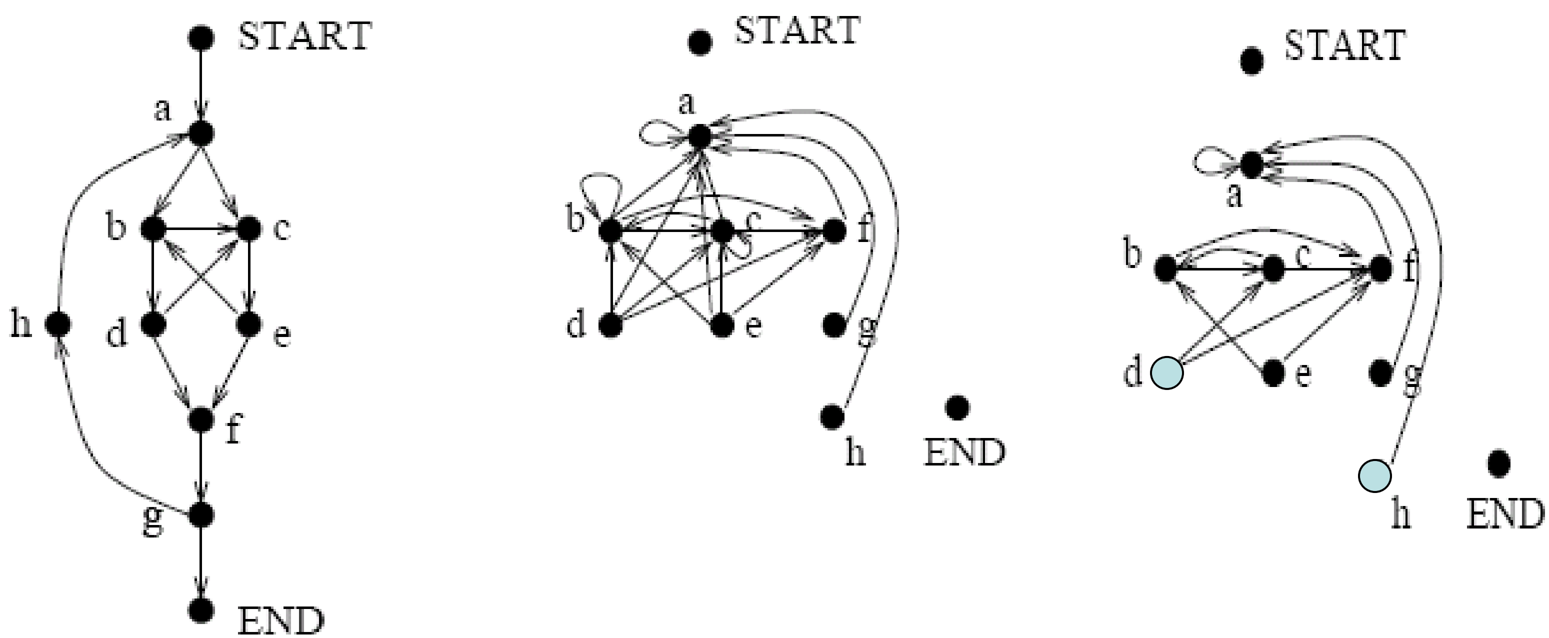


(c) M graph

Transitive reductions of M

- Can we exploit fact that M is transitive?
  - Goal:
    - Compute M_{red} = transitive reduction of M
    - J(S) = set of nodes reachable from nodes in S by non-empty paths in graph of M_{red}
- Unfortunately,
  - M is a cyclic relation in general, so transitive reduction is not unique
  - Not easy to compute a transitive reduction
- *Partial* transitive reduction: dominance frontier (Cytron et al)
  - $v \in DF(w)$ if $\exists$ path $P = w \rightarrow^* u \rightarrow v$ such that
    - w dominates all nodes on prefix $w \rightarrow^* u$
    - w does not strictly dominate v
  - DF-paths are the prime paths corresponding to M-paths
- Computing J from DF
  - same strategy as above

Example



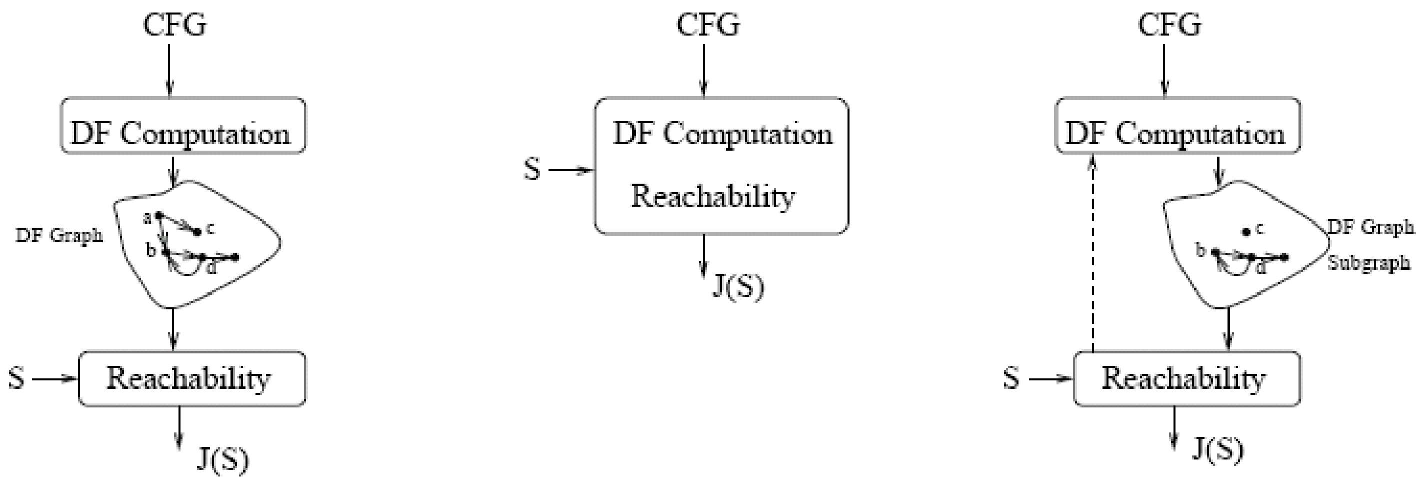
(a) Control Flow Graph (c) M graph (e) *DF* Graph

- In general, DF is neither transitively closed nor transitively reduced
- J({START,d,h}) = M(d) U M(h)

  = set of nodes reachable from d and h in DF graph

  = {c,b,f,a} U {a} = {c,b,f,a}

Three strategies for computing J(S)

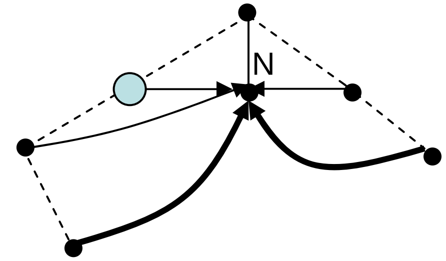


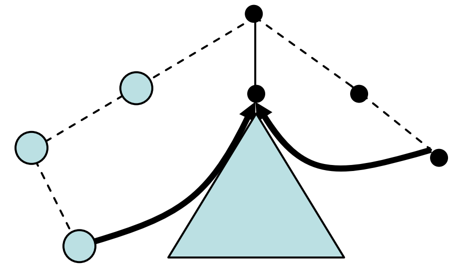
(1) Two-phase algorithms (2) Lock-step algorithms (3) Lazy algorithms

1. Two-phase algorithms: (Cytron et al)
   - Compute entire DF graph
   - Perform reachability computation in DF graph
   - There are graphs for which |E| = O(|V|) but DF = O(|V|^2), so two-phase algorithms cannot be asymptotically optimal for general graphs
2. Lock-step algorithms:
   - Interleave computations of DF graph and reachability to avoid building the entire DF graph
   - Computes a sub-graph DF' that has same nodes as DF but may not have all the edges
3. Lazy algorithms: (Sreedhar and Gao, Bilardi and Pingali)
   - Compute portions of the DF graph on demand as needed for reachability computation
   - Computes a sub-graph DF' that may have fewer nodes and edges than DF graph

Lock-step algorithm

- Intuitive idea:
  - avoid building entire DF graph
  - DF graph is used for reachability computation from nodes in S
  - Once a node N is known to be reachable from nodes in S (so N is in J(S)), further DF edges to node N do not add any information, so do not generate them
- Solution:
  - mark nodes in S
  - propagate marks down the dominator tree
  - when examining node v, if you find up-edge ($u \rightarrow v$) and u is marked, add v to J(S)
  - mark v and propagate marks down dominator tree
- Question: can we order nodes so we never have to examine nodes more than once?
  - yes, use ω-ordering

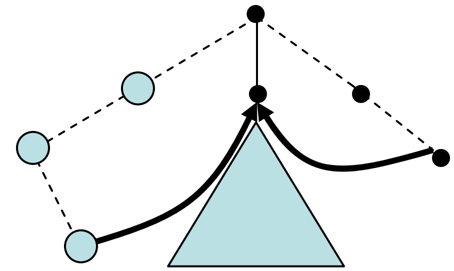




Algorithm

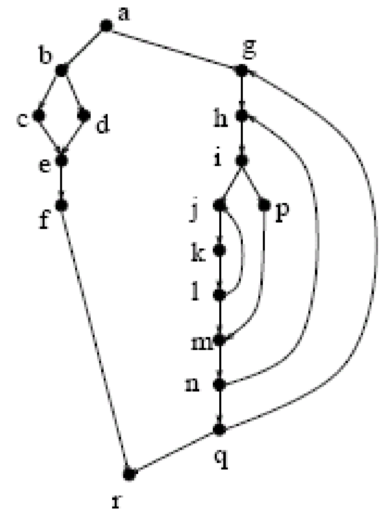
Procedure Pulling(D,S); //D is dominator tree,S is set of assignment nodes
{
Initialize $DF^+(S)$ to $\{\}$;
Initialize all nodes in dominator tree as off;

for each node v in ω-ordering **do**
  if $v \in S$ **then** TurnOn(D,v) **endif** ;
  for each up-edge $u \rightarrow v$ **do**
    if u is on **then**
      Add v to $DF^+(S)$;
      if v is off **then** TurnOn(D,v) **endif** ;
      break //exit inner loop
    endif
od
}
ProcedureTurnOn(D, x);
{
Switch x on;
for each $c \in children(x)$ in D **do**
  if c is off **then** TurnOn(D,c)
}

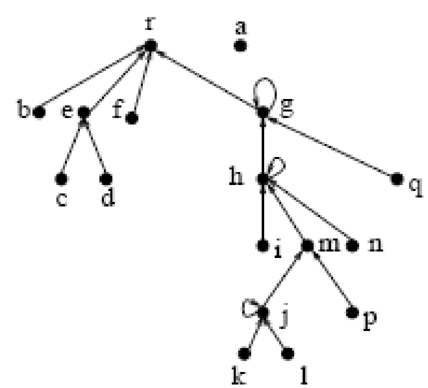


Optimal algorithm for multiple variables

- Theorem:
  - If the transitive reduction of the merge relation for a CFG is forest-structured, J sets can be found in optimal time O(|S|+|J(S)|).
- Theorem:
  - The transitive reduction of the merge relation for a structured program is forest-structured, and can be found in O(|E|+|V|) time



(a) A Structured CFG



(c) Merge Relation

Summary

- Merge relation:
  - Useful relation for ϕ-placement algorithms
  - Close connection to program structure
    - Structured programs: tree-structured relation
    - Reducible programs: DAG (may have self-loops)
    - Irreducible programs: non-trivial cycles
  - Can be used to derive ϕ-placement algorithms
    - Optimal algorithms for single variable problem
    - Optimal algorithm for multiple variables problem for structured programs