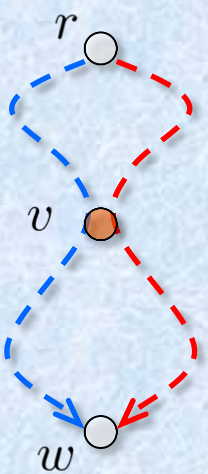
Finding Dominators in Interprocedural Flowgraphs

Loukas Georgiadis
University of Ioannina

Flowgraphs and Dominators

Flowgraph $G = (V, E, r)$: all vertices are reachable from start vertex r

v dominates w if every path from r to w includes v



$Dom(w) =$ set of vertices that dominate w

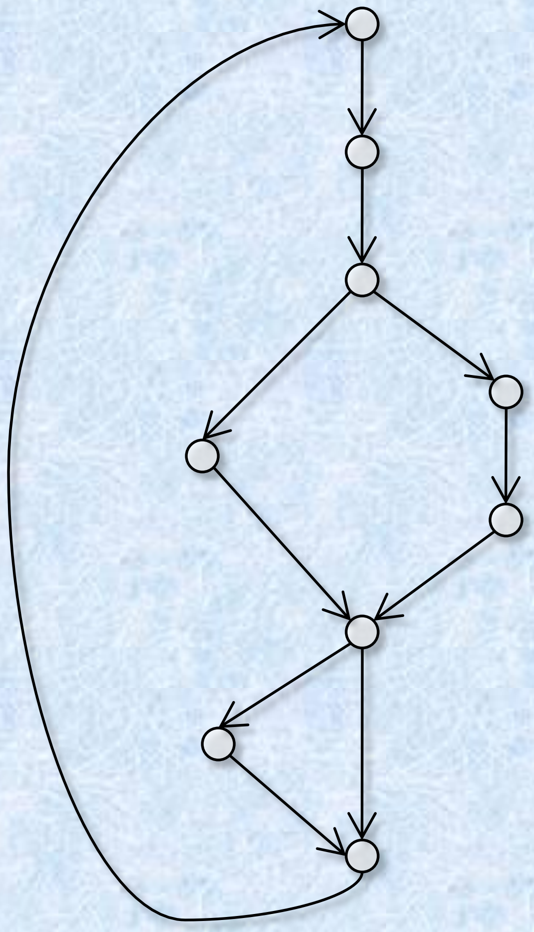
Trivial dominators : $r, w \in Dom(w)$

$idom(w) =$ immediate dominator of w; is dominated by all $Dom(w) - w$

Application areas : Program optimization, VLSI testing, theoretical biology, distributed systems, constraint programming,…

Flowgraphs and Dominators

```
for (; p<stop; p++) {
  int v = order[*p];
  if (v) {
    int u;
    if (v<=i) {u=v;}
    else {
      compress(v);
      u = label[v];
    }
    if (s[u]<s[i]) s[i] = s[u];
  }
}
```



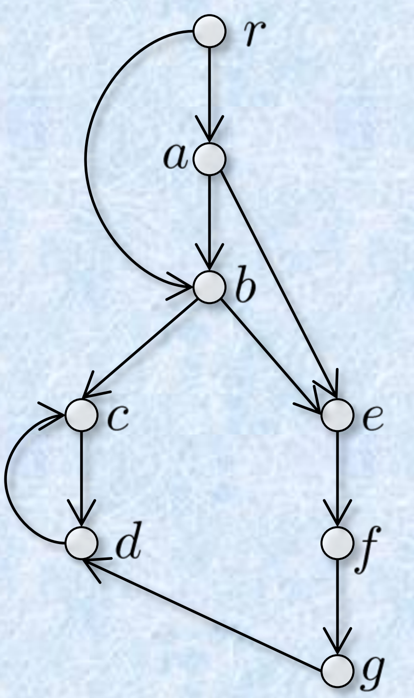
Program analysis and optimization: loop optimizations, structural analysis, control dependences,…

Flowgraphs and Dominators

Flowgraph $G = (V, E, r)$: all vertices are reachable from start vertex r

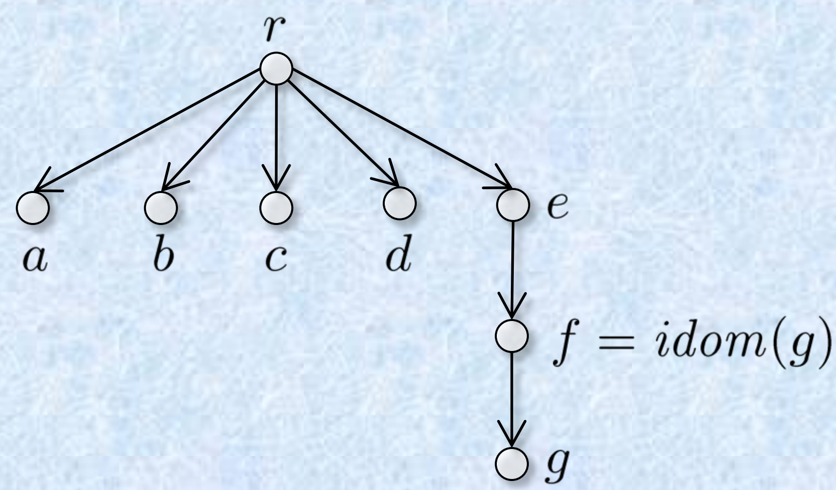
v **dominates** w if every path from r to w includes v

G



$|V| = n, |E| = m$

$D =$ dominator tree of G



$O(m\alpha(m, n))$ algorithm: [Lengauer and Tarjan '79]

$O(m + n)$ algorithms:

[Alstrup, Harel, Lauridsen, and Thorup '97]

[Buchsbaum, Kaplan, Rogers, and Westbrook '04]

[G., and Tarjan '04]

[Buchsbaum et al. '08]

Iterative Algorithm

Dominators can be computed by solving iteratively a set of equations [Allen and Cocke, 1972]

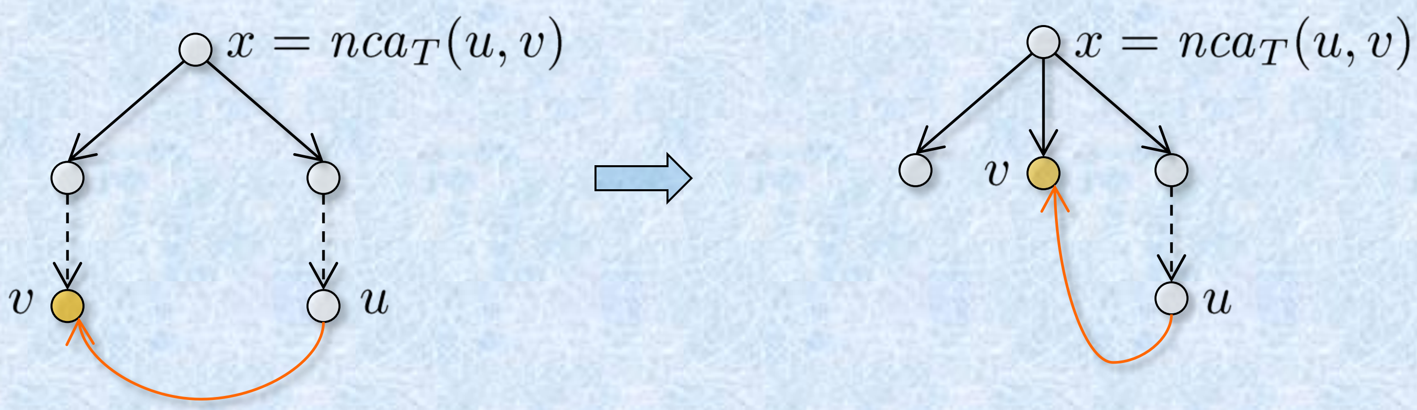
$$Dom(v) = \left(\cap_{(u,v)\in A} Dom(u) \right) \cup \{v\}, v \neq r$$

Initialization $Dom(r) = \{r\}, Dom(v) = \emptyset, v \neq r$

Efficient implementation [Cooper, Harvey and Kennedy 2000]:

Maintain tree T ; process the edges until a fixed-point is reached.

Process (u,v) : compute $x =$ nearest common ancestor of u and v in T.
If x is ancestor of parent of v, make x new parent of v.



Iterative Algorithm

Dominators can be computed by solving iteratively a set of equations [Allen and Cocke, 1972]

$$Dom(v) = \left(\cap_{(u,v)\in A} Dom(u) \right) \cup \{v\}, v \neq r$$

Initialization $Dom(r) = \{r\}, Dom(v) = \emptyset, v \neq r$

Efficient implementation [Cooper, Harvey and Kennedy 2000]:

Maintain tree T ; process the edges until a fixed-point is reached.

Process (u, v): compute $x =$ nearest common ancestor of u and v in T.
If x is ancestor of parent of v, make x new parent of v.

$O(n)$ iterations, $O(m)$ intersections per iteration, $O(n)$ time per intersection

$\Rightarrow$ $O(mn^2)$ total running time

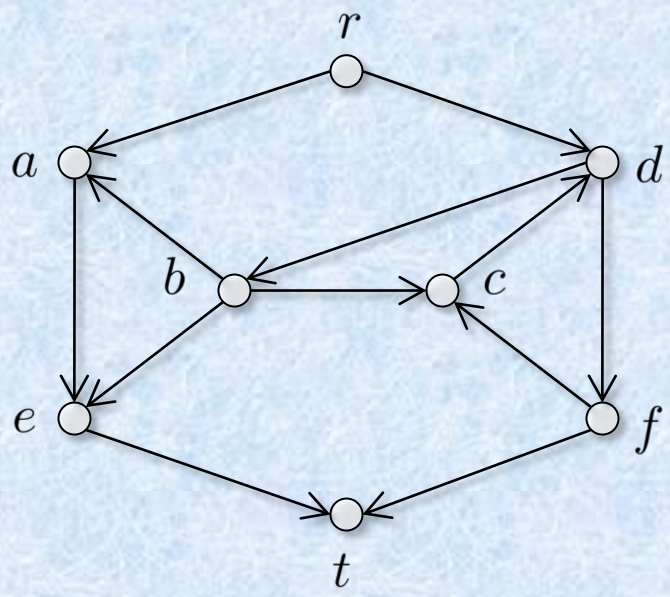
Purdom-Moore Algorithm

Uses n reachability computations $\Rightarrow$ $O(mn)$ running time

For every $x \in V$

Compute the set $U(x)$ of unreachable vertices from r in $G - x$

For every $v \in U(x)$ add x to $Dom(v)$



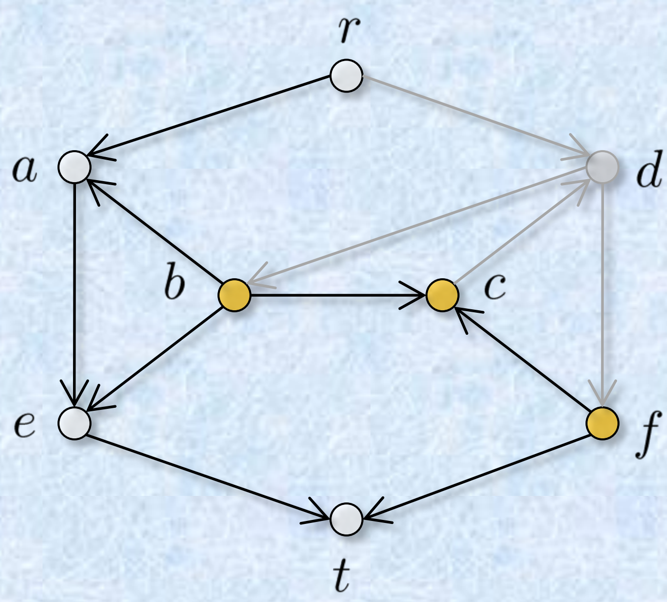
Purdom-Moore Algorithm

Uses n reachability computations $\Rightarrow$ $O(mn)$ running time

For every $x \in V$

Compute the set $U(x)$ of unreachable vertices from r in $G - x$

For every $v \in U(x)$ add x to $Dom(v)$

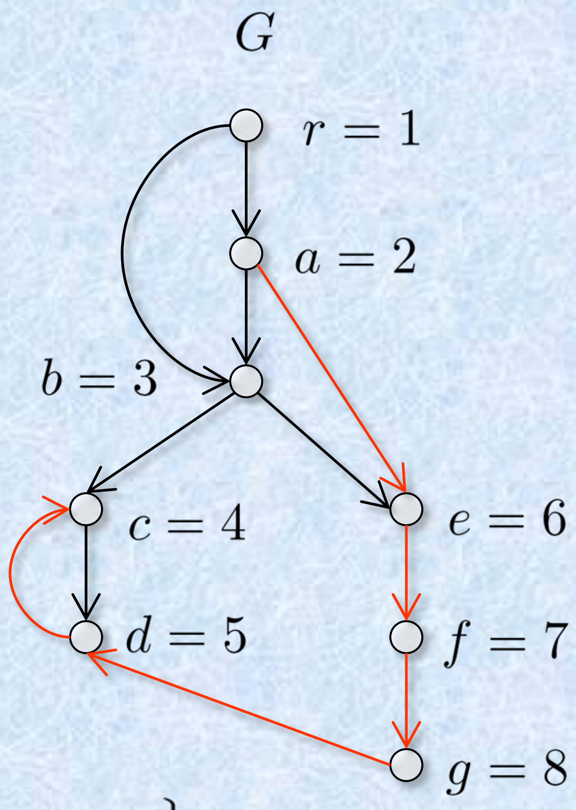


$U(d) = \{b, c, f\}$

Lengauer-Tarjan Algorithm (LT)

Depth-First Search $\Rightarrow$ DFS tree + numbering pre

$pre(v) < pre(w)$: v was visited by DFS before w



Semidominator path (sdom-path) :

$P = (v_0, v_1, v_2, \ldots, v_k)$ such that

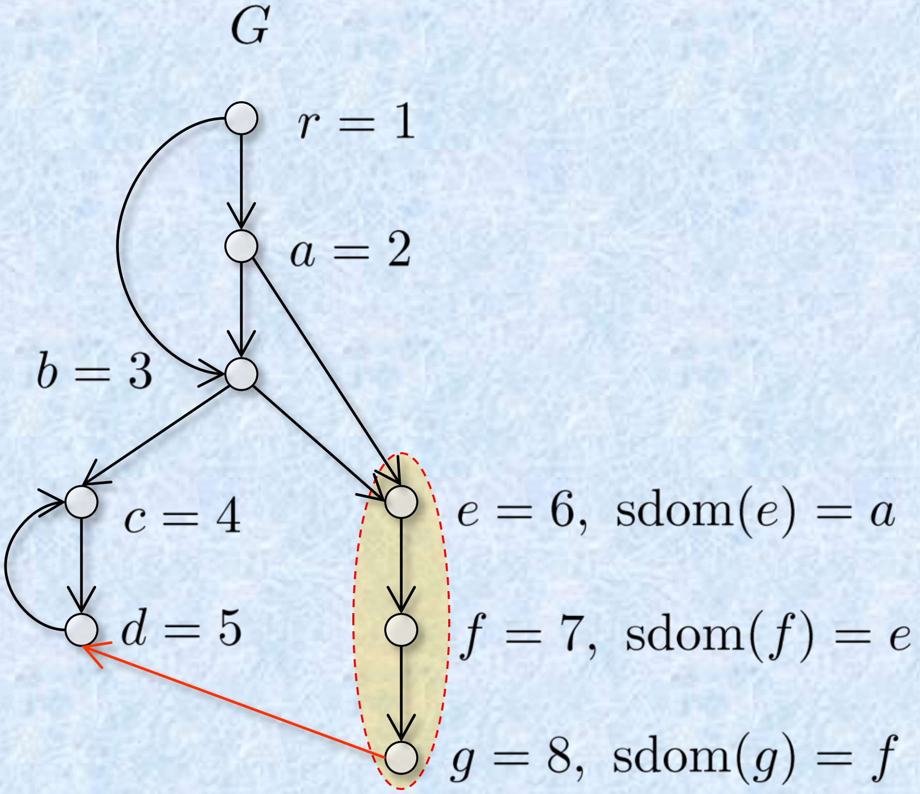
$pre(v_i) > pre(v_k), \; i = 1, 2, \ldots, k-1$

Semidominator :

$$\mathrm{sdom}(w) = \arg\min \; \{ \; pre(v) \mid \exists \text{ sdom-path from } v \text{ to } w \; \}$$

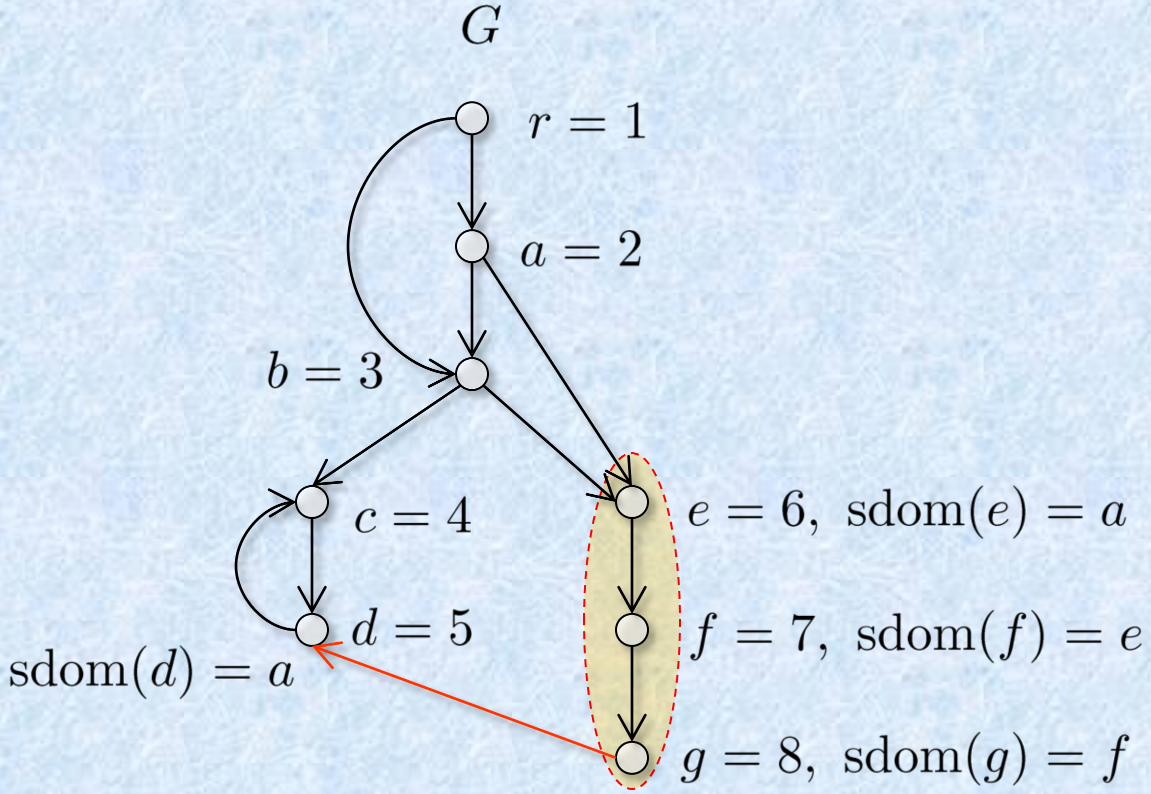
Lengauer-Tarjan Algorithm (LT)

Main operation : Compute path minima on DFS tree



Lengauer-Tarjan Algorithm (LT)

Main operation : Compute path minima on DFS tree

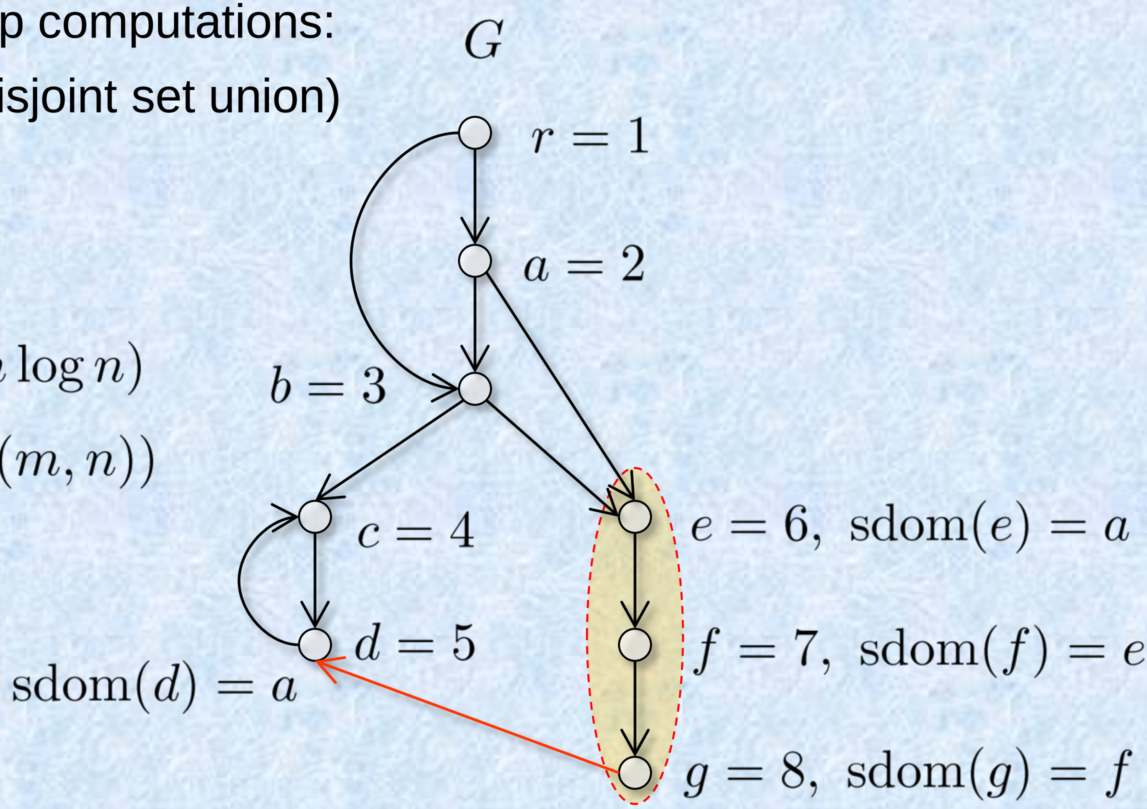


Lengauer-Tarjan Algorithm (LT)

Main operation : Compute path minima on DFS tree

Uses path compression to speed up computations:

link-eval data structure (based on disjoint set union)



Running time

-Link-eval without balancing: $O(m \log n)$

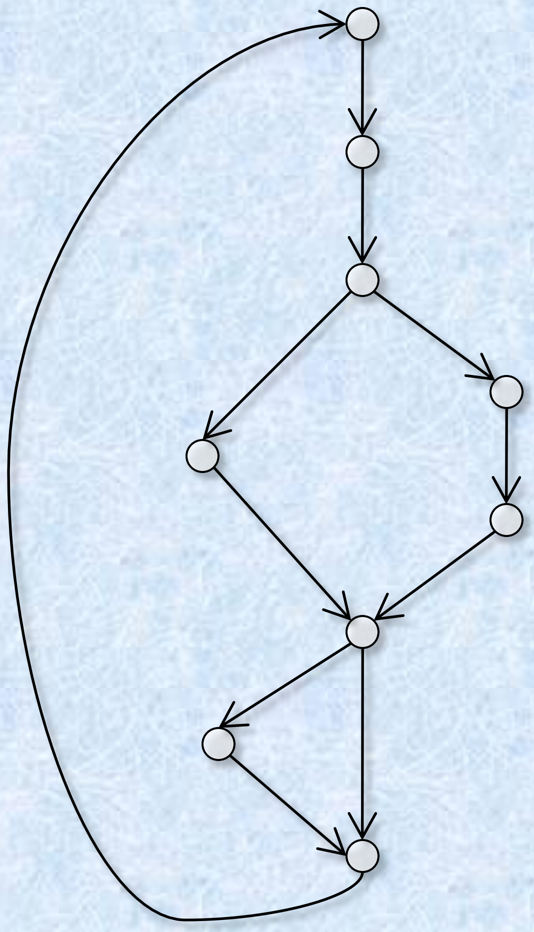
-Link-eval with balancing: $O(m\alpha(m,n))$

In practice the simple version of LT

is faster and runs in linear time

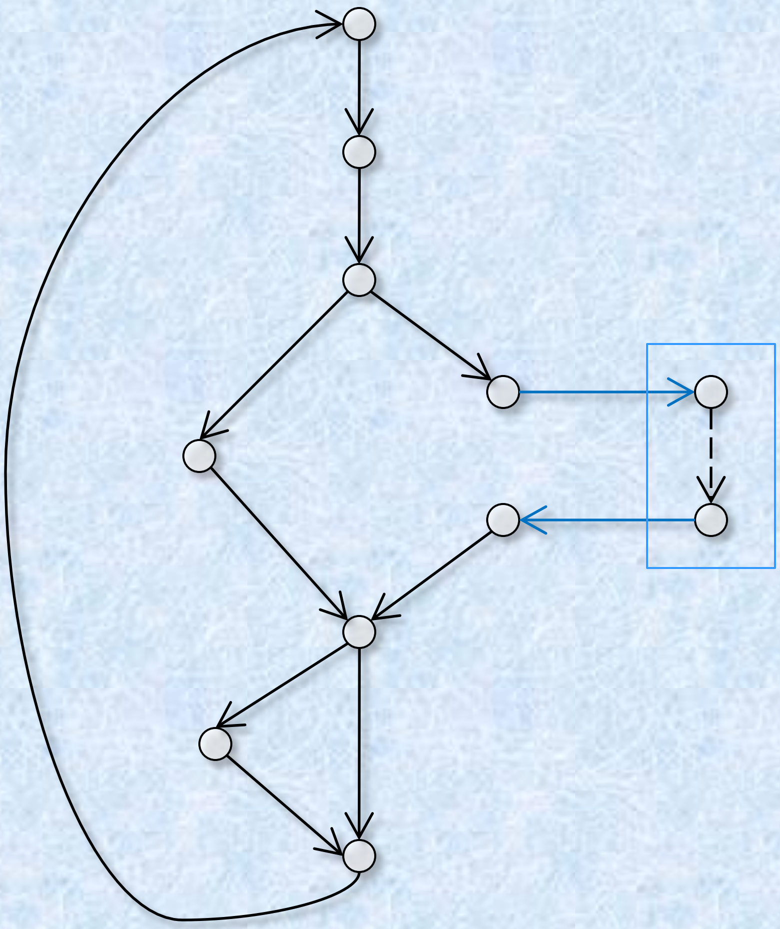
Iterprocedural Flowgraphs

```
for (; p<stop; p++) {
  int v = order[*p];
  if (v) {
    int u;
    if (v<=i) {u=v;}
    else {
      compress(v);
      u = label[v];
    }
    if (s[u]<s[i]) s[i] = s[u];
  }
}
```

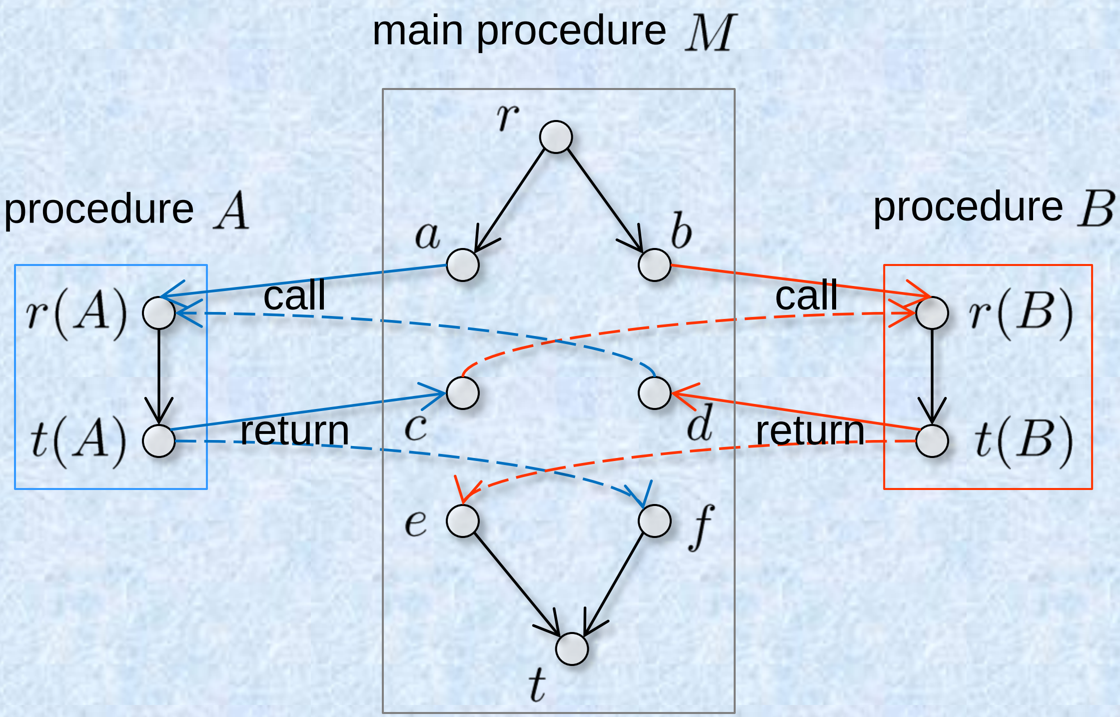


Iterprocedural Flowgraphs

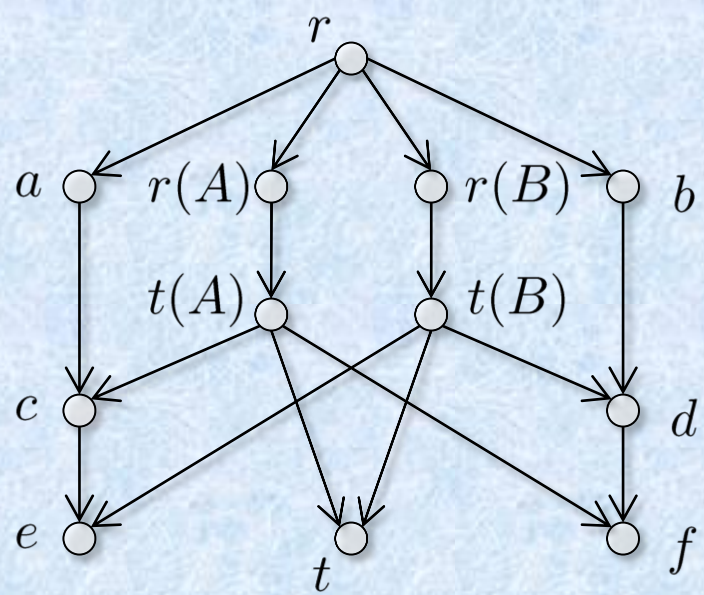
```
for (; p<stop; p++) {
   int v = order[*p];
   if (v) {
      int u;
      if (v<=i) {u=v;}
      else {
         compress(v);
         u = label[v];
      }
      if (s[u]<s[i]) s[i] = s[u];
   }
}
```



Iterprocedural Flowgraphs

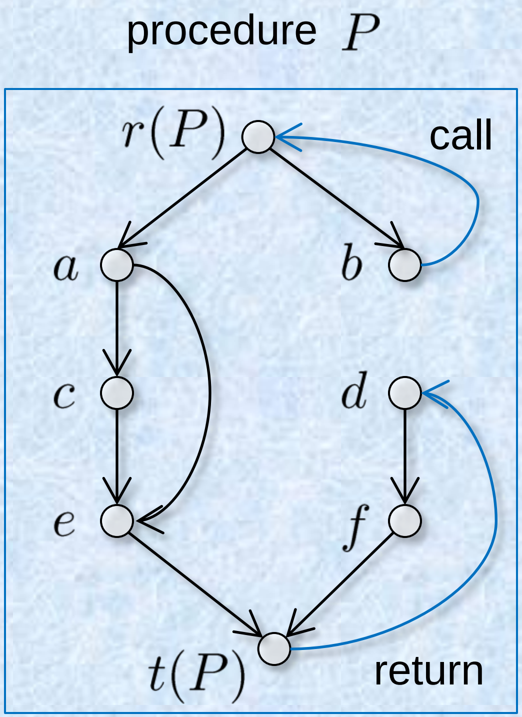


interprocedural flowgraph G

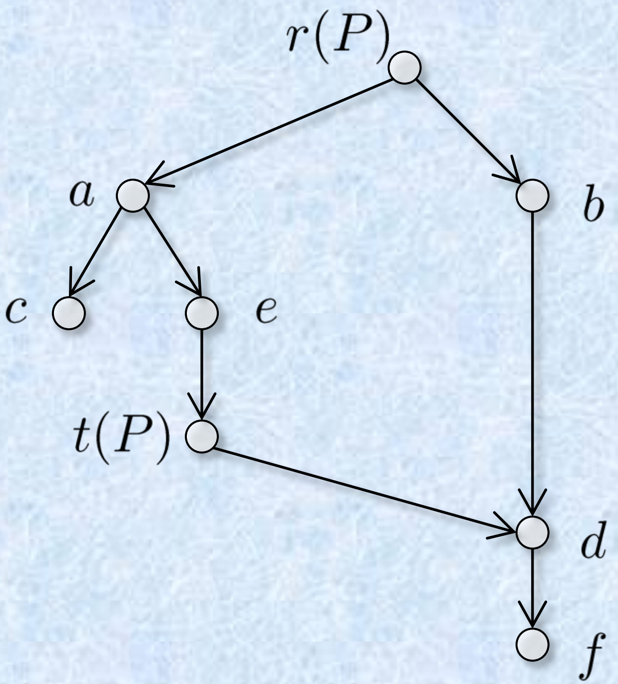


$D =$ transitive reduction of the dominance relation in G

Iterprocedural Flowgraphs



interprocedural flowgraph G



$D =$ transitive reduction of the dominance relation in G

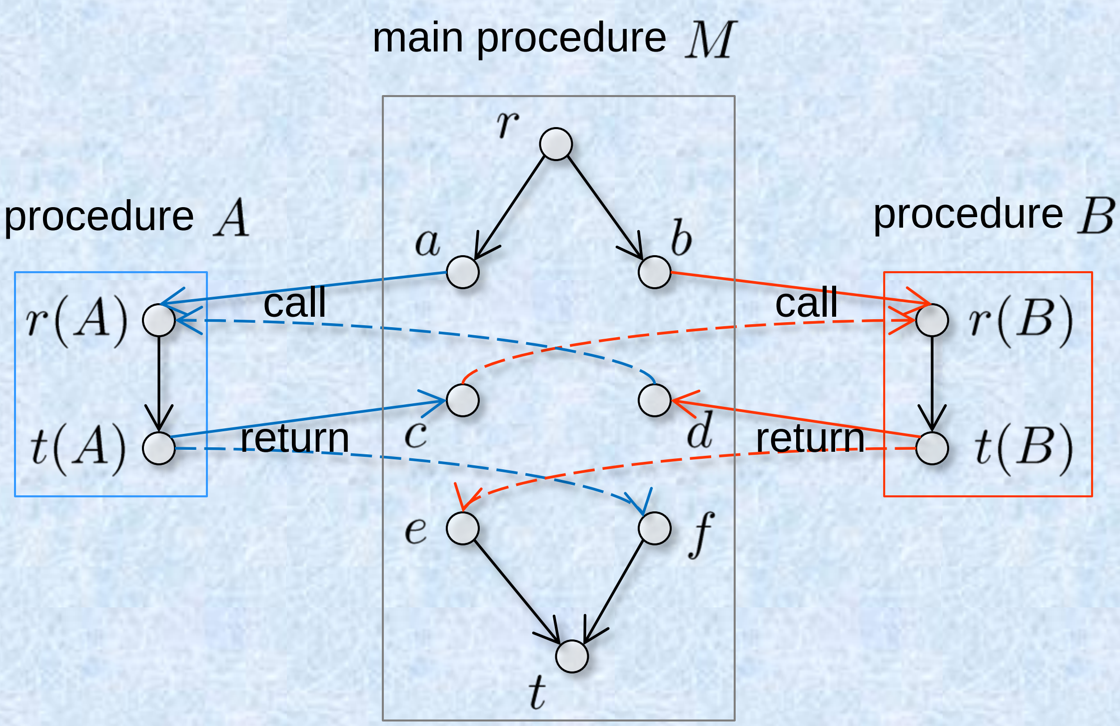
Iterprocedural Flowgraphs

Interprocedural flowgraph $G = (V, E, r, t)$

The vertex set V is partitioned into procedures $P \in \mathcal{P}$; $\mathcal{P} =$ set of procedures

Each procedure $P \in \mathcal{P}$ has unique entry $r(P)$ and exit $t(P)$

$M \in \mathcal{P} =$ main procedure that contains global start $r = r(M)$ and terminal $t = t(M)$



Iterprocedural Flowgraphs

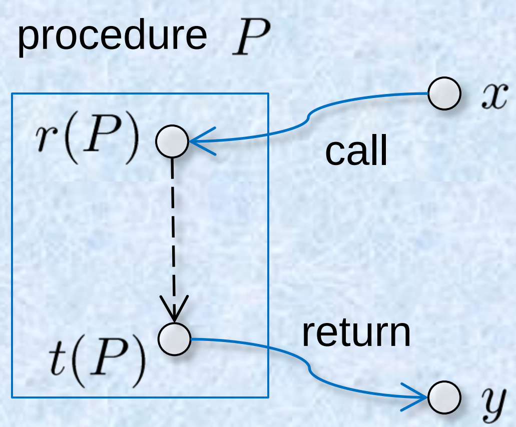
Interprocedural flowgraph $G = (V, E, r, t)$

The vertex set V is partitioned into procedures $P \in \mathcal{P}$; $\mathcal{P} =$ set of procedures

Each procedure $P \in \mathcal{P}$ has unique entry $r(P)$ and exit $t(P)$

$M \in \mathcal{P} =$ main procedure that contains global start $r = r(M)$ and terminal $t = t(M)$

Call/return1-1 correspondence ϕ

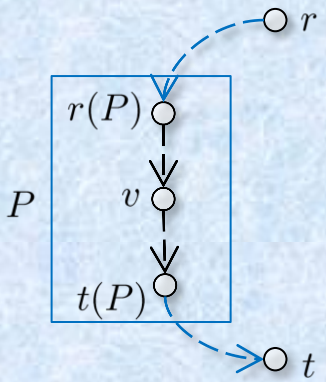


A call edge $(x, r(P))$ has a unique corresponding return edge $\phi((x, r(P))) = (t(P), y)$

$x =$ call node for P

$y =$ return node for P

Iterprocedural Flowgraphs



Interprocedural flowgraph $G = (V, E, r, t)$

Full path: $p = (p_1, p_2, \ldots, p_k)$ such that $p_1 = r$ and $p_k = t$

Valid full path: $p = (p_1 = r, p_2, \ldots, p_k = t)$ with proper nesting of call/return edges

- There is a 1-1 correspondence between the occurrences of call and return edges on p
- Each occurrence of a return edge e on p is preceded by the corresponding occurrence of $\phi^{-1}(e)$
- If there is an occurrence of a call edge e on p that precedes an occurrence of a call edge e' then either the corresponding occurrence of $\phi(e)$ precedes e' or the corresponding occurrence of $\phi(e')$ precedes the corresponding occurrence of $\phi(e)$

Iterprocedural Flowgraphs

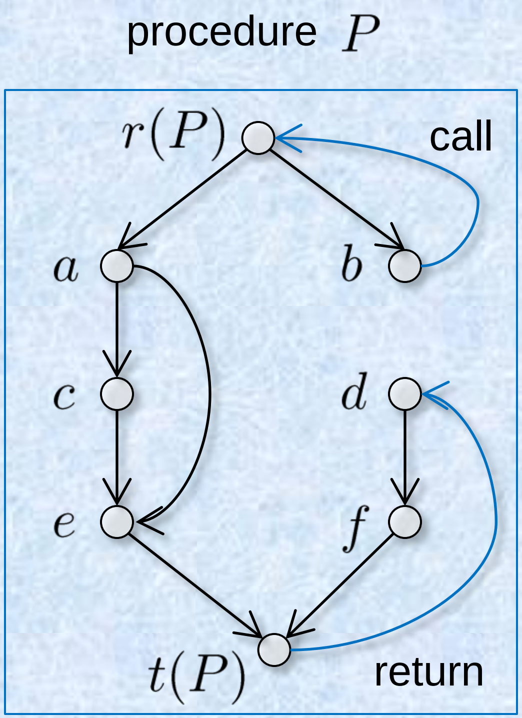
Interprocedural flowgraph $G = (V, E, r, t)$

Full path: $p = (p_1, p_2, \ldots, p_k)$ such that $p_1 = r$ and $p_k = t$

Valid full path: $p = (p_1 = r, p_2, \ldots, p_k = t)$ with proper nesting of call/return edges

Valid path: Prefix of a full valid path

Unlike the intraprocedural case we cannot consider simple paths only



Iterprocedural Flowgraphs

Interprocedural flowgraph $G = (V, E, r, t)$

Full path: $p = (p_1, p_2, \ldots, p_k)$ such that $p_1 = r$ and $p_k = t$

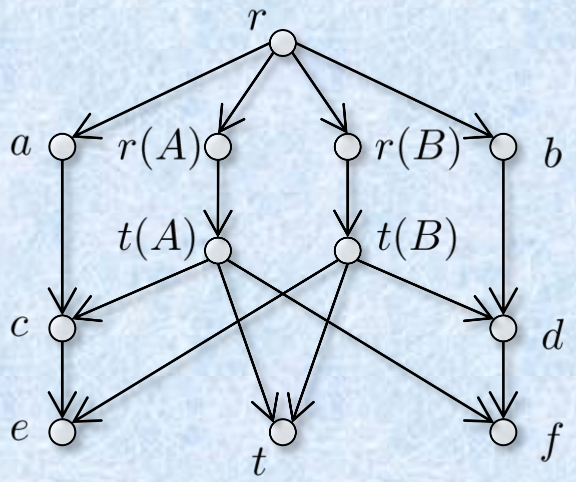
Valid full path: $p = (p_1 = r, p_2, \ldots, p_k = t)$ with proper nesting of call/return edges

Valid path: Prefix of a full valid path

Dominance: A vertex v dominates a vertex w if every valid path to w contains v

$D =$ transitive reduction of dominance relation in G

D is a directed acyclic graph; can have $\Omega(n\lambda)$ arcs, where $\lambda = |\mathcal{P}|$



Algorithms

Reachability algorithm: $O(mn)$ time, $O(n^2)$ space

- Reps, Horwitz and Sagiv (1995)
- Ezick, Bilardi and Pingali (2001)

Iterative algorithm: $O(mn^3)$ time, $O(n^2)$ space

- de Sutter, van Put and de Bosschere (2007)

New algorithm: $O(m\lambda + \lambda^\omega)$ time, $O(n\lambda)$ space

$\lambda = |\mathcal{P}|$, $\omega < 2.3727$ matrix multiplication exponent

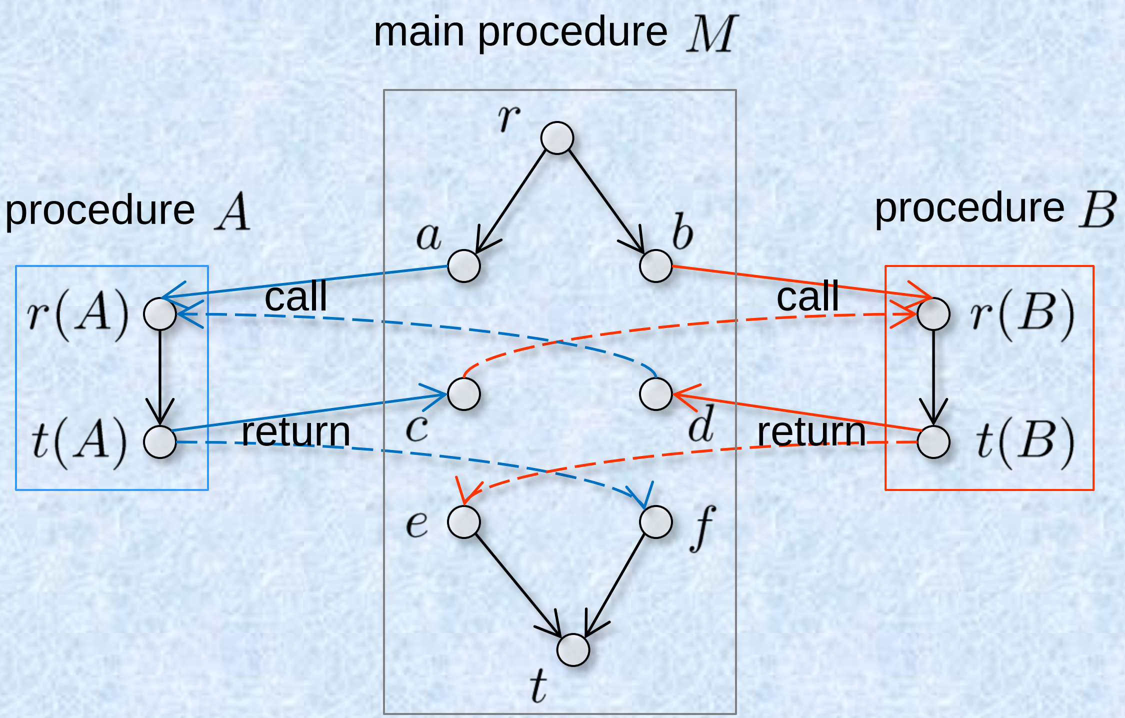
All the above support the following queries:

- Return the dominators of a query vertex v in $O(|Dom(v)|)$ time
- Given query vertices v and w test if v dominates w in $O(1)$ time

Context-Sensitive Reachability

Context-Sensitive Depth-First Search (CSDFS)
[de Sutter, van Put and de Bosschere, TOPLAS 2007]

Traverse a return edge e only if $\phi^{-1}(e)$ has already been processed

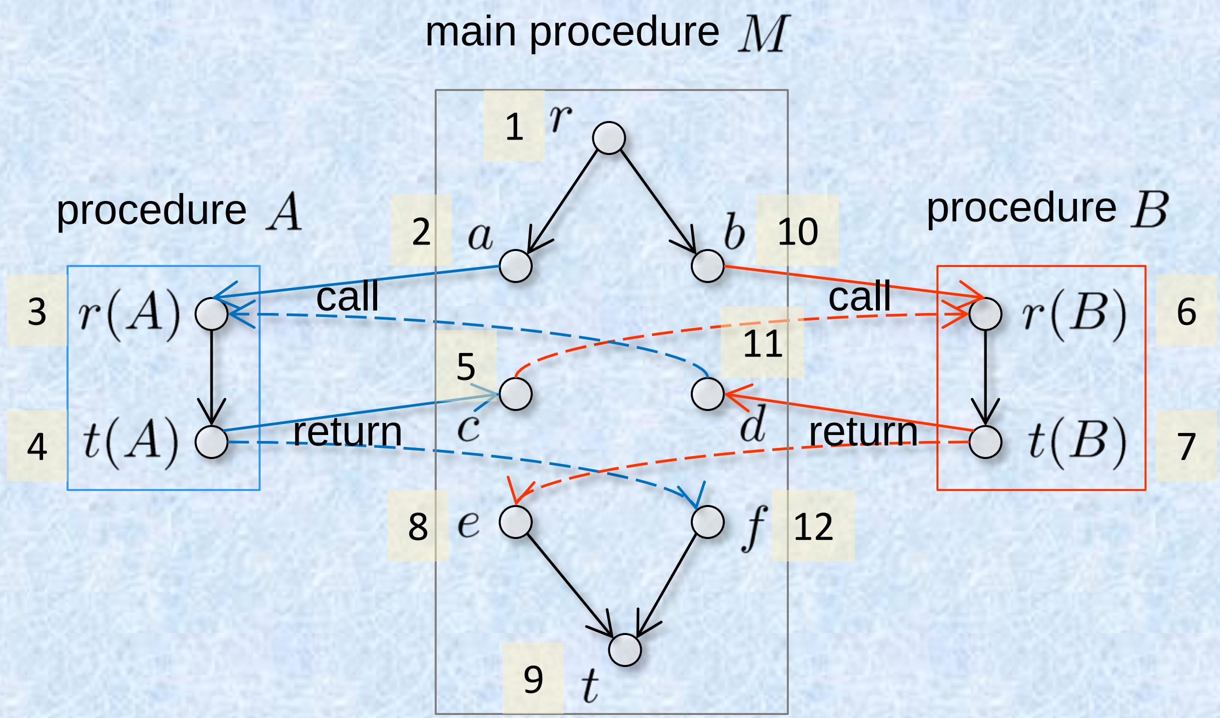


Context-Sensitive Reachability

Context-Sensitive Depth-First Search (CSDFS)
[de Sutter, van Put and de Bosschere, TOPLAS 2007]

Traverse a return edge e only if $\phi^{-1}(e)$ has already been processed

Running time $= O(m)$



Iterprocedural Flowgraphs

Interprocedural flowgraph $G = (V, E, r, t)$

Full path: $p = (p_1, p_2, \ldots, p_k)$ such that $p_1 = r$ and $p_k = t$

Valid full path: $p = (p_1 = r, p_2, \ldots, p_k = t)$ with proper nesting of call/return edges

Valid path: Prefix of a full valid path

Dominance: A vertex v dominates a vertex w if every valid path to w contains v

Valid subpath: Suffix of a valid path

Matched subpath: Valid subpath that has no unmatched occurrence of a call or a return edge

Overview of the new algorithm

Let v be a vertex in procedure P

Internal Dominators

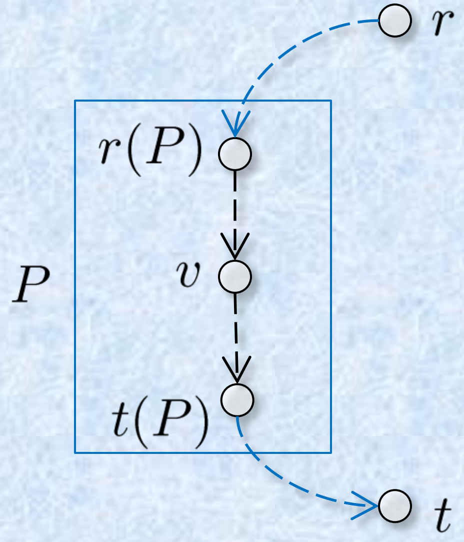
$IntDom(v) =$ vertices included in all matched subpaths from $r(P)$ to v

External Dominators

$ExtDom(v) =$ vertices included in all valid paths from r to $r(P)$

It can be $IntDom(v) \cap ExtDom(v) \neq \emptyset$

We have $Dom(v) = IntDom(v) \cup ExtDom(v)$



Computing Internal Dominators

Internal Post-Dominators

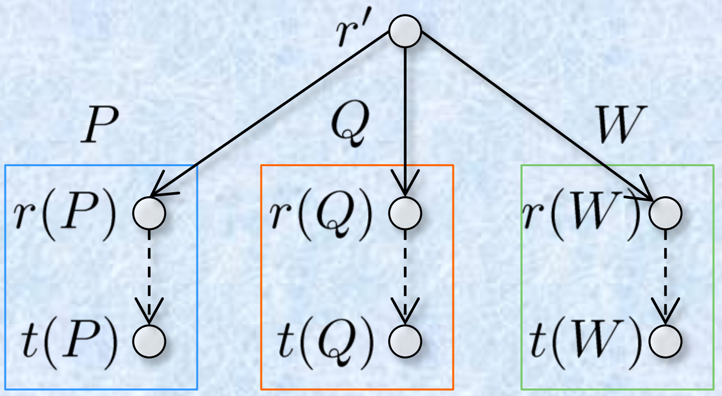
Consider procedures P and Q

Q is an internal post-dominator of P if every matched subpath from $r(P)$ to $t(P)$ contains $r(Q)$

Lemma

We can compute all internal post-dominators of all procedures in $O(m\lambda)$ time

Form a new graph G' from G with new start r' and edges $(r', r(P))$ for all $P \in \mathcal{P}$



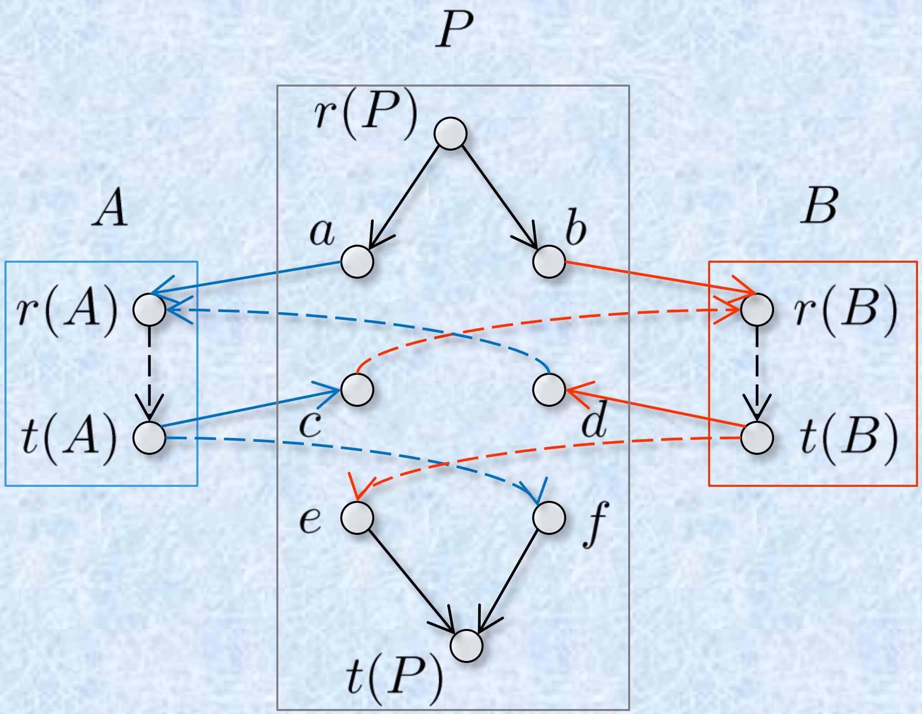
Q is an internal post-dominator of P if and only if $t(P)$ is unreachable in $G' - r(Q)$

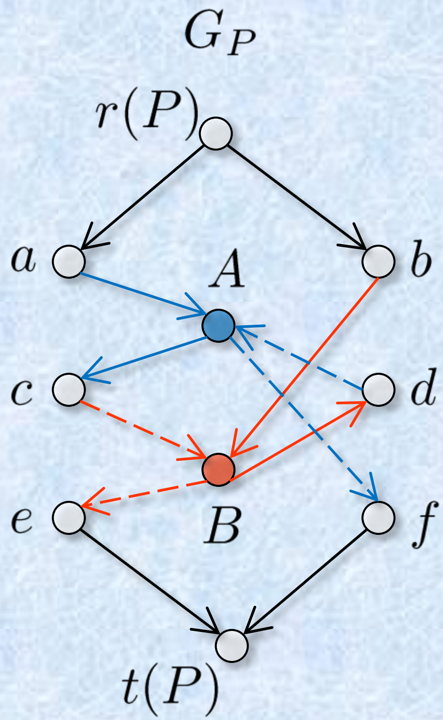
Computing Internal Dominators

Reduced flowgraphs

Graph $G_P = (V_P, E_P, r(P), t(P))$ for procedure P

$V_P = P \cup S$ where $S =$ special vertices, one for each procedure called from P





Computing Internal Dominators

Reduced flowgraphs

Graph $G_P = (V_P, E_P, r(P), t(P))$ for procedure P

$V_P = P \cup S$ where $S =$ special vertices, one for each procedure called from P

Each special vertex Q is assigned a set of labels $L(Q) \subseteq \mathcal{P}$

$W \in L(Q)$ if and only if W is an internal post-dominator of Q

Label-recursive procedure P

G_P contains a special vertex Q such that $P \in L(Q)$

Labels of vertex $v \in P$

$Labels(v) =$ set of labels that appear on every valid path from $r(P)$ to v

We can compute all vertex labels in $O(m\lambda)$ time and $O(n\lambda)$ space

Computing Internal Dominators

Reduced flowgraphs

Graph $G_P = (V_P, E_P, r(P), t(P))$ for procedure P

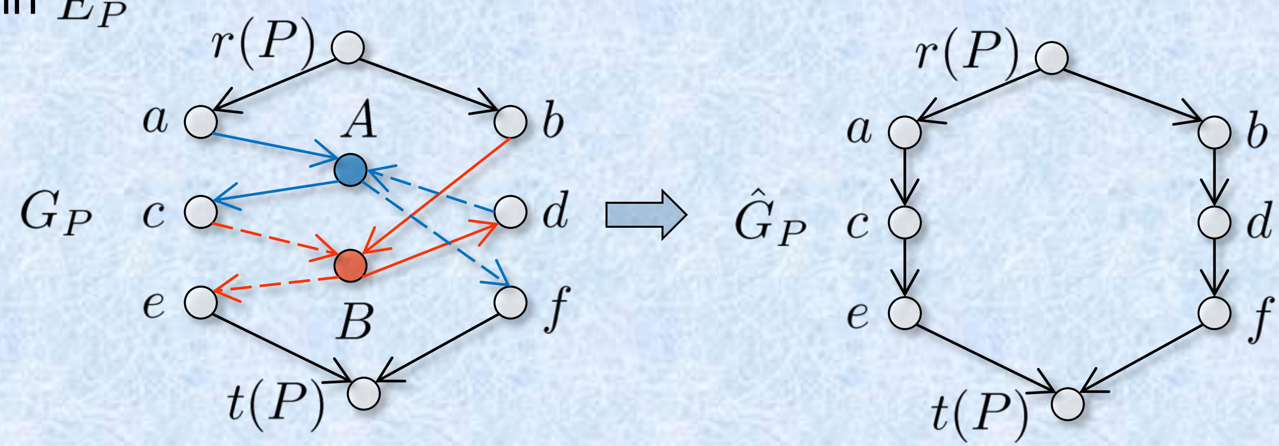
$V_P = P \cup S_P$, $S_P =$ special vertices, one for each procedure called from P

Auxiliary flowgraphs

Graph $\hat{G}_P = (\hat{V}_P, \hat{E}_P, r(P), t(P))$ is formed from G_P as follows

We remove all special vertices and their incident edges

For each call edge $e = (x, Q)$ and corresponding return edge $\phi(e) = (Q, y)$ in G_P we add (x, y) in $\hat{E}_P$



Computing Internal Dominators

Reduced flowgraphs

Graph $G_P = (V_P, E_P, r(P), t(P))$ for procedure P

$V_P = P \cup S_P$, $S_P =$ special vertices, one for each procedure called from P

Auxiliary flowgraphs

Graph $\hat{G}_P = (\hat{V}_P, \hat{E}_P, r(P), t(P))$ is formed from G_P as follows

We remove all special vertices and their incident edges

For each call edge $e = (x, Q)$ and corresponding return edge $\phi(e) = (Q, y)$ in G_P we add (x, y) in $\hat{E}_P$

Graph $\tilde{G}_P = (\tilde{V}_P, \tilde{E}_P, r(P), t(P))$ is formed from $\hat{G}_P$ as follows

For each call edge $e = (x, Q)$ and corresponding return edge $\phi(e) = (Q, y)$ in G_P such that $P \in L(Q)$ we remove (x, y) from $\tilde{E}_P$

Computing Internal Dominators

Non label-recursive procedure

Let $v \in P$ and $w \in Q$, where P is not label-recursive. Then $w \in IntDom(v)$ if and only if either

(a) $P = Q$ and $w \in Dom_{\hat{G}_P}(v)$, or

(b) $P \neq Q$, $Q \in Labels(v)$ and $w \in Dom_{G_Q}(t(Q)) \cap Q$

Computing Internal Dominators

Label-recursive procedure

Let P be a label-recursive procedure. Then

$$Dom_{G_P}(t(P)) \cap P = Dom_{\tilde{G}_P}(t(P))$$

Status of a vertex $v \in \tilde{G}_P$

- **unreachable** if there is no $r(P)$-v path in $\tilde{G}_P$
- **affected** if it is not unreachable but $Dom_{\hat{G}_P}(v) \neq Dom_{\tilde{G}_P}(v)$
- **unaffected** if $Dom_{\hat{G}_P}(v) = Dom_{\tilde{G}_P}(v)$

Computing Internal Dominators

Label-recursive procedure

Let $v \in P$ where P is label-recursive.

- If v is affected then

$$Dom_{G_P}(v) \cap P = Dom_{\hat{G}_P}(v) \cup \left(Dom_{\tilde{G}_P}(v) \cap Dom_{\tilde{G}_P}(t(P))\right)$$

- If v is unreachable then

$$Dom_{G_P}(v) \cap P = Dom_{\tilde{G}_P}(t(P)) \cup Dom_{\hat{G}_P}(v)$$

Computing External Dominators

Full and Partial Dominators

Consider procedures P and Q

- Q fully dominates P if $t(Q)$ dominates $r(P)$
- Q partially dominates P if $r(Q)$ dominates $r(P)$ but $t(Q)$ does not

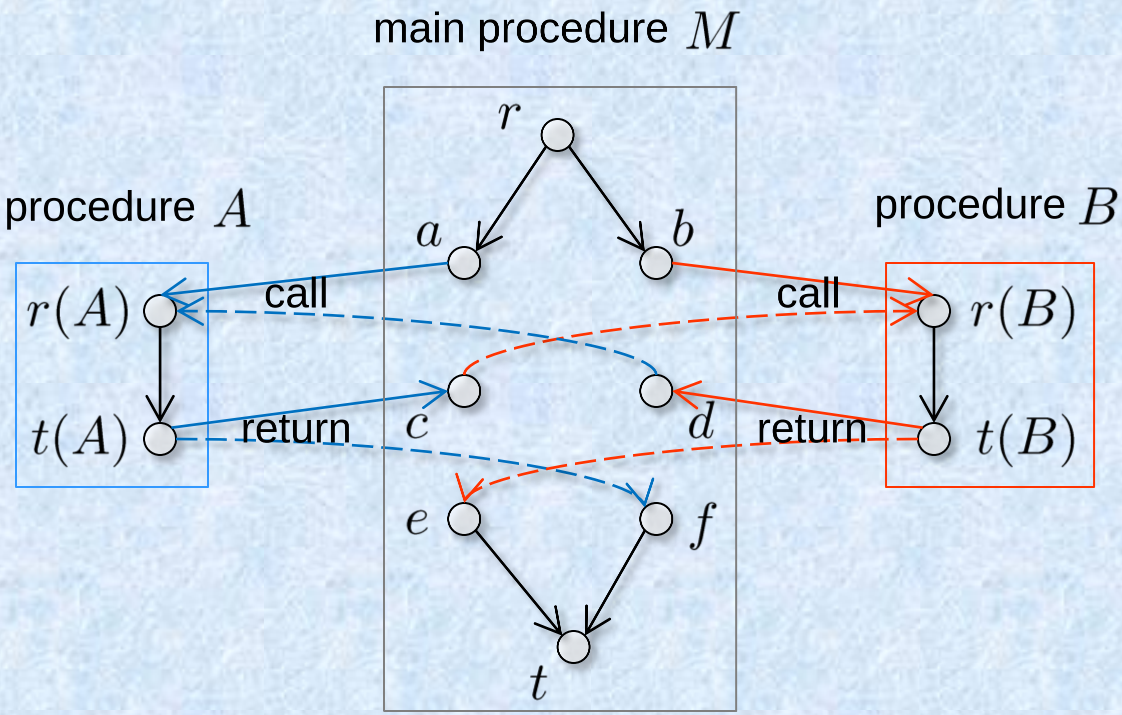
We can compute all full and partial dominators in $O(m\lambda)$ time using the reachability algorithm

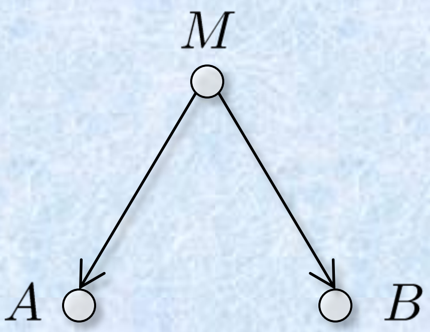
The challenge is to compute $Dom(r(P)) \cap Q$ when Q partially dominates P

Computing External Dominators

Call Graph $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$

$V_{\mathcal{C}} = \mathcal{P}, \quad (P, Q) \in E_{\mathcal{C}}$ if and only if Q is called from P





Computing External Dominators

Call Graph $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$

$V_{\mathcal{C}} = \mathcal{P},\ \ (X, Y) \in E_{\mathcal{C}}$ if and only if Y is called from X

We can use the transitive closure $\mathcal{C}^*$ of $\mathcal{C}$ to compute $Dom(r(P)) \cap Q$ when Q partially dominates P

Computing $\mathcal{C}^*$ takes $O(\lambda^{\omega})$ time

$\lambda = |\mathcal{P}|,\ \omega < 2.3727$ matrix multiplication exponent

Perspective

- Performance of new algorithm(s) in practice?
- Theory for structured programs?
  - Hecht and Ullman (1972): Structured programs (usually) have reducible intraprocedural control-flow graphs
  - Thorup (1998): Structured programs have intraprocedural control-flow graphs with small treewidth (typically <3)
  - Can we say something useful about interprocedural flowgraphs of structured programs?
- Other program optimization problems in intraprocedural flowgraphs?

Thank You!