Dominators in Graphs

Ming-Ho Yee

May 8, 2020

Definitions

d *dominates* n $\quad d \text{ dom } n, \quad d \in \text{Dom}(n)$

- If every path from s_0 (start node) to n contains d.
- Every node dominates itself

d *strictly dominates* n

- If d dominates n and $d \neq n$

d *immediately dominates* n $\quad d = \text{idom}(n)$

- If d strictly dominates n and every other dominator of n dominates d

Dominator tree

- Every child is immediately dominated by its parent
- Root node is the start node
- Ancestor a of node n → $a \text{ dom } n$

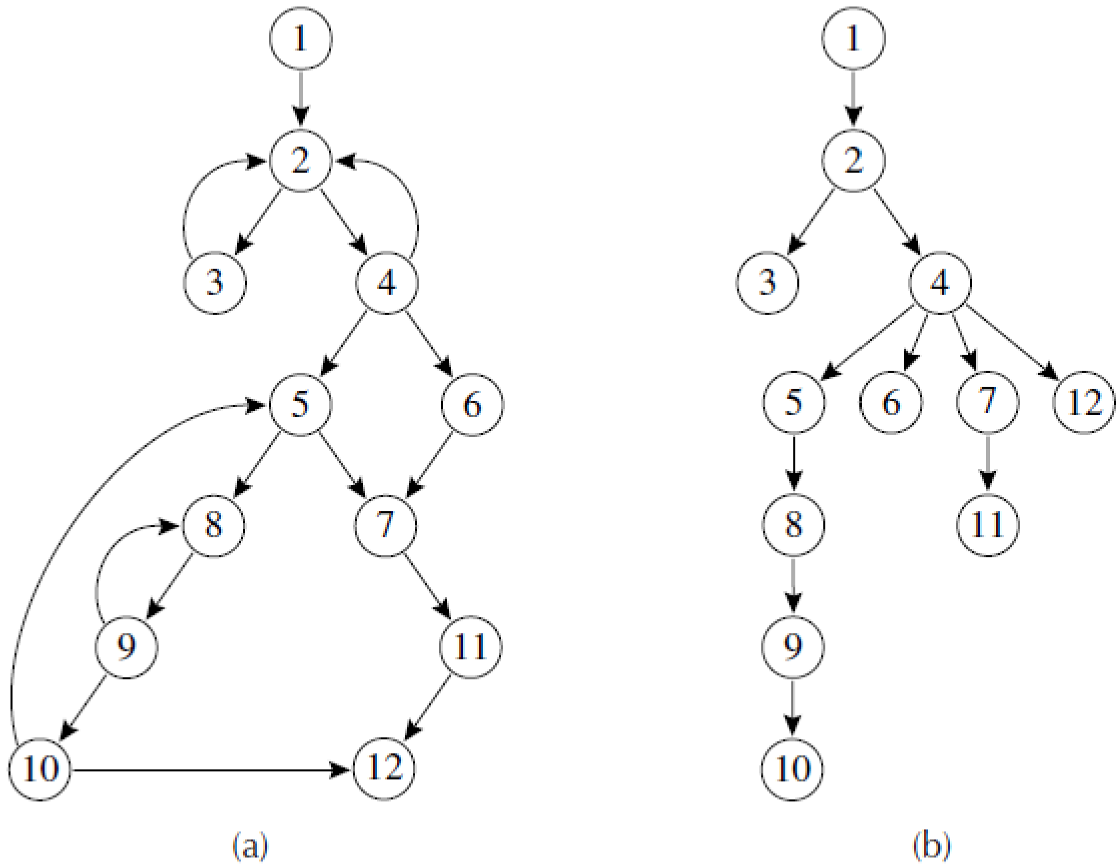


FIGURE 18.3. (a) A flow graph; (b) its dominator tree.

[Appel and Palsberg, 2004]

Applications

- Static Single Assignment
    - Compute the *dominance frontier* first
    - Use the frontier to insert phi nodes
- Finding loops
- Hoisting instructions

- And many other optimizations…

Anecdotal Performance in Ř

	Before		After	
	Total	*Average*	*Total*	*Average*
Benchmark (5 iterations)	89,205 ms	17,841 ms	82,425 ms	16,485 ms
Constructing the dominator tree	4,988 ms	80 µs	90 ms	1.5 µs
a dom? b	42 ms	46.1 ns	169 ms	182 ns
a idom? b	186 ms	24 ns	153 ms	19 ns

History

- 1959: Definition of *dominance* [Prosser 1959]
- 1969: First sketch of algorithm [Lowry and Medlock, 1969]
  - Complexity is at least quadratic
- 1970: Data-flow equations [Allen, 1970]
- 1972: Iterative data-flow algorithm [Allen and Cocke, 1972]

… more quadratic algorithms …

- 1979: Almost linear complexity [Lengauer and Tarjan, 1979]

… more almost linear algorithms …

Data-flow Equations

Let $Dom(n)$ be the set of nodes that dominate n. Then:

$$\mathrm{Dom}(s_0) = \{s_0\}$$

$$\mathrm{Dom}(n) = \{n\} \cup \left(\bigcap_{p \in \mathrm{preds}(n)} \mathrm{Dom}(p) \right)$$

Iterate until you reach a fixed point.

$O(n^2)$ time complexity, and very slow in practice

Another Algorithm [Aho and Ullman, 1972]

For each node $v \neq s_0$:

- Remove v from the graph
- Consider the set of nodes S that are now unreachable
- Then $\mathrm{Dom}(v) = S$

$O(n^2)$ time complexity

Iterative Algorithm, revisited

- Set intersection is the bottleneck
- Idea: use a consistent ordering for sets
  - Perform intersection by walking through both sets in order, with pairwise comparisons
  - Order encodes a path through the dominator tree

- Very simple implementation [Cooper, Harvey, and Kennedy, 2006]
- But slower in practice than the Lengauer-Tarjan algorithm [Georgiadis, Tarjan, and Werneck, 2006]

Lengauer-Tarjan Algorithm

- Simple version: $O(m \log n)$
- Sophisticated version: $O\big(m\, \alpha(m, n)\big)$
  - Where $\alpha(m, n)$ is the inverse of the Ackerman function

The simple Lengauer-Tarjan algorithm is faster in practice, and less sensitive to pathological graphs.

This explanation is adapted from Appel and Palsberg [2004].

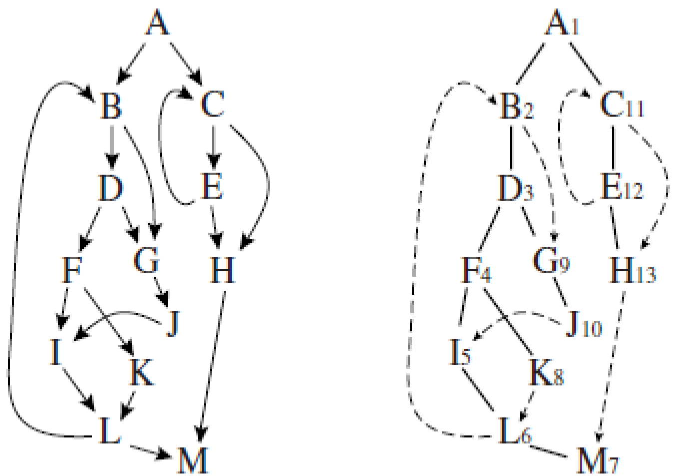
Depth-First Spanning Tree

- Use DFS to compute a spanning tree
  - Assign a *dfnum* to each node

a is an *ancestor* of b
- If $a = b$ or there is a path from a to b in the spanning tree
- *i.e.* $\mathrm{dfnum}(a) \leq \mathrm{dfnum}(b)$

a is a *proper ancestor* of b
- If a is an ancestor of b and $a \neq b$
- *i.e.* $\mathrm{dfnum}(a) < \mathrm{dfnum}(b)$

Note: Can test ancestor relation by comparing *dfnums*



(a) Graph

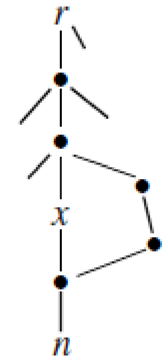
(b) Depth-first spanning tree

FIGURE 19.8. A control-flow graph and trees derived from it. The numeric labels in part (b) are the *dfnums* of the nodes.

[Appel and Palsberg, 2004]

Dominators and *dfnums*

- If $\mathrm{idom}(n) = d$, then d must be an ancestor of n
  - *i.e.*, $\mathrm{dfnum}(d) < \mathrm{dfnum}(n)$



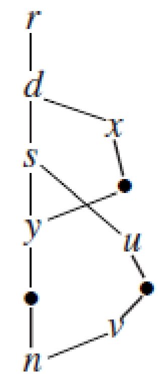
- Therefore: ancestors of n are idom candidates!

- If an ancestor x does not dominate n, there must be some “detour” starting above x.
  - Nodes on the detour are not ancestors of n
  - *i.e.* their *dfnums* must be greater than n’s

Semidominators

s semidominates n $s = \mathrm{semi}(n)$

- If s is the highest ancestor with a path to n, using non-ancestor nodes
  - Highest ancestor → smallest *dfnum*
    - $\mathrm{dfnum}(s) < \mathrm{dfnum}(n)$
  - Path $p = s, u_1, \ldots, u_k, n$ using non-ancestor nodes
    - $\mathrm{dfnum}(u_i) > \mathrm{dfnum}(n)$



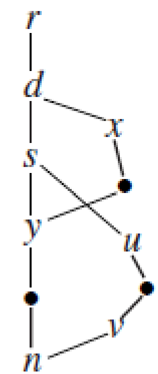
$\mathrm{semi}(n)$ is a candidate for $\mathrm{idom}(n)$

- Often, $\mathrm{semi}(n) = \mathrm{idom}(n)$
- An exception: $\mathrm{semi}(n)$ itself is bypassed

Semidominator Theorem

Consider all predecessors v of n in the CFG. Then:

- If v is a proper ancestor of n $\quad \mathrm{dfnum}(v) < \mathrm{dfnum}(n)$
  - v is a candidate for $\mathrm{semi}(n)$
- If v is a non-ancestor of n $\quad \mathrm{dfnum}(v) > \mathrm{dfnum}(n)$
  - For each u that is an ancestor of v, (and not an ancestor of n) $\mathrm{semi}(u)$ is a candidate for $\mathrm{semi}(n)$



$\mathrm{semi}(n)$ is the candidate with the lowest *dfnum*

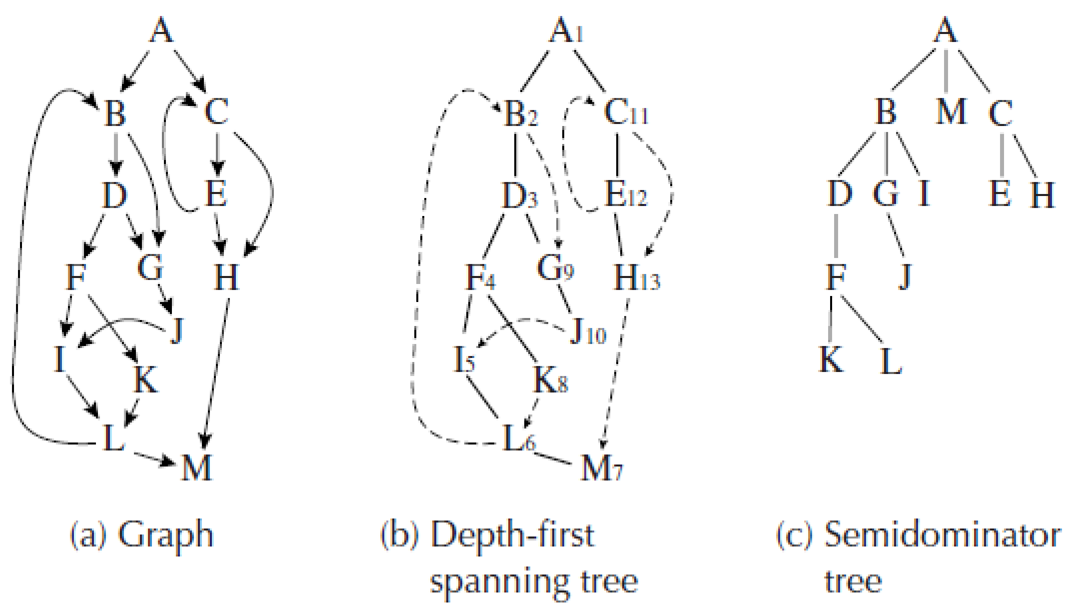


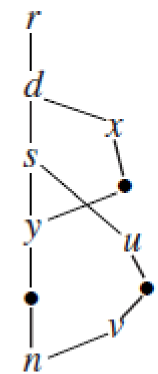
FIGURE 19.8. A control-flow graph and trees derived from it. The numeric labels in part (b) are the *dfnums* of the nodes.

[Appel and Palsberg, 2004]

Dominator Theorem

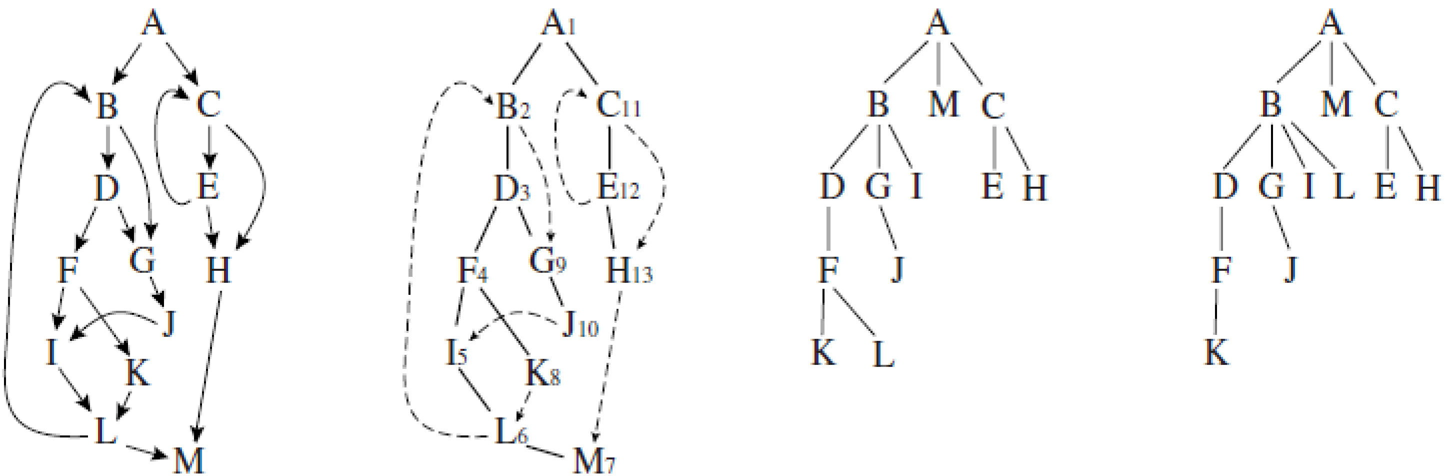
Consider the spanning tree path from $s = \mathrm{semi}(n)$ to n.

Let y be the node with smallest numbered semidominator, *i.e.* minimum $\mathrm{dfnum}(\mathrm{semi}(y))$.



Then:

$$\mathrm{idom}(n) = \begin{cases} \mathrm{semi}(n) & \text{if } \mathrm{semi}(y) = \mathrm{semi}(n) \\ \mathrm{idom}(y) & \text{if } \mathrm{semi}(y) \neq \mathrm{semi}(n) \end{cases}$$



(a) Graph

(b) Depth-first spanning tree

(c) Semidominator tree

(d) Dominator tree

FIGURE 19.8. A control-flow graph and trees derived from it. The numeric labels in part (b) are the *dfnums* of the nodes.

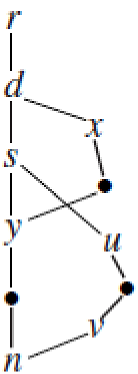
[Appel and Palsberg, 2004]

Lengauer-Tarjan Algorithm

1. Perform DFS to number nodes and create the depth-first spanning tree
2. For each node n (in decreasing *dfnum* order):
   - Use the Semidominator Theorem to compute $\mathrm{semi}(n)$
   - Insert n into the spanning forest
3. Implicitly define the idom by applying the first clause of the Dominator Theorem
4. For each node n (in increasing *dfnum* order):
   - Explicitly define the idom by applying the second clause of the Dominator Theorem

Spanning Forest

- Build a spanning forest as the CFG is traversed
  - When n is processed, only non-ancestors of n are in the forest



- `link(p, n)`
  - Add the edge (n, p) to the spanning forest
- `ancestorWithLowestSemi(v)`
  - Search upwards in the forest, starting from v
  - Find the ancestor of v whose semidominator has the lowest *dfnum*

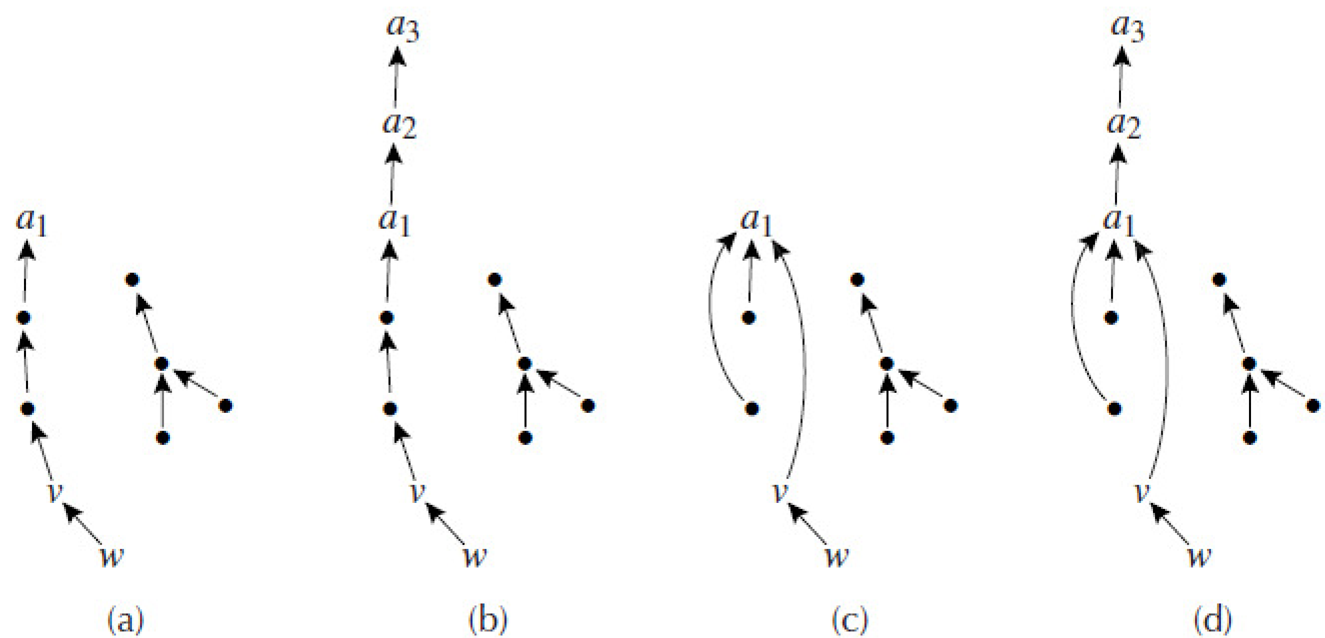


FIGURE 19.11. Path compression. (a) *Ancestor* links in a spanning tree; AncestorWithLowestSemi(v) traverses three links. (b) New nodes a_2, a_3 are linked into the tree. Now AncestorWithLowestSemi(w) would traverse 6 links. (c) AncestorWithLowestSemi(v) with path compression redirects ancestor links, but *best*[v] remembers the best intervening node on the compressed path between v and a_1. (d) Now, after a_2 and a_3 are linked, AncestorWithLowestSemi(w) traverses only 4 links.

[Appel and Palsberg, 2004]

```
Dominators() =
  N ← 0;  ∀n. bucket[n] ← {}
  ∀n. dfnum[n] ← 0,  semi[n] ← ancestor[n] ← idom[n] ← samedom[n] ← none
  DFS(none, r)
  for i ← N − 1 downto 1
    n ← vertex[i];   p ← parent[n];   s ← p
    for each predecessor v of n
      if dfnum[v] ≤ dfnum[n]
        s′ ← v
      else s′ ← semi[AncestorWithLowestSemi(v)]
      if dfnum[s′] < dfnum[s]
        s ← s′
    semi[n] ← s
    bucket[s] ← bucket[s] ∪ {n}
    Link(p, n)
    for each v in bucket[p]
      y ← AncestorWithLowestSemi(v)
      if semi[y] = semi[v]
        idom[v] ← p
      else samedom[v] ← y
    bucket[p] ← {}
  for i ← 1 to N − 1
    n ← vertex[i]
    if samedom[n] ≠ none
      idom[n] ← idom[samedom[n]]
```

```
DFS(node p, node n) =
  if dfnum[n] = 0
    dfnum[n] ← N;   vertex[N] ← n;   parent[n] ← p
    N ← N + 1
    for each successor w of n
      DFS(n, w)
```

```
AncestorWithLowestSemi(node v) =
  a ← ancestor[v]
  if ancestor[a] ≠ none
    b ← AncestorWithLowestSemi(a)
    ancestor[v] ← ancestor[a]
    if dfnum[semi[b]] <
         dfnum[semi[best[v]]]
      best[v] ← b
  return best[v]
```

```
Link(node p, node n) =
  ancestor[n] ← p;   best[n] ← n
```

[Appel and Palsberg, 2004]

Implementation in Ř

Straightforward translation from pseudocode to C++

- About 100 LOC, without comments

https://github.com/reactorlabs/rir/blob/b265f9e/rir/src/compiler/util/cfg.cpp#L52