Optimizing Compilers
3rd Lecture

Bernhard Scholz
Institut f. Computersprachen
Argentinierstr. 8
scholz@complang.tuwien.ac.at

Overview

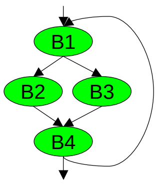
- Domination Relation
- Back Edges
- Loops
- Loop Transformation

Loop Optimizations

- Why loops are so important?
  - Typically programs spend most of their execution time inside loops.
- Basic Idea:
  - Improve performance of inner loops
  - E.g., moving invariant computations outside of loops, restructuring loops to eliminate cycles
- How can we detect loops?

What is a Loop?

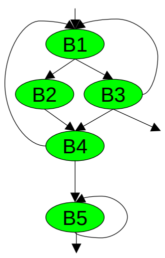
- Goal
  - Graph theoretical notion of loops
  - Insensitive to syntactical constructs, e.g. do/while, if/goto and uniform approach
- Intuition behind loops
  - Single entry point
  - There must be a cycle

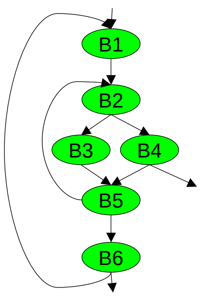


What is a Loop? (cont'd)

- A loop is a set of control flow nodes with an distinctive header node such that:
  - For any node in the loop there is a path to the header node.
  - Every node in the loop can be reached by the header node.
  - Exists a path from a node outside the loop to a node inside the loop, then the path contains the loop header.
- Loop exit nodes: Some successor nodes of loop nodes

Examples of Loops



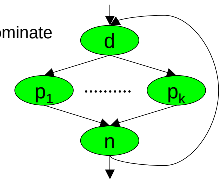


How to Identify Loops?

- Restrict loops to natural loops
- For several concepts required
  - domination relation: a node, i.e. dominator, dominates another node n if every path from the start node to n goes through the dominator.
  - immediate dominator: there is a unique dominator (if there exist one) for a node that does not dominate any other dominator of n.
  - dominator tree: immediate dominators form a dominator tree.
  - back edges: an edge whose head dominates its tail.
  - loop headers: entry nodes of natural loops
  - loop nodes: all nodes of a loop
  - loop forests: loop nests, e.g. loop in loops.

Domination Relation

- x dominates y (x *dom* y):
  - x is on every path from start node *start* to y
  - reflexive, transitive, anti-symmetric
- Observation:
  - If d dominates every predecessor p_i of n then d must dominate n.
  - If d dominates n then d must dominate all predecessors p_i of n.
- Proof: by contradiction



Algo for Domination Relation

- Approach
  - iterative approach
  - definition of dominators

    $dom(y)=\{x | x \; dom \; y\}$
  - local equations

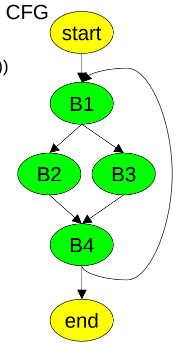
    $$dom(y) = \{y\} \cup \bigcap_{p \in preds(y)} dom(p)$$
- Algorithm

```
for all n∈N: dom(n)=N;
dom(start) = {start};

repeat
 for all n∈N-{start} do
  X =dom(p1)∩dom(p2)... where
        preds(n)={p1,p2,...};
  dom(n) = X ∪ {n}
 end for
until no changes in dom;
```

Example

- Equations
  1. dom(start) = {start}
  2. dom(B1) = {B1} ∪ (dom(start) ∩ dom(B4))
  3. dom(B2) = {B2} ∪ dom(B1)
  4. dom(B3) = {B3} ∪ dom(B1)
  5. dom(B4) = {B4} ∪ (dom(B2) ∩ dom(B3))
  6. dom(end) = {end} ∪ dom(B4)
- Solution
  1. dom(start) = {start}
  2. dom(B1) = {start,B1}
  3. dom(B2) = {start,B1,B2}
  4. dom(B3) = {start,B1,B3}
  5. dom(B4) = {start,B1,B4}
  6. dom(end) = {start,B1,B4,end}
- CFG



Dominator Tree

- Domination Relation
  - expensive data structure, i.e. O(N^2)
  - compressed as tree structure
- Immediate Dominator
  - *idom*(*y*) dominates *y*
  - no other dominator that dominates *y* and is dominated by *idom*(*y*)
  - only one immediate dominator of *y* (unique)
- Dominator Tree
  - nodes are control flow graph nodes
  - edges are given by (*idom*(*y*),*y*)

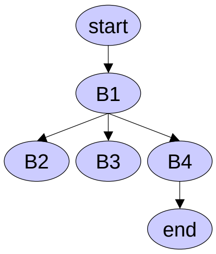
Algo for Immediate Dominator

- Approach
  - iterative approach
  - based on *dom*(*n*)
  - removes all non-immediate dominators
  - compute set *s* for node *n*:
    1. $s = dom(n) - \{v\}$
    2. for all: u *dom* n, v *dom* n, $u \neq v$, v *dom* $u \Rightarrow s = s - \{v\}$
  - set *s* only contains immediate dominator
- Algorithm

```
for all n ∈ N-{start}
 s = dom(n) – {n}
 for all u ∈ dom(n)-{n}
  for all v ∈ dom(n)–{n,u}
   if v ∈ dom(u) then
    s = s – {v}
   fi
  end for
 end for
 idom(n) = <s>
end for
```

Dominator Tree Example

- Domination Relation
  1. dom(start) = {start}
  2. dom(B1) = {start,B1}
  3. dom(B2) = {start,B1,B2}
  4. dom(B3) = {start,B1,B3}
  5. dom(B4) = {start,B1,B4}
  6. dom(end) = {start,B1,B4,end}
- Immediate Dominators
  1. idom(B1)=start
  2. idom(B2)=B1
  3. idom(B3)=B1
  4. idom(B4)=B1
  5. idom(end)=B4
- Dominator Tree

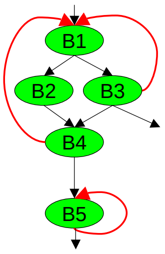


Dominator Algorithms

- [Purdom and Moore, 1972]
  – O(NxE) execution time
- [Lengauer and Tarjan, 1979]
  – simple version: O(E x logN) execution time
  – improved version: O (Exα(E,N)) execution time
- [Alstrup et al., 1997]
  – O(N+E) execution time

Back Edges

- Definition: A back edge is an edge whose tail dominates its source, i.e. an edge (n,d) where d *dom* n.
- Set of back edges
  $B = \{(n,d) \mid d \text{ dom } n\}$
- Example
  $B = \{(B3,B1),(B4,B1),(B5,B5)\}$

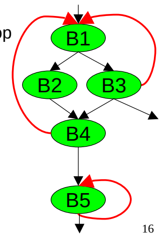


Natural Loops & Loop Headers

- Def: The natural loop of back edge (n,d) is the set of nodes where there exists a path to n without going through d.
- Def: A loop header dominates all nodes in a loop.
  - Header is unique for each loop
  - Header is the unique entry point for a loop
- Set of loop-headers

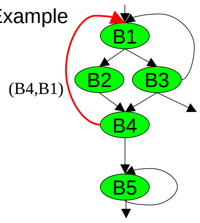
  $L = \{d \mid (n,d) \in B\}$
- Example:

  $L = \{B1,B5\}$



Algorithm for Detecting Loops

- Approach
  - simple work-list algorithm
  - L is set of nodes inside loop.
  - loop is formed by back-edge (n,d) where d is loop-header.
- Example



- Algorithm

```
W = {n}
L = {d}
repeat
 select u ∈ W
 L = L ∪ {u}
 W = ( W ∪ pred(u)) - L;
until W = 0;
```

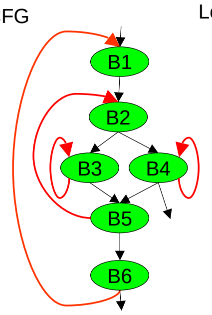
- Steps
  1. W={B4}, L={B1}
  2. W={B2,B3}, L={B1,B4}
  3. W={B3}, L={B1,B2,B4}
  4. W={}, L={B1,B2,B3,B4}

Inner Loops & Loop Forest

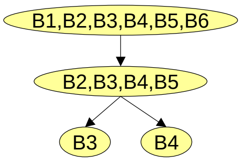
- If two loops do not have the same header
  - they are either disjoint, or
  - one is entirely contained (nested within) the other, or
- If two loops share the same header
  - difficulties to state which is the inner one
  - combine both loops (see example)
- Loop Nesting forest
  - gives a nesting relation for loops
  - the ancestor of a loop gives the nesting
  - If $L_1 \subseteq L_2$, then loop L_1 is nested in L_2.
  - nodes: loops of CFG
  - edges: immediate nesting relation

Example

CFG



Loop Nesting Forest



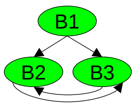
Reducible Flow Graphs

- Def. A flow graph is called reducible iff we can partition the edges into 2 sets:
  1. back ward edges, i.e. (n,d) where d dominates n.
  2. forward edges: should form a DAG in which every node is reachable from start node.

- A reducible graph has only natural loops
- Every "cycle" has at least one back edge

What's wrong with natural loops?

- Irreducible CFGs
  - CFG has loops that are not "natural".
  - i.e. more than one loop entry



- Approach for irreducible CFGs:
  - Tarjan's interval analysis
  - Drawback: not as intuitive as natural loops

Loop Transformations

- Loop Inversion

```
for(i=0;i<100;i++){
   a[i] = 2*b[i];
}
```

```
i=0;
do{
  a[i] = 2 * b[i];
  i++;
} while (i<100);
```

  – eliminate goto at the end

- Loop Unrolling

```
for(i=0;i<100;i++){
   a[i] = 2*b[i];
}
```

```
for(i=0;i<100;i+=2){
   a[i] = 2*b[i];
   a[i+1] = 2*b[i+1];
}
```

  – eliminate gotos for several iterations

Loop Transformations(2)

- Loop Peeling

```
for(i=0;i<100;i++){
   if (i<1) {
     a[i] = 2*b[i];
   } else {
     a[i] = b[i]-1;
   }
}
```

```
a[0] = 2*b[0];
for(i=1;i<100;i++){
  a[i] = b[i]-1;
}
```

- remove iteration anomalies at the begin and end

Loop Transformations(3)

- Loop Fusion / Loop Jamming

```
for(i=0;i<100;i++){
   a[i] = 2*b[i];
}
for(i=0;i<100;i++){
   c[i] = d[i]-1;
}
```

```
for(i=0;i<100;i++){
   a[i] = 2*b[i];
   c[i] = d[i]-1;
}
```

- dependence analysis required!

Stop

- Next lecture: 10.4.2003, 13:45 – 14:45
- 2nd Assignment!