**Dependence Analysis**

Important for
- Optimization
  - Instruction scheduling
  - Data-cache optimization
- Software engineering
  - Program understanding
  - Reverse engineering
  - Debugging

Various kinds of dependences
- Data related
- Control related

**Data-Dependence Graph**

A data-dependence graph has one node for every variable (basic block) and one edge representing the flow of data between the two nodes.

Different types of data dependence
- Flow: def to use
- Anti: use to def
- Output: def to def
- Input: use to use

Further classifiable as
- Loop-carried: requires iterations
- Loop-independent: occurs anyway
Control-Dependence Analysis

Intuition
- A statement S1 is control dependent on a statement S2 if the outcome of S2 directly determines whether S1 may not be reached.
- What are the control dependences for each statement in the CFG on the right?

Data-Dependence Graph

Z = X - 3
X
Z > 1
X = 4

B4

Z
Z
X = 2
X
Z
Z > 2
X = 1
Z

B2

entry

X = 1

B1

Y = X + 1

B3

Z

exit

X

B5

Y

B6

Z

X

B2

entry

Z

X = 2

B4

X

Z

X = 3

Z

X = 4

B4

X

Z

exit
Control-Dependence Analysis

Intuition

- A statement S1 is control dependent on a statement S2 if the outcome of S2 directly determines whether S1 may not be reached.
- What are the control dependences for each statement in the CFG on the right?
  - entry, B1, exit - entering code
  - B2, B1T
  - B3, B2T
  - B4, B1F
  - B5, B2F, B1F
  - B6, B2F, B1F

Control-Dependence Analysis

Definition

Let G be a CFG, with X and Y nodes in G. Y is control-dependent on X iff

1. There exists a path P from X to Y with any Z in P (excluding X and Y) postdominated by Y and
2. X is not postdominated by Y

(there are two edges out of X; traversing one edges always leads to Y, the other may not lead to Y)
Computing CD Using FOW

1. Construct basic CDG
2. Add region nodes

Augment the CFG by adding a node
Start with edge (Start, entry)
labeled “T” and edge (Start, exit)
labeled “F”; call this AugCFG
Computing CD Using FOW

Construct the postdominator tree for AugCFG

AugCFG

En

T

F

F

T

3

5

4

6

Start

Ex

1

En

Computing CD Using FOW

Construct the postdominator tree for AugCFG

Pdom Tree

Ex

Start

2

3

6

1

En

4
Computing CD Using FOW

Consider set $S$ of edges $(m, n)$ in AugCFG such that $n$ does not postdominate $m$.

For AugCFG $S$ consists of:

1. $(1, 2)$
2. $(1, 3)$
3. $(1, 4)$
4. $(2, 3)$
5. $(2, 5)$
6. $(3, 3)$

Consider set $S$ of edges $(m, n)$ in AugCFG such that $n$ does not postdominate $m$.

For AugCFG $S$ consists of:

- $(\text{Start}, \text{En})$
- $(1,2)$
- $(1,4)$
- $(2,3)$
- $(2,5)$
Computing CD Using FOW

Consider, for each edge \((A, B)\) in \(S\), those nodes in the Pdom tree from \(B\) to least common ancestor \(L\) of \(A\) and \(B\)

- Including \(L\) if \(L\) is \(A\)
- Excluding \(L\) if \(L\) is not \(A\)

<table>
<thead>
<tr>
<th>Edge</th>
<th>(L)</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 4)</td>
<td>Ex</td>
<td>(E_n, 1)</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>Ex</td>
<td>2</td>
</tr>
<tr>
<td>(1, 4)</td>
<td>Ex</td>
<td>(4, 5, 6)</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>Ex</td>
<td>3</td>
</tr>
<tr>
<td>(2, 5)</td>
<td>Ex</td>
<td>5, 6</td>
</tr>
</tbody>
</table>
**Computing CD Using FOW**

Consider, for each edge \((A,B)\) in \(S\), those nodes in the Pdom tree from \(B\) to least common ancestor \(L\) of \(A\) and \(B\):
- Including \(L\) if \(L\) is \(A\)
- Excluding \(L\) if \(L\) is not \(A\)

<table>
<thead>
<tr>
<th>Edge</th>
<th>(L)</th>
<th>Nodes</th>
<th>CD on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start, En</td>
<td>Ex</td>
<td>En, 1</td>
<td>Start, T</td>
</tr>
<tr>
<td>1, 2</td>
<td>Ex</td>
<td>2</td>
<td>1, T</td>
</tr>
<tr>
<td>1, 4</td>
<td>Ex</td>
<td>4, 5, 6</td>
<td>1, F</td>
</tr>
<tr>
<td>2, 3</td>
<td>Ex</td>
<td>3</td>
<td>2, T</td>
</tr>
<tr>
<td>2, 5</td>
<td>Ex</td>
<td>5, 6</td>
<td>2, F</td>
</tr>
</tbody>
</table>

All identified nodes are control dependent on \(A\)

---

**Computing CD Using FOW**

Consider, for each edge \((A,B)\) in \(S\), those nodes in the Pdom tree from \(B\) to least common ancestor \(L\) of \(A\) and \(B\):
- Including \(L\) if \(L\) is \(A\)
- Excluding \(L\) if \(L\) is not \(A\)

<table>
<thead>
<tr>
<th>Edge</th>
<th>(L)</th>
<th>Nodes</th>
<th>CD on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start, En</td>
<td>Ex</td>
<td>En, 1</td>
<td>Start, T</td>
</tr>
<tr>
<td>1, 2</td>
<td>Ex</td>
<td>2</td>
<td>1, T</td>
</tr>
<tr>
<td>1, 4</td>
<td>Ex</td>
<td>4, 5, 6</td>
<td>1, F</td>
</tr>
<tr>
<td>2, 3</td>
<td>Ex</td>
<td>3</td>
<td>2, T</td>
</tr>
<tr>
<td>2, 5</td>
<td>Ex</td>
<td>5, 6</td>
<td>2, F</td>
</tr>
</tbody>
</table>

All identified nodes are control dependent on \(A\)
Computing CD Using FOW

Given (A,B), how can you find the set of nodes control dependent on A easily?
Why is the set we obtain correct?

<table>
<thead>
<tr>
<th>Edge</th>
<th>L</th>
<th>Nodes</th>
<th>CD on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start, En</td>
<td>Ex</td>
<td>En, 1</td>
<td>Start, T</td>
</tr>
<tr>
<td>1, 2</td>
<td>Ex</td>
<td>2</td>
<td>1, T</td>
</tr>
<tr>
<td>1, 4</td>
<td>Ex</td>
<td>4, 5, 6</td>
<td>1, F</td>
</tr>
<tr>
<td>2, 3</td>
<td>Ex</td>
<td>3</td>
<td>2, T</td>
</tr>
<tr>
<td>2, 5</td>
<td>Ex</td>
<td>5, 6</td>
<td>2, F</td>
</tr>
</tbody>
</table>

Create CDG
Computing CD Using FOW

1. Construct basic CDG
2. Add region nodes

Computing CD Using FOW

Why adding regions?
- Group nodes with same set of control conditions
- Only two successors for predicate nodes (=> hierarchical organization)
Computing CD Using FOW

Method for adding regions:
• Consider set CD of control dependence predecessors of each non-region node with more than one unlabeled control-dependence predecessor
• Create region node R for each CD and move edges accordingly
  • Nodes with control dependence CD connected to R
  • R connected to CD

Add region nodes to CDG
Computing CD Using FOW

Add region nodes to CDG

Creating new regions if necessary
Computing CD Using FOW

Create new regions if necessary

Merge region nodes if possible
Program-Dependence Graph (PDG)

A program dependence graph (PDG) for a program P is the combination of the control-dependence graph for P and the data-dependence graph for P.

A PDG contains nodes representing statements in P, edges representing control dependence between nodes, and edges representing data dependence between nodes.

Program-Dependence Graph (PDG)

1. read (n)
2. if (n < 0)
3. n = 0
4. i := 1
5. sum := 0
6. product := 1
7. while i <= n do
8. sum := sum + i
9. product := product * i
10. i := i + 1
11. write (sum)
12. write (product)

Compute the PDG for the program on the left.
Program-Dependence Graph (PDG)

1. read a
2. if (a <= 0) {
3.     print a
4.     return
5. }
6. i = a
7. while (i > 0) {
8.     a = a + i
9.     i = i - 1
10. }
11. print a

Compute the PDG for the program on the left.