Control Flow Analysis

Tim Blazytko

@mr_phrazer

tim@blazytko.to

https://synthesis.to

Why?

Motivation

- **high-level** structure of a function
- detect branches and **loops**
- **pattern matching** to spot interesting code parts
- foundation for **automated** program analysis

Basic Block

Basic Block

- sequence of **ordered** instructions

Basic Block

- sequence of **ordered** instructions
- **single entry**: only first instruction can be target of a branch

Basic Block

- sequence of **ordered** instructions
- **single entry**: only first instruction can be target of a branch
- **single exit**: only last instruction can branch to other basic blocks

Basic Block Identification

Rules: Leader Instruction Identification

1. first instruction is a leader
2. target of a control flow transfer is a leader
3. instruction that immediately follows a control flow transfer is a leader

Split on calls?

- strict basic block definition: yes
    - calls interrupt the control flow

Split on calls?

- strict basic block definition: yes
    - calls interrupt the control flow
- many tools handle it differently due to readability reasons
    - most calls return to the next instruction

Split on calls?

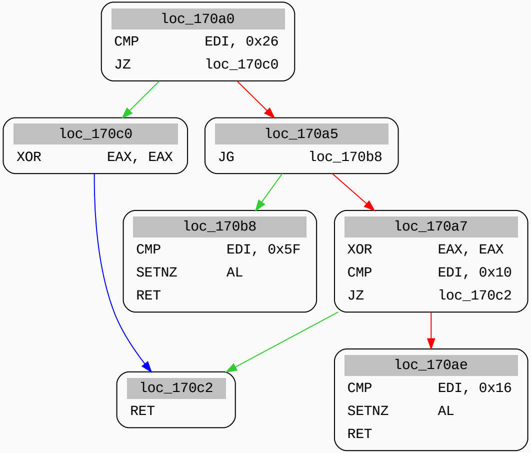
- strict basic block definition: yes
  - calls interrupt the control flow
- many tools handle it differently due to readability reasons
  - most calls return to the next instruction

Know how your tool handles it.

Basic Blocks

```
; leader: first instruction
0x170A0: cmp edi, 26h
0x170A3: jz  short 0x170C0

; leader: follows a control flow transfer
0x170A5: jg  short 0x170B8

; leader: follows a control flow transfer
0x170A7: xor eax, eax
0x170A9: cmp edi, 10h
0x170AC: jz  short 0x170C2

; leader: follows a control flow transfer
0x170AE: cmp edi, 16h
0x170B1: setnz al
0x170B4: retn

; leader: target of control flow transfer
0x170B8: cmp edi, 5Fh
0x170BB: setnz al
0x170BE: retn

; leader: target of control flow transfer
0x170C0: xor eax, eax

; leader: target of control flow transfer
0x170C2: retn
```

Control Flow Graph



Control Flow Graph

Control Flow Graph

- directed multigraph
- *nodes* are basic blocks
- *edges* represent control flow between basic blocks
- represents **all program paths** that might be traversed

Control Flow Graph

Entry
A node that has no incoming edges.

Control Flow Graph

Entry

A node that has no incoming edges.

Exit

A node that has no outgoing edges.

Control Flow Graph

Entry

A node that has no incoming edges.

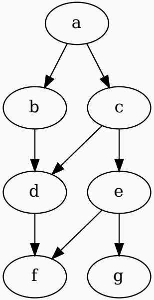
Exit

A node that has no outgoing edges.

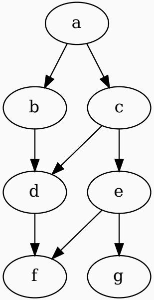
Path

A chain of transition between nodes.

Control Flow Graph



Control Flow Graph



- a is a **entry** node
- f and g are **exists**
- $a \to c \to d \to f$ is a **path** between a and f

Dominance Relations

Motivation

- graph-theoretic concept

Motivation

- graph-theoretic concept
- analyze relations between basic blocks

Motivation

- graph-theoretic concept
- analyze relations between basic blocks
- provide **guarantees** that a basic block *x* is **always** executed before *y*

Motivation

- graph-theoretic concept
- analyze relations between basic blocks
- provide **guarantees** that a basic block x is **always** executed before y
- loop detection and analysis

Motivation

- graph-theoretic concept
- analyze relations between basic blocks
- provide **guarantees** that a basic block *x* is **always** executed before *y*
- loop detection and analysis
- foundation for many **compiler optimizations** and other analysis techniques

Dominator

Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x

Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x
- x is a **dominator** of y: $(x \leq y)$

Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x
- x is a **dominator** of y: $(x \leq y)$
- y is **dominated by** x: $(y \geq x)$

Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x
- x is a **dominator** of y: $(x \leq y)$
- y is **dominated by** x: $(y \geq x)$
- **dom**(y) is the set of all dominators of y (dominator set)

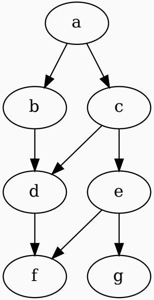
Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x
- x is a **dominator** of y: ($x \leq y$)
- y is **dominated by** x: ($y \geq x$)
- **dom**(y) is the set of all dominators of y (dominator set)
- **each** node dominates **itself**: $y \in$ **dom**(y)

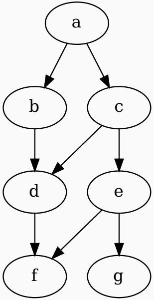
Dominator

- a node x **dominates** a node y if **every path** from the entry node to y **goes through** x
- x is a **dominator** of y: $(x \leq y)$
- y is **dominated by** x: $(y \geq x)$
- $\mathrm{dom}(y)$ is the set of all dominators of y (dominator set)
- **each** node dominates **itself**: $y \in \mathrm{dom}(y)$
- **entry** node dominates **all** nodes in the graph

Dominator Sets

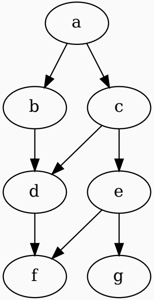


Dominator Sets



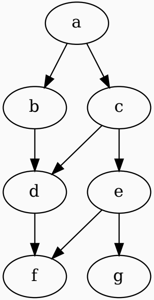
- $\mathrm{dom}(a) = \{a\}$

Dominator Sets



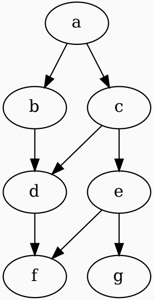
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$

Dominator Sets



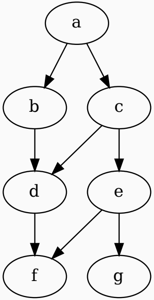
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$

Dominator Sets



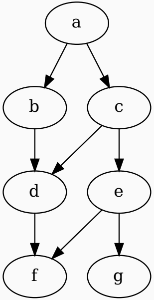
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$

Dominator Sets



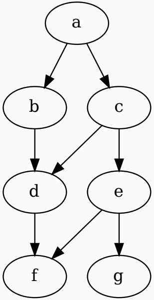
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$

Dominator Sets



- dom(a) $= \{a\}$
- dom(b) $= \{a, b\}$
- dom(c) $= \{a, c\}$
- dom(d) $= \{a, d\}$
- dom(e) $= \{a, c, e\}$
- dom(f) $= \{a, f\}$

Dominator Sets



- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$
- $\text{dom}(f) = \{a, f\}$
- $\text{dom}(g) = \{a, c, e, g\}$

Immediate Dominator

Immediate Dominator

- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$

Immediate Dominator

- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$
- x is an **immediate dominator** of y if

Immediate Dominator

- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$

- x is an **immediate dominator** of y if

  1. $x < y$

Immediate Dominator

- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$
- x is an **immediate dominator** of y if
  1. $x < y$
  2. $\nexists c : x < c < y$

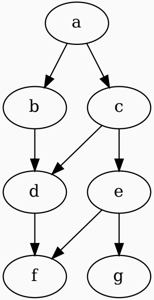
Immediate Dominator

- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$
- x is an **immediate dominator** of y if
  1. $x < y$
  2. $\nexists c : x < c < y$
- x is the **closest dominator** to y with $x \neq y$

Immediate Dominator

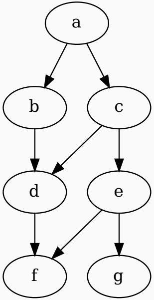
- a node x **strictly dominates** a node y if $x \leq y$ and $x \neq y$: $x < y$
- x is an **immediate dominator** of y if
  1. $x < y$
  2. $\nexists c : x < c < y$
- x is the **closest dominator** to y with $x \neq y$
- **every** node (except entry) has an immediate dominator

Immediate Dominators



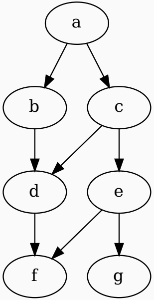
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$
- $\text{dom}(f) = \{a, f\}$
- $\text{dom}(g) = \{a, c, e, g\}$

Immediate Dominators



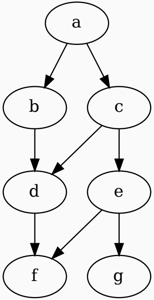
- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$
- $\text{dom}(f) = \{a, f\}$
- $\text{dom}(g) = \{a, c, e, g\}$

Immediate Dominators



- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$
- $\text{dom}(f) = \{a, f\}$
- $\text{dom}(g) = \{a, c, e, g\}$

Immediate Dominators



- $\text{dom}(a) = \{a\}$
- $\text{dom}(b) = \{a, b\}$
- $\text{dom}(c) = \{a, c\}$
- $\text{dom}(d) = \{a, d\}$
- $\text{dom}(e) = \{a, c, e\}$
- $\text{dom}(f) = \{a, f\}$
- $\text{dom}(g) = \{a, c, e, g\}$

Dominator Tree

Dominator Tree

- **compact** representation of dominance relations

Dominator Tree

- **compact** representation of dominance relations
- build from **immediate dominators**

Dominator Tree

- **compact** representation of dominance relations
- build from **immediate dominators**
- x is an immediate dominator of $y \Leftrightarrow (x, y)$ is an edge in the tree

Dominator Tree

- **compact** representation of dominance relations
- build from **immediate dominators**
- x is an immediate dominator of $y \Leftrightarrow (x, y)$ is an edge in the tree
- start node: graph entry

Dominator Tree

- **compact** representation of dominance relations
- build from **immediate dominators**
- x is an immediate dominator of $y \Leftrightarrow (x, y)$ is an edge in the tree
- start node: graph entry
- each node dominates its descendants in the tree

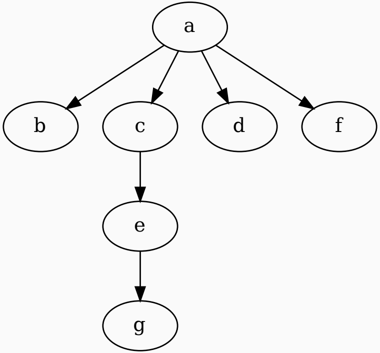
Dominator Tree

$$
\begin{aligned}
\text{dom}(a) &= \{a\} \\
\text{dom}(b) &= \{a, b\} \\
\text{dom}(c) &= \{a, c\} \\
\text{dom}(d) &= \{a, d\} \\
\text{dom}(e) &= \{a, c, e\} \\
\text{dom}(f) &= \{a, f\} \\
\text{dom}(g) &= \{a, c, e, g\}
\end{aligned}
$$

Dominator Tree

$$
\begin{aligned}
\text{dom}(a) &= \{a\} &\Rightarrow&\ \text{root} \\
\text{dom}(b) &= \{a, b\} &\Rightarrow&\ (a, b) \\
\text{dom}(c) &= \{a, c\} &\Rightarrow&\ (a, c) \\
\text{dom}(d) &= \{a, d\} &\Rightarrow&\ (a, d) \\
\text{dom}(e) &= \{a, c, e\} &\Rightarrow&\ (c, e) \\
\text{dom}(f) &= \{a, f\} &\Rightarrow&\ (a, f) \\
\text{dom}(g) &= \{a, c, e, g\} &\Rightarrow&\ (e, g)
\end{aligned}
$$

Dominator Tree



$$
\begin{aligned}
\text{dom}(a) &= \{a\} &\Rightarrow\quad &\text{root} \\
\text{dom}(b) &= \{a, b\} &\Rightarrow\quad &(a, b) \\
\text{dom}(c) &= \{a, c\} &\Rightarrow\quad &(a, c) \\
\text{dom}(d) &= \{a, d\} &\Rightarrow\quad &(a, d) \\
\text{dom}(e) &= \{a, c, e\} &\Rightarrow\quad &(c, e) \\
\text{dom}(f) &= \{a, f\} &\Rightarrow\quad &(a, f) \\
\text{dom}(g) &= \{a, c, e, g\} &\Rightarrow\quad &(e, g)
\end{aligned}
$$

Loops

Motivation

- common construct on function level

Motivation

- common construct on function level
- **easy** to spot in control flow graphs

Motivation

- common construct on function level
- **easy** to spot in control flow graphs
- graph-theoretical **properties** that **facilitate** many kinds of analysis

Motivation

- common construct on function level
- **easy** to spot in control flow graphs
- graph-theoretical **properties** that **facilitate** many kinds of analysis
- **automated loop analysis** fundamental for many reverse engineering tasks

Motivation

- common construct on function level
- **easy** to spot in control flow graphs
- graph-theoretical **properties** that **facilitate** many kinds of analysis
- **automated loop analysis** fundamental for many reverse engineering tasks

What are loops and how can we find them?

Strongly Connected Component

A subgraph in which **each** node is **reachable** from **every** other node.

Loops

Strongly Connected Component

A subgraph in which **each** node is **reachable** from **every** other node.

- natural loops
  - compiler generated
  - strong mathematical properties

Loops

Strongly Connected Component

A subgraph in which **each** node is **reachable** from **every** other node.

- natural loops
    - compiler generated
    - strong mathematical properties
- irreducible loops
    - complicate to analyze
    - rarely seen (hand-written assembly, code obfuscation)

Loops

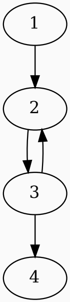
Strongly Connected Component

A subgraph in which **each** node is **reachable** from **every** other node.

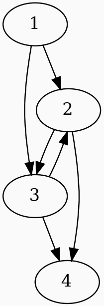
- natural loops
  - compiler generated
  - strong mathematical properties
- irreducible loops
  - complicate to analyze
  - rarely seen (hand-written assembly, code obfuscation)

We focus only on natural loops.

Natural and Irreducible Loop



natural loop



irreducible loop

Natural Loop

Natural Loop

- strong mathematical properties

Natural Loop

- strong mathematical properties
- generated by compilers

Natural Loop

- strong mathematical properties
- generated by compilers
- **loop header**: **single** entry point that **dominates** a loop

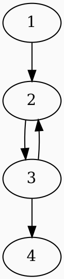
Natural Loop

- strong mathematical properties
- generated by compilers
- **loop header**: **single** entry point that **dominates** a loop
- **back edge**: edge to a **dominator**

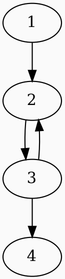
Natural Loop

- strong mathematical properties
- generated by compilers
- **loop header**: **single** entry point that **dominates** a loop
- **back edge**: edge to a **dominator**
- **loop body**: set of **nodes within** a loop

Natural Loop

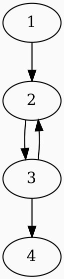


Natural Loop



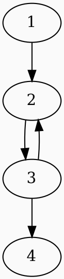
- 2 is loop **header** that **dominates** loop

Natural Loop



- 2 is loop **header** that **dominates** loop
- $\{2, 3\}$ is loop **body**

Natural Loop



- 2 is loop **header** that **dominates** loop
- $\{2, 3\}$ is loop **body**
- $(3, 2)$ is **back edge** to the dominator

Natural Loop Detection

Natural Loop Detection

- find a back edge

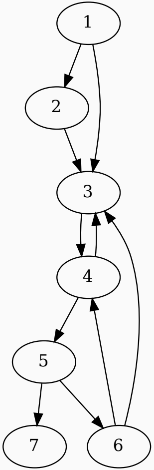
    1. x dominates y

    2. there is an edge (y, x)

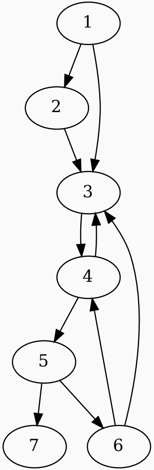
Natural Loop Detection

- find a back edge
  1. x dominates y
  2. there is an edge (y, x)
- identify the loop body
  1. collect all **nodes** that are **dominated by** x
  2. **filter** nodes that can **reach** y without visiting x

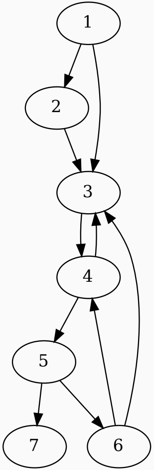
Natural Loop Detection



Natural Loop Detection

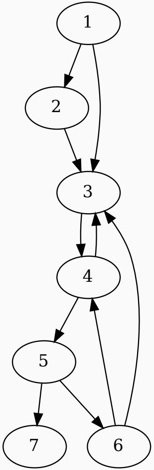


edge	body



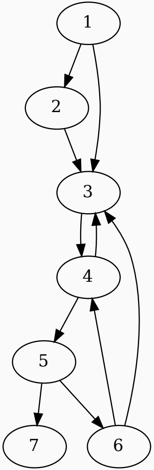
edge	body
$(4, 3)$:	

Natural Loop Detection



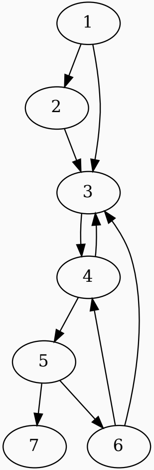
edge	body
$(4, 3)$:	$\{3, 4, 5, 6\}$

Natural Loop Detection



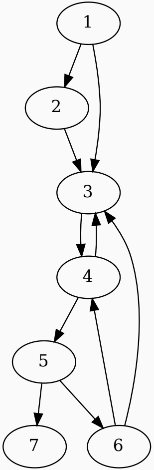
edge	body
$(4, 3)$:	$\{3, 4, 5, 6\}$
$(6, 4)$:	

Natural Loop Detection



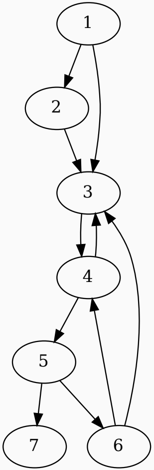
edge	body
$(4, 3)$:	$\{3, 4, 5, 6\}$
$(6, 4)$:	$\{4, 5, 6\}$

Natural Loop Detection



edge	body
$(4, 3)$:	$\{3, 4, 5, 6\}$
$(6, 4)$:	$\{4, 5, 6\}$
$(6, 3)$:	

Natural Loop Detection



edge	body
$(4, 3)$:	$\{3, 4, 5, 6\}$
$(6, 4)$:	$\{4, 5, 6\}$
$(6, 3)$:	$\{3, 4, 5, 6\}$

Nested Loops

Nesting Relations

Loops can be

- merged
- disjoint
- nested

Nesting Relations

Loops can be

- merged
  - they have the **same** header
- disjoint
- nested

Nesting Relations

Loops can be

- merged
  - they have the **same** header
  - **hard to tell** how they relate to each other
- disjoint
- nested

Nesting Relations

Loops can be

- merged
  - they have the **same** header
  - **hard to tell** how they relate to each other
- disjoint
  - if they have **different** headers
- nested

Nesting Relations

Loops can be

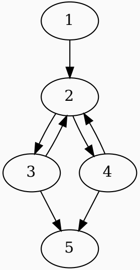
- merged
  - they have the **same** header
  - **hard to tell** how they relate to each other
- disjoint
  - if they have **different** headers
  - their **intersection** is empty
- nested

Nesting Relations

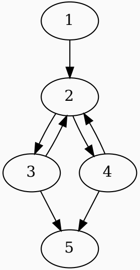
Loops can be

- merged
  - they have the **same** header
  - **hard to tell** how they relate to each other
- disjoint
  - if they have **different** headers
  - their **intersection** is empty
- nested
  - one function body is **entirely contained** within the other

Merged Loops



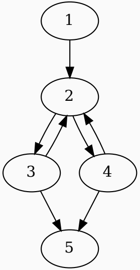
Merged Loops



l_1: $\{2, 3\}$

l_2: $\{2, 4\}$

Merged Loops

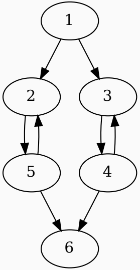


l_1: $\{2, 3\}$

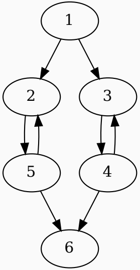
l_2: $\{2, 4\}$

$l_1 \cap l_2 = \{2\}$

Disjoint Loops



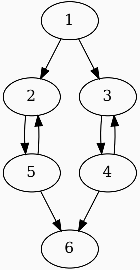
Disjoint Loops



l_1: $\{2, 5\}$

l_2: $\{3, 4\}$

Disjoint Loops

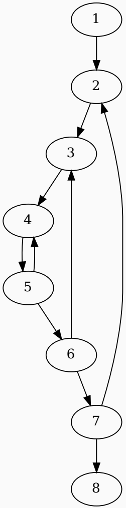


l_1: $\{2, 5\}$

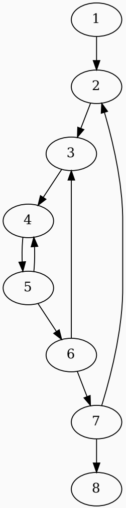
l_2: $\{3, 4\}$

$l_1 \cap l_2 = \emptyset$

Nested Loops

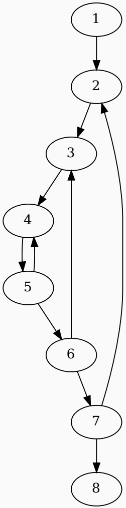


Nested Loops



l_1: $\{4, 5\}$

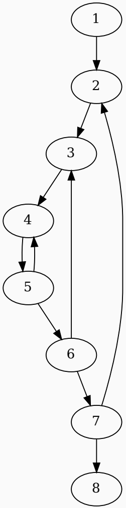
Nested Loops



l_1: $\{4, 5\}$

l_2: $\{3, 4, 5, 6\}$

Nested Loops

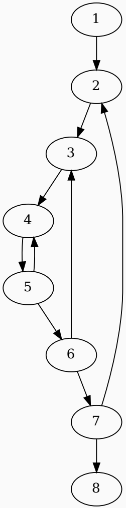


l_1: $\{4, 5\}$

l_2: $\{3, 4, 5, 6\}$

l_3: $\{2, 3, 4, 5, 6, 7\}$

Nested Loops

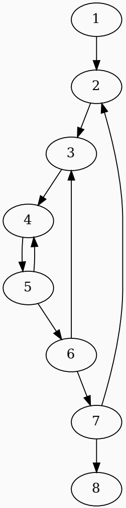


l_1: $\{4, 5\}$ innermost loop

l_2: $\{3, 4, 5, 6\}$

l_3: $\{2, 3, 4, 5, 6, 7\}$

Nested Loops

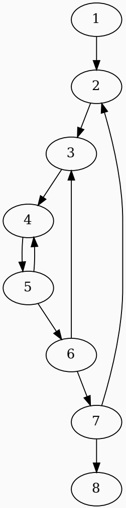


l_1: $\{4, 5\}$ innermost loop

l_2: $\{3, 4, 5, 6\}$ inner/outer loop of l_3/l_1

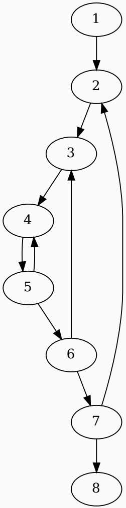
l_3: $\{2, 3, 4, 5, 6, 7\}$

Nested Loops



l_1:	$\{4, 5\}$	innermost loop
l_2:	$\{3, 4, 5, 6\}$	inner/outer loop of l_3/l_1
l_3:	$\{2, 3, 4, 5, 6, 7\}$	outermost loop

Nested Loops



l_1: $\{4, 5\}$ innermost loop

l_2: $\{3, 4, 5, 6\}$ inner/outer loop of l_3/l_1

l_3: $\{2, 3, 4, 5, 6, 7\}$ outermost loop

$l_1 \subset l_2 \subset l_3$

Loop Unrolling

Motivation

- reasoning about loops can be hard

Motivation

- reasoning about loops can be hard
  - undecidability

Motivation

- reasoning about loops can be hard
    - undecidability
    - termination condition

Motivation

- reasoning about loops can be hard
  - undecidability
  - termination condition
  - path explosion

Motivation

- reasoning about loops can be hard
  - undecidability
  - termination condition
  - path explosion
  - large number of iterations

Motivation

- reasoning about loops can be hard
  - undecidability
  - termination condition
  - path explosion
  - large number of iterations
- analysis with a fixed number of loop iterations beneficial

Motivation

- reasoning about loops can be hard
  - undecidability
  - termination condition
  - path explosion
  - large number of iterations
- analysis with a fixed number of loop iterations beneficial
  - many questions remain decidable

Motivation

- reasoning about loops can be hard
  - undecidability
  - termination condition
  - path explosion
  - large number of iterations
- analysis with a fixed number of loop iterations beneficial
  - many questions remain decidable
  - limited analysis scope

Loop Unrolling

- set an **upper** iteration **bound** k

Loop Unrolling

- set an **upper** iteration **bound** k
- transform control flow graph into semantically a **directed acyclic graph**

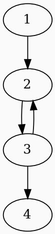
Loop Unrolling

- set an **upper** iteration **bound** k
- transform control flow graph into semantically a **directed acyclic graph**
  1. **remove** back edge
  2. **duplicate** nodes of loop body k times and **preserve** edge structure

Loop Unrolling

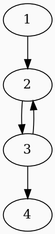
- set an **upper** iteration **bound** k
- transform control flow graph into semantically a **directed acyclic graph**
  1. **remove** back edge
  2. **duplicate** nodes of loop body k times and **preserve** edge structure
- transformed graph is **semantically equivalent** for up to k loop iterations

Loop Unrolling

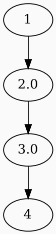


natural loop

Loop Unrolling

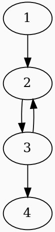


natural loop

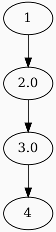


unrolling depth 0

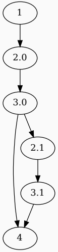
Loop Unrolling



natural loop



unrolling depth 0



unrolling depth 1

Conclusion

Control Flow Analysis

- basic blocks
- control flow graph construction
- dominance relations
- natural loop detection
- loop properties and transformations