Control Flow Analysis

Control Flow Analysis – Running Example

Example (cont.): Control Flow Graph

Detecting basic blocks

basic block (BB) = max. sequence of consecutive statements (IR or target level) that can be entered by program control only via the first one and left only via the last one.

= max sequence of consecutive statements (IR or target level)

basic block (BB)

1. Dominator-based analysis
2. Interval analysis
3. Structural analysis

Detecting basic blocks

identifying basic blocks of a routine

construct the flow graph / basic block graph

- if-then-else: candidates for predication
- loops: candidates for loop transformations, software pipelining from MIR to unstructured source code or from target code

- necessary to enable global optimizations beyond basic blocks

- exception-based control transfer not considered here

- for most cases, but not for e.g. instruction scheduling

- call instructions need not define the basic block

- instruction immediately following a branch or return + entry point of a routine, or

1. first instruction (leader) of a BB: either

and last only via the last one.

= max sequence of consecutive statements (IR or target level)

basic block (BB)

1. Dominator-based analysis
2. Interval analysis
3. Structural analysis

Detecting basic blocks

identifying basic blocks of a routine

construct the flow graph / basic block graph

- if-then-else: candidates for predication
- loops: candidates for loop transformations, software pipelining from MIR to unstructured source code or from target code

- necessary to enable global optimizations beyond basic blocks

- exception-based control transfer not considered here

- for most cases, but not for e.g. instruction scheduling

- call instructions need not define the basic block

- instruction immediately following a branch or return + entry point of a routine, or

1. first instruction (leader) of a BB: either

and last only via the last one.

= max sequence of consecutive statements (IR or target level)

basic block (BB)

1. Dominator-based analysis
2. Interval analysis
3. Structural analysis

Detecting basic blocks

identifying basic blocks of a routine

construct the flow graph / basic block graph

- if-then-else: candidates for predication
- loops: candidates for loop transformations, software pipelining from MIR to unstructured source code or from target code

- necessary to enable global optimizations beyond basic blocks

- exception-based control transfer not considered here

- for most cases, but not for e.g. instruction scheduling

- call instructions need not define the basic block

- instruction immediately following a branch or return + entry point of a routine, or

1. first instruction (leader) of a BB: either

and last only via the last one.

= max sequence of consecutive statements (IR or target level)

basic block (BB)

1. Dominator-based analysis
2. Interval analysis
3. Structural analysis

Detecting basic blocks

identifying basic blocks of a routine

construct the flow graph / basic block graph

- if-then-else: candidates for predication
- loops: candidates for loop transformations, software pipelining from MIR to unstructured source code or from target code

- necessary to enable global optimizations beyond basic blocks

- exception-based control transfer not considered here

- for most cases, but not for e.g. instruction scheduling

- call instructions need not define the basic block

- instruction immediately following a branch or return + entry point of a routine, or

1. first instruction (leader) of a BB: either

and last only via the last one.

= max sequence of consecutive statements (IR or target level)

basic block (BB)
Basic-block graph

Terminology: in [Muchnick'97] called control-flow graph (CFG), whereas "our" CFG (statement level) is there called a "flowchart".

rooted, directed graph $G = (N; E)$:
- nodes = basic blocks + entry + exit
- edges = control flow edges from CFG/flowchart: a successor BB's of a BB $b$:
  $\text{Succ}(b) = \{ n \in N : (b; n) \in E \}$
- predecessor BB's of a BB $b$:
  $\text{Pred}(b) = \{ n \in N : (n; b) \in E \}$

Extended basic blocks, regions

Extended basic block (EBB) = max. sequence of instructions beginning with a leader that contains no join nodes other than (maybe) its first node = max. sequence of instructions beginning with a leader.

EBBs are useful for some optimizations e.g. instruction scheduling.

EBB also known as region

EBB's are useful for some optimizations e.g. instruction scheduling.

Algorithm for computing the EBB's of a CFG: see e.g. [Muchnick 7.1]

Example (cont.): CFG + Basic blocks

Example (cont.): CFG: CFG — Basic block graph

Example (cont.): CFG: CFG — Basic block leaders

Extended basic block graph

The extended basic block graph is a directed graph where each node represents an extended basic block, and each edge represents a control flow edge from the source EBB to the destination EBB.

Predicate EBBs of a BB $b$: $\text{Pred}(b) = \{ n \in N : (n; b) \in E \}$

Successor EBBs of a BB $b$: $\text{Succ}(b) = \{ n \in N : (b; n) \in E \}$

Initial basic block $\text{init} = \text{entry}$, final basic block $\text{exit}$

The control flow graph (CFG) is a directed graph where each node represents a basic block, and each edge represents a control flow edge from the source BB to the destination BB.

Region $= \text{strongly connected subgraph (SCC) of the CFG with a single entry}$

Extended basic block (EBB) = max. sequence of instructions beginning with a leader that contains no join nodes other than (maybe) its first node.
Finding Loops

Example: DFS-tree, edge classification

dfs-numbers: order in which dfs enters nodes

tree edges: edges followed by dfs via recursive calls

Depth-first search (dfs): recursively explore descendants of a node

Forward edges, Back edges, Cross edges

not unique, depends on ordering of descendants

example:


Example:

Find uniform treatment for program loops

while, for, goto, ... (compiler-inserted exit-revision)

Loops may be expressed in programs by different constructs

- Loop unrolling, look parallelization, software pipelining
- Optimizations that exploit the loop structure are important

Loops spend most of the execution time in loops.

Use a general approach based on graph-theoretic properties of CFG.

See also DFS-slide on course homepage.

Graph-theoretic concepts of control-flow analysis (1)

Example: DFS-tree, edge classification


Graph-theoretic concepts of control-flow analysis

### preorder traversal

Let \( G = (N, E) \) be a digraph. A preorder traversal of \( G \) is a linear ordering of its nodes such that for every edge \( (u, v) \in E \), node \( u \) precedes node \( v \) in the ordering.

#### Definition

**preorder traversal** of a digraph \( G = (N, E) \):

1. **Visit** the source node (entry node).
2. **Visit** each node \( v \) in \( N \) in such a way that if \( (u, v) \in E \), then node \( u \) is visited before node \( v \).
3. **Visit** each node \( v \) in \( N \) in reverse order.

#### Example

Imagine a source of light going into the entry node, nodes are transparent and edges are optical fibers. Place an opaque barrier at node \( v \). Nodes dominated by \( v \) get dark.

**Nodes are transparent and edges are optical fibers.**

**Place an opaque barrier at node \( v \) → nodes dominated by \( v \) get dark.**

---

**Dominance intuition (1)**

The entry node dominates all nodes:

- Entry node dominates all nodes.
- If \( q \neq p \) and \( q \) dominates \( p \) in the dominator tree, \( q \) is uniquely marked.
- If \( q \) dominates in the dominator tree, \( q \) is immediately dominated.
- Domination is reflexive, transitive, and antisymmetric.
- If \( q \) dominates in the dominator tree, \( q \) is processable after its descendants.

---

**Dominance intuition (2)**

- Strict dominance: If \( q \neq p \) and \( q \) dominates \( p \) in the dominator tree, \( q \) is uniquely marked.
- Immediately dominated: Domination is reflexive, transitive, and antisymmetric.
- If every possible execution path includes \( q \) as a node, \( q \) is definitely the dominator.
- Given: Flow graph \( G = (N, E) \), \( N \) denotes processable after its descendants.

---

**Dominance, Immediate Dominance, Strict Dominance**

- Direct dominance: \( d \) directly dominates \( b \) if \( d \) is the dominator of \( b \).
- Immediate dominance: \( i \) immediately dominates \( b \) if \( i \) is the dominator of \( b \) and there is no \( c \neq i \), \( c \neq b \), with \( i \) dominating \( c \) and \( c \) dominating \( b \).
Dominance intuition (3)

Node $B_3$ dominates $B_3$, $B_4$, $B_5$, $B_6$:

- Entry $B_1$
- $B_2$ $B_3$
- $B_4$ $B_6$
- Exit $B_5$

$Y$ $N$ $Y$ $Lamp$

Postdominance $p$ postdominates $b$ if every possible execution path $b$ does not exit includes $p$ in the reversed flow graph.

$P_{dom}$:

Algorithm 1:

1. For all $n \in N$ in dfsnum order
   
   - $D(n) = \text{true}$; $\text{Dom}(r) = f$;
   
   - Change $\text{change} \rightarrow \text{false}$
   
   - For all $n \in N$: $D(n) \leftarrow \text{false}$;
   
   - While ($\text{change})$
     
     - For all $n \in N$
       
       - Dom($p$) = Dom($n$); $\text{change} \leftarrow \text{true}

$P_{dom}$:

Node $B_3$ dominates $B_3$, $B_4$, $B_5$, $B_6$:
Example: Computing dominators

\[ \text{Algorithm 2 [Lengauer/Tarjan'79]} \]

based on depth first search and path compression

Total time: \( O\left(\frac{n^2}{\log n}\right) \) or \( O\left(\frac{n}{\alpha(n)}\right) \)

(see e.g. [Muchnick pp. 185–190])
Loops and Strongly Connected Components

We call a (backward, $B$) edge $m / (m /;; n)$ a loop back edge if $n \text{dom} m$.

Remark: Not every $B$ edge is a loop back edge!

Natural loop of a loop back edge

Algorithm: Compute the loop node set for a given loop back edge $m / (m /;; n)$

- Start by marking $m$ and $n$ as loop nodes.
- Backwards from $m$, (df)search predecessors $v$, stopping recursive backward search at already found loop nodes.

Example (cont.): Natural Loop

entry $B_1$ $B_2$ $B_3$ $B_4$ exit $B_5$

$\exists$ Can't exist because $n \text{dom} m$

Identifying the natural loop of a loop back edge

For technical reasons, add a pre-header (initially empty).

$\exists$ header, pre-header

Natural loop of a loop back edge

If the header has more than 2 predecessors:

For technical reasons, add a pre-header (initially empty).

Example (cont.): Natural Loop

entry $B_1$ $B_2$ $B_1$ $B_2$

We call a (backward, $B$) edge $(m / (m /;; n))$ a loop back edge $m / (m /;; n)$ dom $m$.

$\exists$ Does not dominate a natural loop ($\exists$ entry points)

$\exists$ does not dominate a natural loop ($\exists$ entry points)

$\exists$ is the loop header.

From which $m$ can be reached without passing through $m$

Natural loop of a loop back edge

Remark: Not every $B$ edge is a loop back edge.

Loops and Strongly Connected Components

Easier to place new instructions immediately before the loop
Properties of Natural Loops

Two natural loops with different headers are either disjoint or one is nested in the other.

Each natural loop is a SCC.

Background:
Strongly connected component (SCC) = subgraph \( S = (V_S, E) \), where every node in \( V_S \) is reachable in \( S \).

SCCs can be computed with Tarjan's algorithm (extension of dfs).

A SCC is reducible if all \( g \) edges in any DFS tree are loop back edges.

Reducibility of flow graphs

A flow-graph is reducible if all \( B \) edges in any DFS tree are loop back edges.

Intuitively: "If there are no jumps into the middle of loops (e.g., goto's), the flow graph is reducible."

Reducible flow graphs are well-structured (loops properly nested).

Irreducible flow graphs are rare and can be made reducible by replicating nodes.

Interval analysis

Divide the flowgraph into regions (e.g., loops in CFA).

Repeatedly collapse a region to an abstract node.

Abstract flowgraph

Nested regions (control tree)

Hierarchical folding structure allows for faster / simpler data flow analysis.

Example:

A SCC \( S \) is maximal if every SCC containing \( S \) is just \( S \) itself.

\[ S^2 = \{B2,B3\} \text{ is a SCC but not a maximal SCC.} \]

\[ S^1 = \{B1\} \text{ is a maximal SCC.} \]

\[ S^2 = \{B2\} \text{ is a SCC but not a maximal SCC.} \]

\[ S^1 = \{B1\} \text{ is a maximal SCC.} \]

Simplified variant: [Ullman '73]

in time \( O(|V|+|E|) \) [Tarjan '72] 

Each natural loop is a SCC.

Two natural loops with different headers are either disjoint or one is nested in the other.
**Structural Analysis**

- Structural analysis is a special case of interval analysis.
- CFG folding follows the hierarchical structure of the program.
- Folding transformations for loops, if-then-else, switch, etc.
- Every region has 1 entry point.
- Works only for well-structured programs.
- Every region has 1 entry point.
- Every region has 1 entry point.
- Folding transformations for loops, if-then-else, switch, etc.
- CFG folding follows the hierarchical structure of the program.
- Is a special case of interval analysis.

---

**Example (cont.): Structural analysis**

For each construct:
- Equations, etc., for dataflow analysis can be pre-formulated.

---

**Remark:** If only loop-based regions are of interest, the hierarchy flattens accordingly (R8 and R9 merged at top level).
Computing a bottom-up order of regions of a reducible flow graph

1. \( R \rightarrow \{B_1, B_2, ..., B_n\} = \) all leaf regions, i.e., all single blocks in \( G \), in any order

2. Repeat

   \( R \rightarrow \{\) the loop region for \( L \)\}

   \( R \rightarrow \{\) the body of \( L \) without the back edges to the header of \( L \)\}

   Choose a natural loop \( L \) such that, if there are any natural loops \( L' \) contained within \( L \), then the (body and loop) regions for these \( L' \) were already added to \( R \).

3. If the entire flow graph is not itself a natural loop, \( R \rightarrow \{\) the region consisting of the entire flow graph \}.\)

Output: A bottom-up ordered list \( R \) of loop-based regions of \( G \).

Summary:

Control-Flow Analysis

- Structural analysis
- Internal-based CFA
- Dominator-based CFA
- Loop detection
- Basic blocks, extended basic blocks