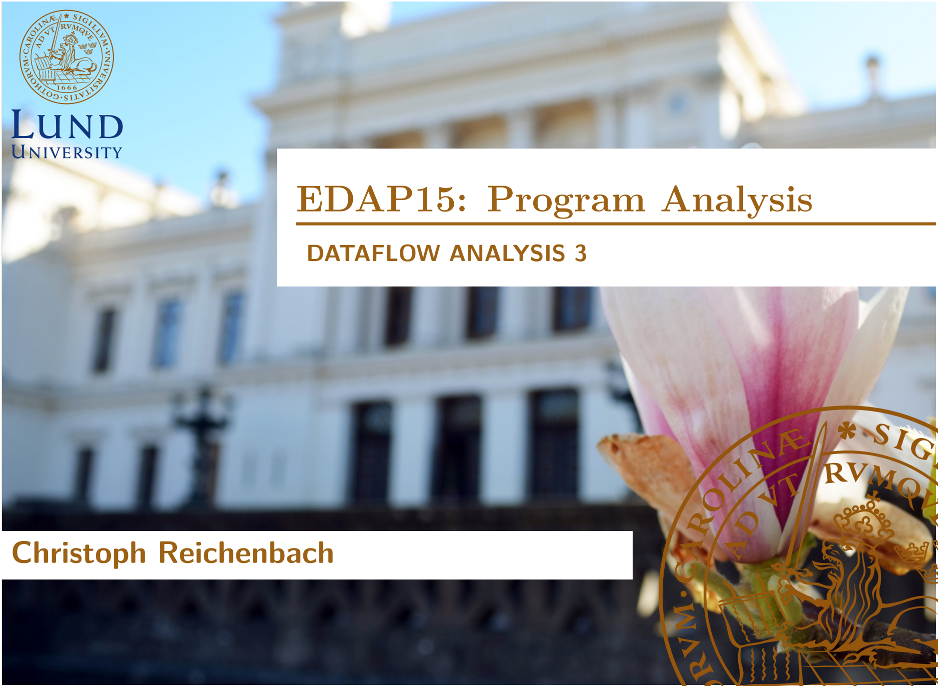
EDAP15: Program Analysis

DATAFLOW ANALYSIS 3



Christoph Reichenbach

Welcome back!

- No new homework this week
- Questions?

Monotonicity Revisited

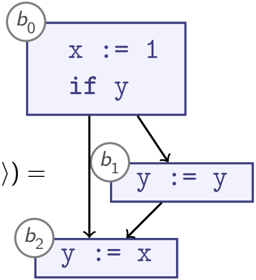
- f is monotonic (wrt $\sqsubseteq$) iff:

$$x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$$

- What does this tell us about $f(f(x))$ vs. $f(x)$?
- No direct connection to fixpoints!

Naïve Iteration Revisited

Analysis on $\mathbb{Z}_\perp^\top \times \mathbb{Z}_\perp^\top$



$$trans_0(\{x \mapsto v_x,\ y \mapsto v_y\}) = \{x \mapsto \mathbf{1},\ y \mapsto v_y\}$$

$$trans_{all}(\langle \mathbf{in}_0, \mathbf{out}_0, \mathbf{out}_1, \mathbf{out}_2 \rangle) = \left\langle \begin{array}{l} \mathbf{in}_0, \\ trans_0(\mathbf{out}_0), \\ trans_1(\mathbf{out}_1), \\ trans_2(\mathbf{out}_0 \sqcup \mathbf{out}_1) \end{array} \right\rangle$$

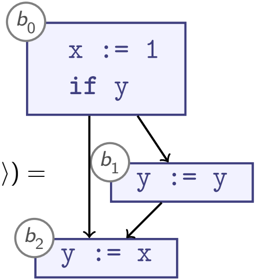
$$trans_1(S) = S$$

$$trans_2(\{x \mapsto \mathbf{v_x},\ y \mapsto v_y\}) = \{x \mapsto v_x,\ y \mapsto \mathbf{v_x}\}$$

	$\mathbf{I}$	$trans_{all}^1(\mathbf{I})$	$trans_{all}^2(\mathbf{I})$	$trans_{all}^3(\mathbf{I})$
$\mathbf{in}_0$	$\perp$	$\perp$	$\perp$	$\perp$
$\mathbf{out}_0$	$\perp$	$x \mapsto 1$	$x \mapsto 1$	$x \mapsto 1$
$\mathbf{out}_1$	$\perp$	$\perp$	$x \mapsto 1$	$x \mapsto 1$
$\mathbf{out}_2$	$\perp$	$\perp$	$x \mapsto 1, y \mapsto 1$	$x \mapsto 1, y \mapsto 1$

Naïve Iteration Revisited

Analysis on
$\mathbb{Z}_\perp^\top \times \mathbb{Z}_\perp^\top$



$$trans_0(\{x \mapsto v_x,\ y \mapsto v_y\}) = \{x \mapsto \mathbf{1},\ y \mapsto v_y\}$$

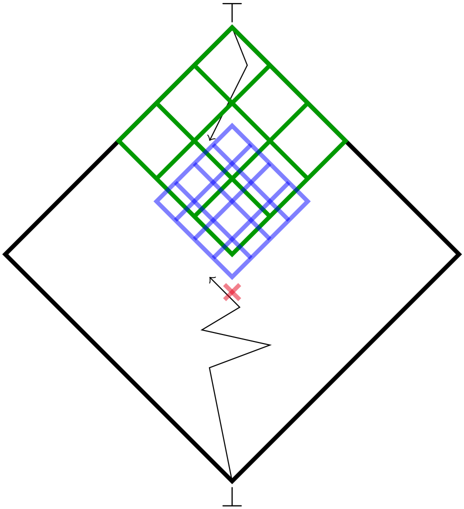
$$trans_1(S) = S$$

$$trans_2(\{x \mapsto \mathbf{v_x},\ y \mapsto v_y\}) = \{x \mapsto v_x,\ y \mapsto \mathbf{v_x}\}$$

$$trans_{all}(\langle \mathbf{in}_0, \mathbf{out}_0, \mathbf{out}_1, \mathbf{out}_2 \rangle) = \left\langle \begin{array}{l} \mathbf{in}_0, \\ trans_0(\mathbf{out}_0), \\ trans_1(\mathbf{out}_1), \\ trans_2(\mathbf{out}_0 \sqcup \mathbf{out}_1) \end{array} \right\rangle$$

	$\mathbf{I}$	$trans^1_{all}(\mathbf{I})$	$trans^2_{all}(\mathbf{I})$	$trans^3_{all}(\mathbf{I})$
$\mathbf{in}_0$	$\top$	$\top$	$\top$	$\top$
$\mathbf{out}_0$	$\top$	$x \mapsto 1, y \mapsto \top$	$x \mapsto 1, y \mapsto \top$	$x \mapsto 1, y \mapsto \top$
$\mathbf{out}_1$	$\top$	$\top$	$x \mapsto 1, y \mapsto \top$	$x \mapsto 1, y \mapsto \top$
$\mathbf{out}_2$	$\top$	$\top$	$\top$	$x \mapsto 1, y \mapsto 1$

Least Fixed Point vs MFP



MFP

Naïve Iteration

MOP

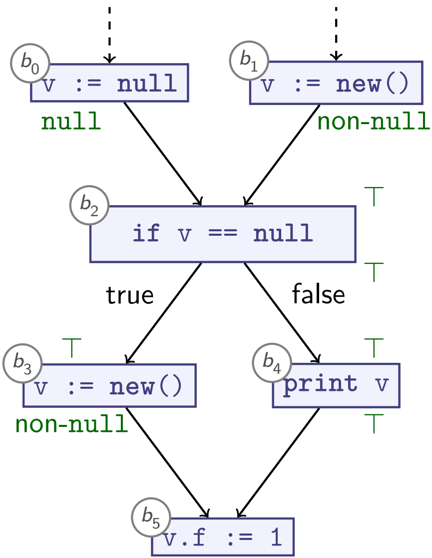
Summary

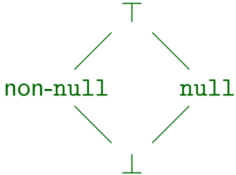
- **MFP**
  - Efficient
  - Fixpoint $\sqsupseteq$ starting point
- **Naïve fixpoint iteration**
  - Fixpoint may be *above* or *below* starting point
- **MOP**
  - One fixpoint, no “starting point”
  - Maximal Precision
  - Undecidable in general
- This list of fixpoint algorithms is not exhaustive
- Different fixpoint lattices per algorithm
- All fixpoints are sound overapproximations

Dimensions of Data Flow

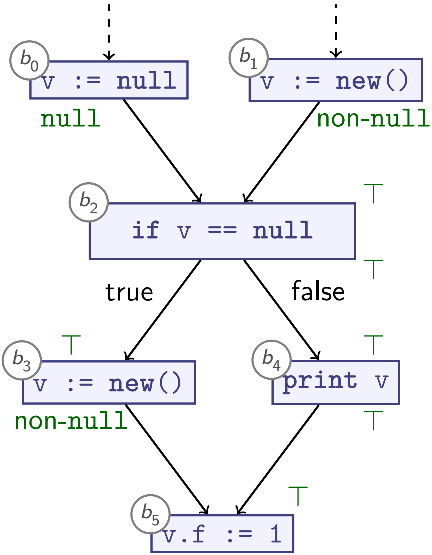
- Data Flow analysis is highly versatile
- Scalable by adjusting:
  - Lattice and transfer functions
  - Treatment of subroutine calls
  - Data representation
- Today we explore four dimensions of scalability:
  - **More precision**: Control- and Path sensitivity
  - **More speed**: Gen/Kill sets
  - **Infinite lattices**: Widening
  - **Subroutines**

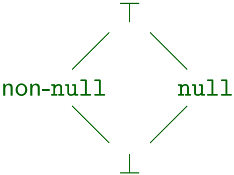
Control Sensitivity





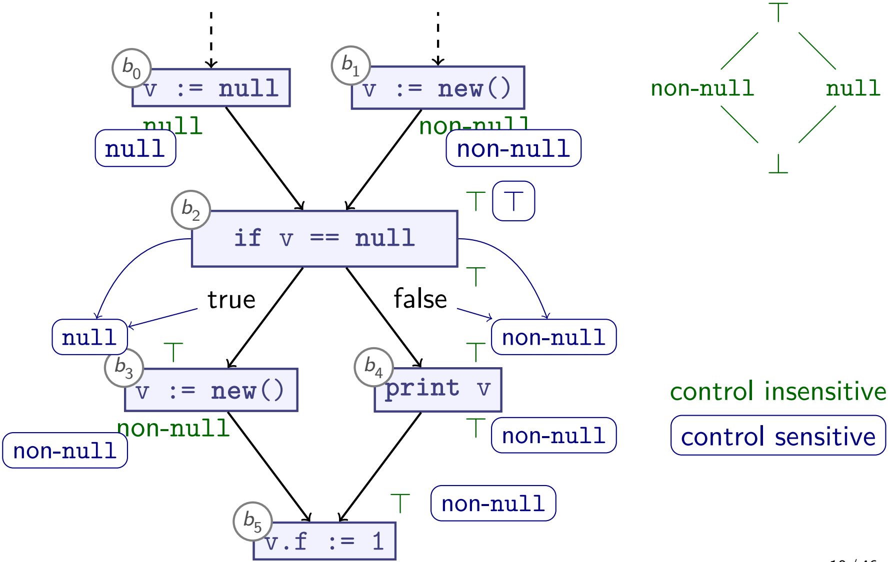
Control Sensitivity



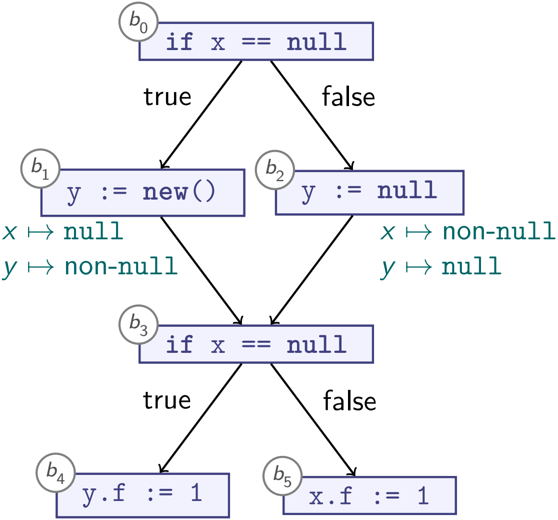


control insensitive

Control Sensitivity

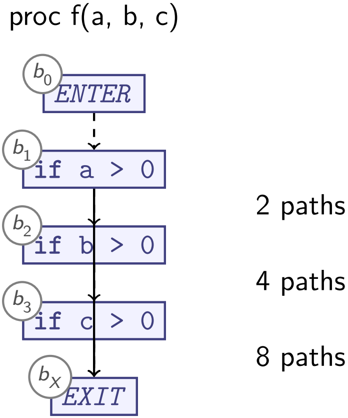


Multiple Conditionals



Should we carry path information across merge points?

Path Sensitivity

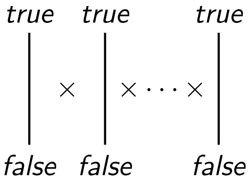


Number of paths grows exponentially

Summary

- **Control sensitive** analysis considers conditionals:
  - May propagate different information along different edges:
    - **if** P:
    - Special transfer function for ‘**assert** P’ on ‘true’ edge
    - Special transfer function for ‘**assert** `not` P’ on ‘false’ edge
- **Path sensitive** analysis considers one sequence of CFG edges (execution path) at a time:
  - May propagate different information along different paths
  - High precision possible, but must cover *all* paths
  - Number of paths $O(\#$ of conditionals$)$
  - Avoid exponential blow-up by merging (as before)
  - Path-sensitive procedure summaries might require exponential number of cases
  - *Usually* not practical

Product Lattices over Binary Lattices



- Recall binary lattices:
  - $\top$ = *true*
  - $\bot$ = *false*
  - $\sqcup$ = *logical "or"*
  - $\sqcap$ = *logical "and"*
- Computer hardware can compute $\sqcup$, $\sqcap$ of multiple lattices in parallel:
  - Bitwise or/and

  $\Longrightarrow$ Highly efficient
- Can represent other lattices efficiently, too

Give rise to highly efficient *Gen-/Kill-Set* based program analysis

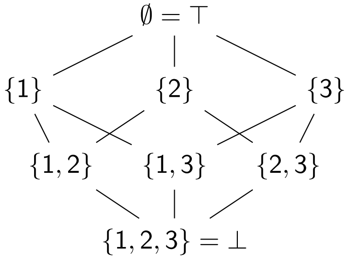
Dataflow Analysis

Analyse properties of variables or basic blocks

Examples in practice:

- *Live Variables*
  Is this variable ever read?
- *Reaching Definitions*
  What are the possible values for this variable?
- *Available Expressions*
  What variable definitely has which expression?

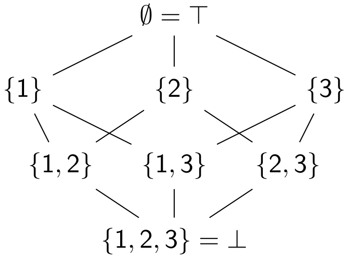
Analyses on Powersets (1/2)



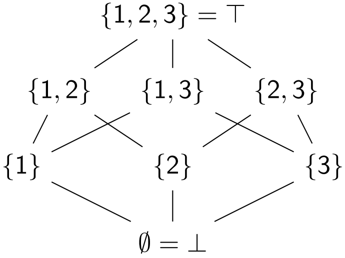
$join_b = \cup$

- Common: 'Which elements of S are possible / necessary?'
  - $S \subseteq \mathbb{Z}$ (*Reaching Definitions*)
    - S = Numeric Constants in code $\cup \{0, 1\}$
  - S = Variables (*Live Variables*)
  - S = Program Locations (*alt. Reaching Definitions*)
  - S = Types
- Abstract Domain: Powerset $\mathcal{P}(S)$
  - Finite iff S is finite

Analyses on Powersets (2/2)



$join_b = \cup$



$join_b = \cap$

- $join_b$ can be $\cup$ or $\cap$
- $\cup$:
  - Property that is true over *any* path
  - **May**-analysis (e.g., Reaching Definitions)
- $\cap$:
  - Property that is true over *all* paths
  - **Must**-analysis

Gen-Sets and Kill-Sets

- Many transfer functions $trans_b$ have the following form:
  - Remove set of options $kill_{x,b}$ from each variable x
  - Add set of options $gen_{x,b}$ to each variable x
  - Don't depend on other variables

  $$trans_b(\{x \mapsto A, \ldots\}) = \{x \mapsto (A \setminus kill_{x,b}) \cup gen_{x,b}, \ldots\}$$

- Bit-vector implementation:
  - $A \setminus B$: bitwise-AND and bitwise-NOT
  - $A \cup B$: bitwise-OR
- Examples:
  - *Reaching Definitions* on finite domain
    - *gen*: assignment to var in current basic block
    - *kill*: other existing assignments to same var
  - *Live Variables*
    - *gen*: used variables
    - *kill*: overwritten variables

Gen/Kill: Available Expressions

"Which expressions do we currently have evaluated and stored?"

C

```c
int x = 3 + z;
int y = 2 + z;
if (z > 0) {
  x = 4;
}
f(2 + z); // Can re-use y here!
```

- Forward analysis
- *gen*: any expression assigned to the variable
- *kill*: any other expression
- $join_b = \cap$

Gen/Kill: Very Busy Expressions

"Which expression do we definitely need to evaluate at least once?"

C

```c
// (x / 42) is very busy:  (A),(B)
if (z > 0) {
  x = 4 + x / 42; // (A)
  y = 1;
} else {
  x = x / 42; // (B)
}
g(x);
```

- Backward analysis
- *gen*: any expression assigned to the variable
- *kill*: any other expression
- $join_b = \cap$

Summary

- Common: Abstract Domain is powerset of some set S
- Transfer function $trans_b$:

$$trans_b(\{x \mapsto A, \ldots\}) = \{x \mapsto (A \setminus kill_{x,b}) \cup gen_{x,b}, \ldots\}$$

- *kill*: 'Kill set': Entries of S to remove
- *gen*: 'Gen set': Entries of S to add
- $join_b$ is $\cup$ or $\cap$
- Often admits very efficient implementation

	May	**Must**
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Very Busy Expressions

Numerical Domains

Teal

```
// valid index range:  [0, 2]
var a := [1, 2, 3];
var i := 0;
var result = 0;
while i <= 3 {
  result += a[i];
  i := i + 1;
}
```

- Bug: `i` may be 3, and out of bounds for `a`
- Analysis: Compute bounding intervals [*min*, *max*]
  - **Interval Abstract Domain**
- `i` : $[0, 3]$

Numerical Domains

Teal

```
var a := [1, 2, 3];
var i := 0;
var r          new array[int](3);
while i < 3 {
  var j := 0;
  var c :
  while j < 3 - i {
    c := c + a[i + j];
    j := j + 1;
  }
  result[i] := c;
  i := i + 1;
}
```

i: $[0, 2]$

j: $[0, 2]$

i + j: $[0, 4]$

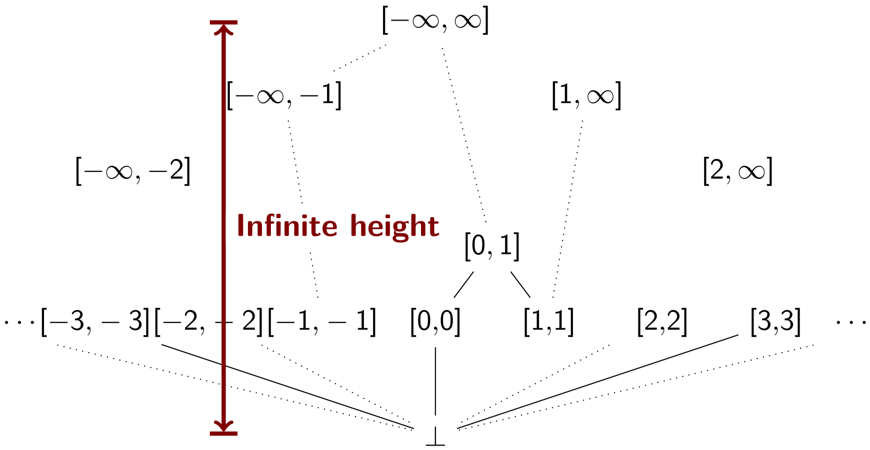
Out of bounds?

- Guarantee: $j < 3 - i$ $\implies j + i < 3$
- Array access is safe!
- Analysis must capture relations between variables
  - **Octagon Abstract Domain**

Numerical Domains

- **Interval Abstract Domain**
  - Constraints: $x \in [min_x, max_x]$
- **Octagon Abstract Domain**
  - Constraints: $\pm x \pm y \leq c$
  - (x, y variables, c constant number)
- **Polyhedra Abstract Domain**
  - $c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \leq c_0$
  - $c_1 x_1 + c_2 x_2 + \ldots + c_n x_n = c_0$
- Increasingly powerful, increasingly expensive to analyse

Interval Domain

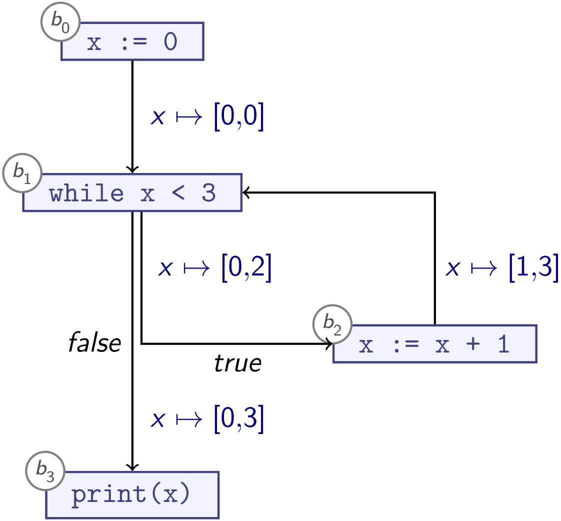


- $\top = [-\infty, \infty]$
- $[l_1, r_1] \sqcup [l_2, r_2] = [min(l_1, l_2), max(r_1, r_2)]$

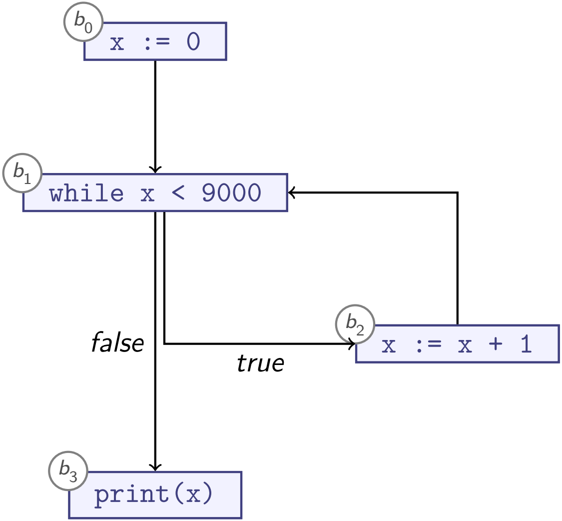
Summary

- Numerical Abstract Domains capture linear relations between variables and constants
  - **Interval Abstract Domain**: $x \in [min_x, max_x]$
  - Octagon Abstract Domain: $\pm x \pm y \leq c$
  - Polyhedra Abstract Domain: Arbitrary linear relationships
- Infinite Domain height: No termination guarantee with our current tools

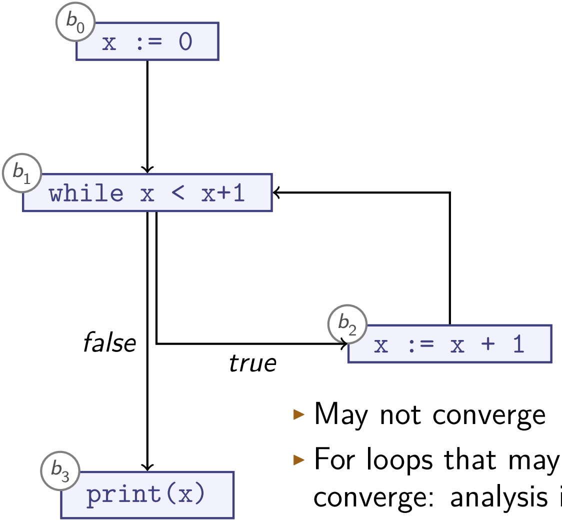
Applying the Interval Domain



Applying the Interval Domain

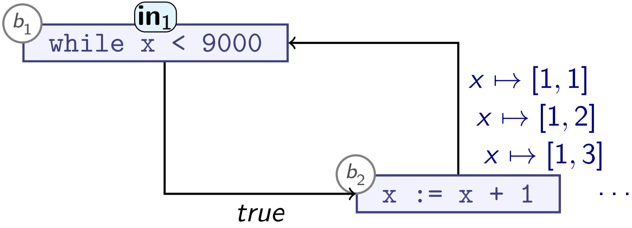


Applying the Interval Domain



- May not converge
- For loops that may take long to converge: analysis is slow

Widening



- Inefficient: no reason to assume 2, 3, ... will help us converge
- Detection: when updating $\mathbf{in}_1$:
  - Check if we have converged
  - Otherwise, **widen**

$$v_1 \nabla v_2 = \begin{cases} v_1 & \iff v_1 = v_2 \\ \mathbf{widen}(v_1 \sqcup v_2) & \iff v_1 \neq v_2 \end{cases}$$

  - For a suitable **widen** function

Widening Functions

- For convergence: satisfy Ascending Chain Condition on:

$$v_{i+1} = \textbf{widen}(v_i)$$

- Suitable functions for Interval Domain?
  - $\textbf{widen}_\top(v) = \top$
    - Very conservative
    - Ensures convergence
  - $\textbf{widen}_{10000}([l,r]) = [l - 10000, r + 10000]$
    - *No convergence*: still allows infinite ascending chain
  - $\textbf{widen}_\mathcal{K}([l,r]) = [max(\{v \in \mathcal{K} | v < l\}), min(\{v \in \mathcal{K} | v > r\})]$
    - Ensures convergence *iff $\mathcal{K}$ is finite*
    - Must pick "good" $\mathcal{K}$
    - Common strategy:
      $\mathcal{K} = \{-\infty, \infty\} \cup$ all numeric literals in program
      Our example: $\mathcal{K} = \{-\infty, 0, 1, 9000, \infty\}$

```
var x := 0;
while x < 9000 {
    x := x + 1;
}
```

Summary

- **Widening** allows us to use infinite domains $\mathcal{L}$
- Use **widen** function
  - **widen** must satisfy Ascending Chain Condition on $\mathcal{L}$
  - **widen**($\mathcal{L}$) generates finite lattice
- Widening operator ∇ applies **widen** function iff needed
- Approach:

1. Before analysis runs: we design analysis on infintie-height lattice
2. When analysis runs on concrete program:
3. **widen** constructs finite-height lattice specific to program
4. ∇ applies **widen** on demand
   - MFP: When updating: $\mathbf{in}_i := \mathbf{in}_i \nabla \mathbf{out}_j$

Inter- vs. Intra-Procedural Analysis

- **Intra**procedural: Within one procedure
  - Data flow analysis so far
- **Inter**procedural: Across multiple procedures
  - Type Analysis, especially. with polymorphic type inference

Limitations of Intra-Procedural Analysis

Teal-0

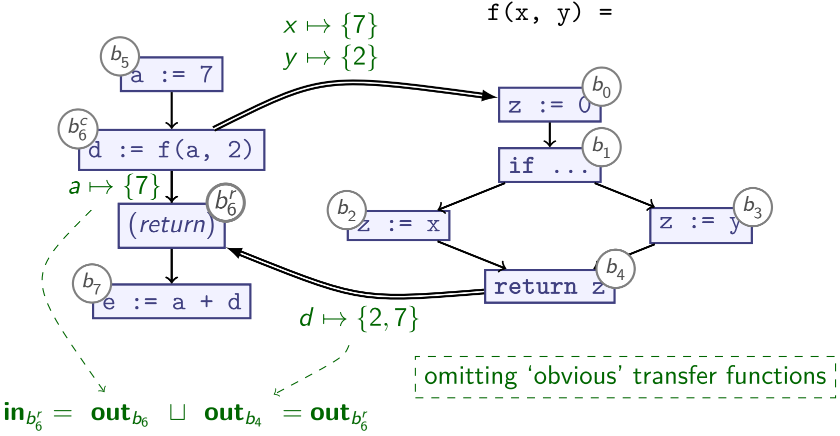
```
a := 7;
d := f(a, 2);
e := a + d;
```

Teal-0

```
fun f(x, y) = {
  z := 0;
  if x > y {
    z := x;
  } else {
    z := y;
  }
  return z;
}
```

How can we compute Reachable Definitions here?

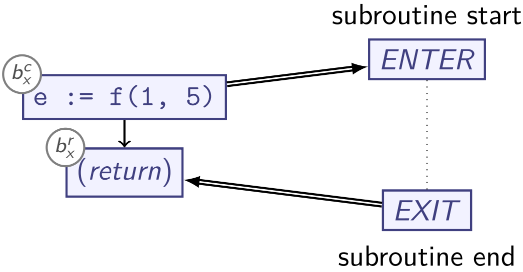
A Naïve Inter-Procedural Analysis



- $\mathbf{out}_{b_7}$: $e \mapsto \{9, 14\}$

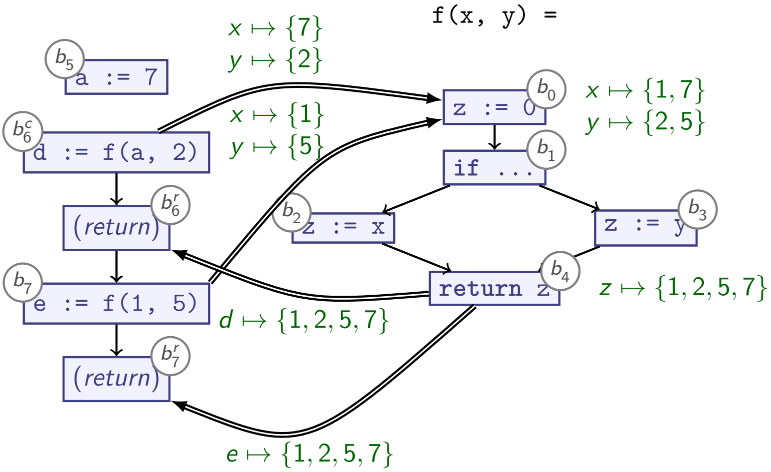
Works rather straightforwardly!

Inter-Procedural Data Flow Analysis



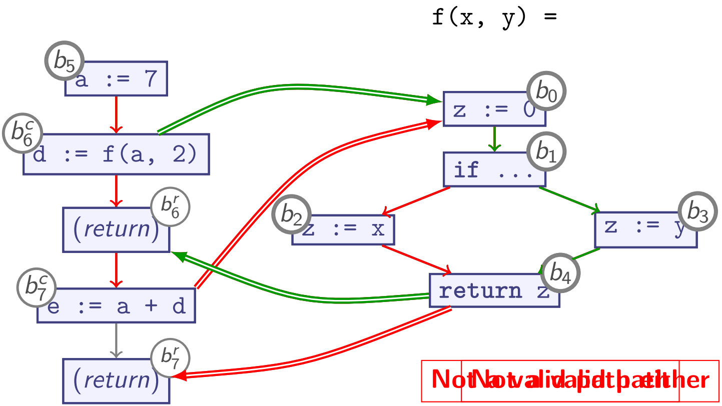
- Split call sites b_x into *call* (b_x^c) and *return* (b_x^r) nodes
- Intra-procedural edge $b_x^c \longrightarrow b_x^r$ carries environment/store
- Inter-procedural edge ($\Longrightarrow$):
  - Caller $\Longrightarrow$ subroutine, substitutes parameters (for pass-by-value)
  - Caller $\Longleftarrow$ return, substitutes result (for pass-by-result)
  - Otherwise as intra-procedural data flow edge

A Naïve Inter-Procedural Analysis



Imprecision!

Valid Paths



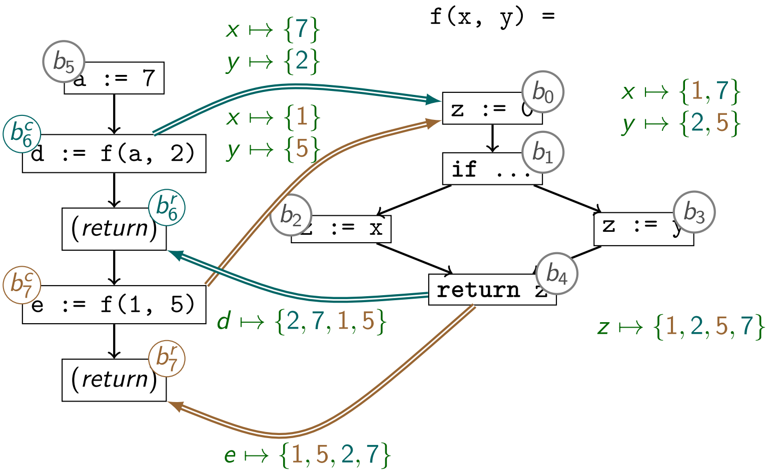
- $[b_5, b_6^c, b_0, b_1, b_3, b_4, b_6^r]$

Context-sensitive interprocedural analyses consider only valid paths

Summary

- **Intraprocedural** Data Flow Analysis is highly imprecise with subroutine calls
- **Interprocedural** Data Flow Analysis is more precise:
  - Split call site into call site + return site
  - Add flow edges between call sites, subroutine entry
  - Add flow edges between subroutine return, return site
  - Carry environment from call site to return site
- Interprocedural analysis must typically consider the entire program
  $\Rightarrow$ **whole-program analysis**
- Naïve interprocedural analysis is **call-site insensitive**
  - Merge all callers into one
  - Analyses paths that are not **valid** $\implies$ imprecision

Interprocedural Data Flow Analysis

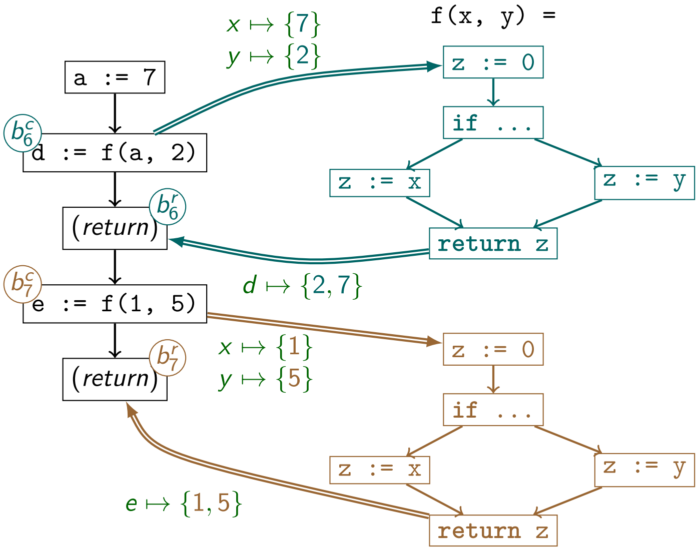


Call-site insensitive: **analysis merges all callers to** `f()`

Interprocedural Data Flow Analysis

- Call-site insensitive
- Call-site sensitive
  1. Via **Inlining** or **AST cloning**
  2. Via Call Strings

Inlining



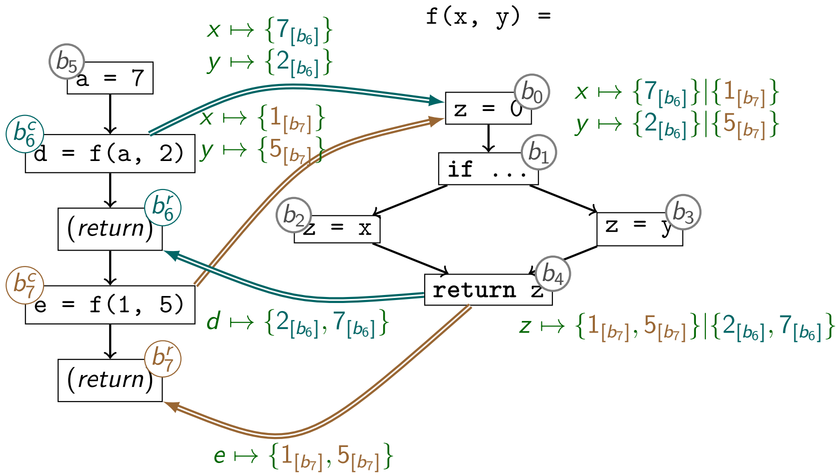
Clone subroutine IRs for each *calling context*

Interprocedural Data Flow Analysis

- Call-site insensitive
- Call-site sensitive

1. Via Inlining
2. Via **Call Strings**

Call Strings of Length 1



Degrees of Call-Site Sensitivity

- We used call sites to make call sites explicit:
  - $[b_6]$ in $2_{[b_6]}$
- Generalisation:
  - *Call Strings* support deeper nesting
  - Examples: $[b_0, b_6]$, $[b_1, b_6]$

Teal

```
fun g(y: int): int = { return y }
fun f(x: int): int = {
  return g(x) // b6
       + g(5); // b7
}
...
  f(1); // b0
  f(2); // b1
```

Must bound length of call strings to ensure termination

Summary

- Strategies for call-site sensitive analysis
- **Inlining**
  - Copy subroutine bodies for each caller
  - Not usually efficient, unless part of compiler backend (which has already decided to inline)
  - Problematic with recursion
- **Call Strings**
  - Call string length:
    - Unbounded: Maximum precision, may not terminate with recursion
    - Bounded to length k: k degrees of call site sensitivity (speed/precision trade-off)

Outlook

- No new homework this week
- Next Week: Heap Analysis

`http://cs.lth.se/EDAP15`