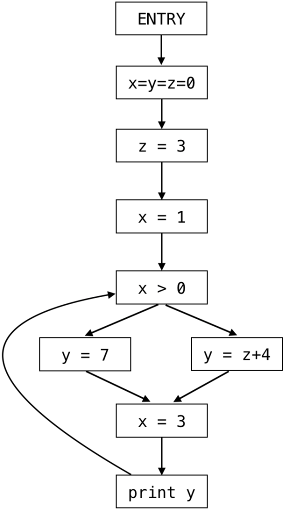
COSE312: Compilers

Lecture 21 — Data-Flow Analysis (3)

Hakjoo Oh
2017 Spring

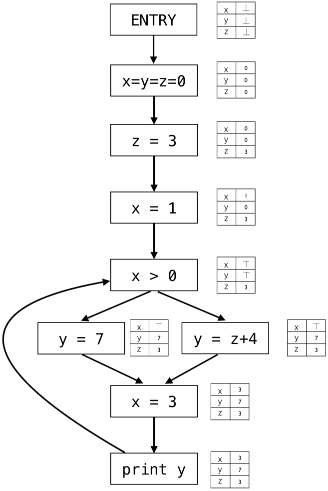
Constant Folding

Decide that the value of an expression is a constant and use it instead.

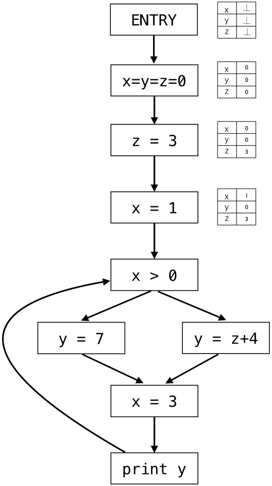


Constant Propagation Analysis

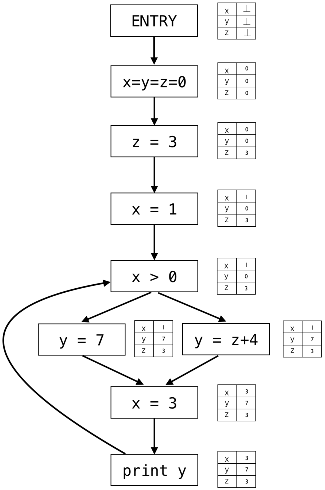
For each program point, determine whether a variable has a constant value whenever execution reaches that point.



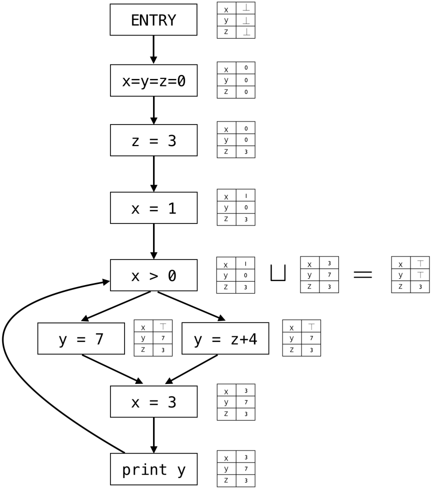
How It Works (1)



How It Works (2)



How It Works (3)

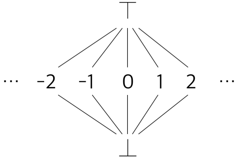


Constant Analysis

The goal is to compute

$$\begin{array}{rcl} \textbf{in} & : & Block \to (Var \to \mathbb{C}) \\ \textbf{out} & : & Block \to (Var \to \mathbb{C}) \end{array}$$

where $\mathbb{C}$ is a partially ordered set:



with the order:

$$\forall c_1, c_2 \in \mathbb{C}.\ c_1 \sqsubseteq c_2 \text{ iff } c_1 = \bot \ \vee \ c_2 = \top \ \vee \ c_1 = c_2$$

Functions in $Var \to \mathbb{C}$ are also partially ordered:

$$\forall d_1, d_2 \in (Var \to \mathbb{C}).\ d_1 \sqsubseteq d_2 \text{ iff } \forall x \in Var.\ d_1(x) \sqsubseteq d_2(x)$$

Join (Least Upper Bound)

The *join* between domain elements:

$$c_1 \sqcup c_2 = \begin{cases} c_2 & c_1 = \bot \\ c_1 & c_2 = \bot \\ c_1 & c_1 = c_2 \\ \top & \text{o.w.} \end{cases}$$

$$d_1 \sqcup d_2 = \lambda x \in \mathit{Var}.\ d_1(x) \sqcup d_2(x)$$

Transfer Function

The transfer function

$$f_B : (Var \to \mathbb{C}) \to (Var \to \mathbb{C})$$

models the program execution in terms of the abstract values: e.g.,

- Transfer function for $z = 3$:

$$\lambda d.\ [z \mapsto 3]d$$

- Transfer function for $x > 0$:

$$\lambda d.\ d$$

- Transfer function for $y = z + 4$:

$$\lambda d.\ \begin{cases} \bot & d(z) = \bot \\ \top & d(z) = \top \\ d(z) + 4 & \text{o.w.} \end{cases}$$

Transfer Function

A simple set of commands:

$$\begin{array}{rcl} c & \rightarrow & x := e \mid x > n \mid \\ e & \rightarrow & n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \end{array}$$

The transfer function:

$$\begin{array}{rcl} f_{x:=e}(d) & = & [x \mapsto [\![e]\!](d)]d \\ f_{x>n}(d) & = & d \end{array}$$

$$\begin{array}{rcl} [\![n]\!](d) & = & n \\ [\![x]\!](d) & = & d(x) \\ [\![e_1 + e_2]\!](d) & = & [\![e_1]\!](d) + [\![e_2]\!](d) \\ [\![e_1 - e_2]\!](d) & = & [\![e_1]\!](d) - [\![e_2]\!](d) \end{array}$$

Data-Flow Equations

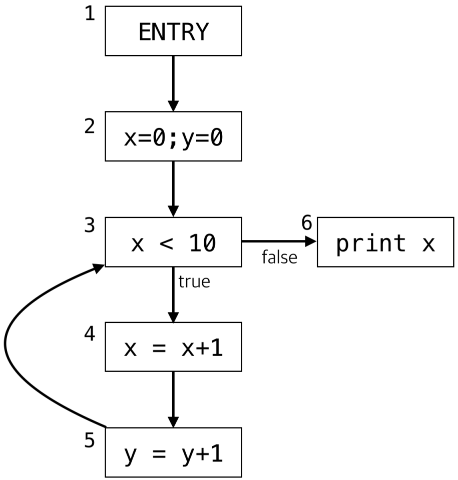
Equation:

$$\mathbf{in}(B) = \bigsqcup_{P \hookrightarrow B} \mathbf{out}(P)$$
$$\mathbf{out}(B) = f_B(\mathbf{in}(B))$$

Fixed point computation:

For all i, $\mathbf{in}(B_i) = \mathbf{out}(B_i) = \lambda x.\bot$
while (changes to any **in** and **out** occur) {
  For all i, update
    $\mathbf{in}(B_i) = \bigsqcup_{P \hookrightarrow B} \mathbf{out}(P)$
    $\mathbf{out}(B_i) = f_{B_i}(\mathbf{in}(B_i))$
}

Extension to Interval Analysis



Node	Result
1	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$
3	$x \mapsto [0, 9]$ $y \mapsto [0, +\infty]$
4	$x \mapsto [1, 10]$ $y \mapsto [0, +\infty]$
5	$x \mapsto [1, 10]$ $y \mapsto [1, +\infty]$
6	$x \mapsto [10, 10]$ $y \mapsto [0, +\infty]$

Applications of Interval Analysis

- Static buffer-overflow detection: e.g.,

  ```
  ... a[x] ...
  ```

  where $a.size = [10, 20]$ and $x = [5, 15]$.

  - E.g., Sparrow[1], Facebook Infer[2], etc use interval analysis to detect buffer overruns

- More precise constant analysis:

  ```
  if (...) {
    x = 1;  y = 2;
  } else {
    x = 2;  y = 1
  }
  z = x + y;
  ```

- Many others

[1] http://www.fasoo.com/

[2] http://fbinfer.com

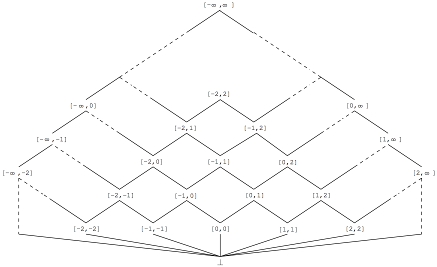
Interval Analysis

The goal is to compute

$$\begin{array}{rcl} \textbf{in} & : & Block \to (Var \to \mathbb{I}) \\ \textbf{out} & : & Block \to (Var \to \mathbb{I}) \end{array}$$

where $\mathbb{I}$ is a partially ordered set:

$$\mathbb{I} = \{\bot\} \cup \{[l,u] \mid l,u \in \mathbb{Z} \cup \{-\infty,+\infty\} \ \wedge \ l \leq u\}$$



with the order:

$$\forall c_1, c_2 \in \mathbb{C}.\ c_1 \sqsubseteq c_2 \text{ iff } c_1 = \bot \ \vee \ c_2 = \top \ \vee \ c_1 = c_2$$

Join (Least Upper Bound)

- The join operator merges multiple data flows: e.g.,
  - $[1,3] \sqcup [2,4] = [1,4]$
  - $[1,3] \sqcup [7,9] = [1,9]$
- Definition:

$$\begin{array}{rcl} \bot \sqcup i & = & i \\ i \sqcup \bot & = & i \\ [l_1, u_1] \sqcup [l_2, u_2] & = & [\min(l_1, l_2), \max(l_1, l_2)] \end{array}$$

Transfer Function

$$
\begin{array}{lcl}
c & \rightarrow & x := e \mid x > n \mid \\
e & \rightarrow & n \mid x \mid e_1 + e_2 \mid e_1 - e_2
\end{array}
$$

The transfer function:

$$
\begin{array}{rcl}
f_{x:=e}(d) & = & [x \mapsto [\![e]\!](d)]d \\
f_{x>n}(d) & = & [x \mapsto d(x) \sqcap [n+1, +\infty]]d \\
[\![n]\!](d) & = & [n, n] \\
[\![x]\!](d) & = & d(x) \\
[\![e_1 + e_2]\!](d) & = & [\![e_1]\!](d) \hat{+} [\![e_2]\!](d) \\
[\![e_1 - e_2]\!](d) & = & [\![e_1]\!](d) \hat{-} [\![e_2]\!](d)
\end{array}
$$

Data-Flow Equations

Equation:

$$\textbf{in}(B) = \bigsqcup_{P \hookrightarrow B} \textbf{out}(P)$$

$$\textbf{out}(B) = f_B(\textbf{in}(B))$$

Fixed point computation:

For all i, $\textbf{in}(B_i) = \textbf{out}(B_i) = \lambda x.\bot$
while (changes to any **in** and **out** occur) {
  For all i, update
    $\textbf{in}(B_i) = \bigsqcup_{P \hookrightarrow B} \textbf{out}(P)$
    $\textbf{out}(B_i) = f_{B_i}(\textbf{in}(B_i))$
}

Fixed Point Computation Does Not Terminate

The conventional fixed point computation requires an infinite number of iterations to converge:

Node	initial	1	2	3	10	11	k	∞
1	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$
3	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,1]$ $y \mapsto [0,1]$	$x \mapsto [0,2]$ $y \mapsto [0,2]$	$x \mapsto [0,9]$ $y \mapsto [0,9]$	$x \mapsto [0,9]$ $y \mapsto [0,10]$	$x \mapsto [0,9]$ $y \mapsto [0,k-1]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$
4	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [0,0]$	$x \mapsto [1,2]$ $y \mapsto [0,1]$	$x \mapsto [1,3]$ $y \mapsto [0,2]$	$x \mapsto [1,10]$ $y \mapsto [0,9]$	$x \mapsto [1,10]$ $y \mapsto [0,10]$	$x \mapsto [1,10]$ $y \mapsto [0,k-1]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$
5	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [1,1]$	$x \mapsto [1,2]$ $y \mapsto [1,2]$	$x \mapsto [1,3]$ $y \mapsto [1,3]$	$x \mapsto [1,10]$ $y \mapsto [1,10]$	$x \mapsto [1,10]$ $y \mapsto [1,11]$	$x \mapsto [1,10]$ $y \mapsto [1,k]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$
6	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto [0,0]$	$x \mapsto \bot$ $y \mapsto [0,1]$	$x \mapsto \bot$ $y \mapsto [0,2]$	$x \mapsto [10,10]$ $y \mapsto [0,9]$	$x \mapsto [10,10]$ $y \mapsto [0,10]$	$x \mapsto [10,10]$ $y \mapsto [0,k-1]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$

To ensure termination and precision, two staged fixed point computation is required:

1. increasing widening sequence
2. decreasing narrowing sequence

1. Fixed Point Computation with Widening

Node	initial	1	2	3
1	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$
3	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$
4	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [0,0]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$
5	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [1,1]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$
6	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto [0,0]$	$x \mapsto [10,+\infty]$ $y \mapsto [0,+\infty]$	$x \mapsto [10,+\infty]$ $y \mapsto [0,+\infty]$

2. Fixed Point Computation with Narrowing

Node	initial	1	2
1	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$
3	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$
4	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$
5	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$
6	$x \mapsto [10,+\infty]$ $y \mapsto [0,+\infty]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$

Summary

- Data-flow analyses we covered:
  - Reaching definitions analysis
  - Liveness analysis
  - Available expressions analysis
  - Constant propagation analysis
  - Interval analysis
- Static program analysis refers to a general method for automatic and sound approximation of sw run-time behaviors before the execution
  - "before": statically, without running sw
  - "automatic": sw analyzes sw
  - "sound": all possibilities into account
  - "approximation": cannot be exact
  - "general": for any source language and property
    - C, C++, C#, F#, Java, JavaScript, ML, Scala, JVM, x86, etc
    - "buffer-overrun?", "memory leak?", "type errors?", "x = y at line 2?", "memory use $\leq 2K$?", etc
- Foundational theory: Google "Abstract Interpretation"