Abstract Interpretation

(Galois, Collections, Widening)
What Are We Doing?

- Sam Block  
  Variable Renaming for Decompilation and Program Readability
- William Burns  
  Surveying Malware's Relation to Programming Languages
- Tim Chaplin  
  Generalizing the Burrows-Wheeler Transform
- Kirti Chawla  
  Detection of Covert Channels in RFID Supply Chains
- Joel Coffman
- Derek Davis  
  Classification of and Confidence in Repairs to Software Bugs
- Daniel Dougherty  
  Branch Prediction in Dynamic Binary Translation Systems
- Zak Fry  
  A Large-Scale Code Readability Metric
- Andrew Jurik  
  Surveying The Use of Static Analyses to Verify Security Properties
- Dan Lepage  
  A Bayesian Approach to SVBRDF Decomposition
- Ming Mao  
  A Process Execution and Collaboration Policy Assertion Language
- Irwin Reyes  
  The Accuracy of Face-Recognition Systems
- Arkaitz Ruiz Alvarez  
  Verifying Programs that Use Intel's Threading Building Blocks Library
- Mona Sergi  
  Improving Automatic Program Repair with Template-based Mutations
- Blake Sheridan  
  Implementing Support for Hardware-based Profiling in Embedded Systems
- Michael Skalak  
  Implementing and Interpreting Visual Languages
- Elizabeth Soechting  
  Semantic Regression Testing for Tree-Structured Output
- Luther Tychonievich  
  A Unified Compiler Representation to Support Debugging Optimized Code
- Kristen Walcott  
  A Test Case Generation Technique for Multithreaded Programs
- Ren Xu  
  Surveying Refinement Types
Tool Time

• How’s Homework 5 going?
• Get started early
• Compilation problems?
  - See FAQ
(trivia: what tool brand is this?)
More Power!

• You can handle it!
Abstract Interpretation

• We have an abstract domain $A$
  - e.g., $A = \{ \text{positive, negative, zero} \}$
  - An abstraction function $\beta : \mathbb{Z} \rightarrow A$
    • $\mathbb{Z}$ is our concrete domain
  - A concretization function $\gamma : A \rightarrow \mathcal{P}(\mathbb{Z})$

• Positive + Positive = ????
• Positive + Negative = ????
• Positive / Zero = ????
We don't want security to get suspicious ...
crazy delicious
Review

- We introduced **abstract interpretation**
- An abstraction mapping from concrete to abstract values
  - Has a concretization mapping which forms a Galois connection

- We’ll look a bit more at Galois connections
- We’ll lift AI from expressions to programs
- ... and we’ll discuss the mythic “widening”
Why Galois Connections?

- We have an abstract domain $A$
  - An abstraction function $\beta : \mathbb{Z} \to A$
  - Induces $\alpha : \mathcal{P}(\mathbb{Z}) \to A$ and $\gamma : A \to \mathcal{P}(\mathbb{Z})$

- We argued that for correctness
  \[ \gamma(a_1 \, \text{op} \, a_2) \supseteq \gamma(a_1) \, \text{op} \, \gamma(a_2) \]
  - We wish for the set on the left to be as small as possible
  - To reduce the loss of information through abstraction

- For each set $S \subseteq C$, define $\alpha(S)$ as follows:
  - Pick smallest $S'$ that includes $S$ and is in the image of $\gamma$
  - Define $\alpha(S) = \gamma^{-1}(S')$
  - Then we define: $a_1 \, \text{op} \, a_2 = \alpha(\gamma(a_1) \, \text{op} \, \gamma(a_2))$

- Then $\alpha$ and $\gamma$ form a Galois connection
Galois Connections

- A **Galois connection** between complete lattices $A$ and $\mathcal{P}(C)$ is a pair of functions $\alpha$ and $\gamma$ such that:
  - $\gamma$ and $\alpha$ are monotonic
  - (with the $\subseteq$ ordering on $\mathcal{P}(C)$)
  - $\alpha(\gamma(a)) = a$ for all $a \in A$
  - $\gamma(\alpha(S)) \supseteq S$ for all $S \in \mathcal{P}(C)$
More on Galois Connections

- All Galois connections are monotonic.
- In a Galois connection one function uniquely and absolutely determines the other.
Abstract Interpretation for Imperative Programs

• So far we abstracted the value of expressions
• Now we want to abstract the state at each point in the program
• First we define the concrete semantics that we are abstracting
  - We’ll use a collecting semantics
Collecting Semantics

• Recall
  - A state $\sigma \in \Sigma$. Any state $\sigma$ has type $\text{Var} \rightarrow \mathbb{Z}$
  - States vary from program point to program point

• We introduce a set of program points: labels

• We want to answer questions like:
  - Is $x$ always positive at label $i$?
  - Is $x$ always greater or equal to $y$ at label $j$?

• To answer these questions we’ll construct
  \[ C \in \text{Contexts}. \text{C has type } \text{Labels} \rightarrow \mathcal{P}(\Sigma) \]
  - For each label $i$, $C(i) =$ all possible states at label $i$
  - This is called the collecting semantics of the program
  - This is basically what SLAM (and BLAST, ESP, ...) approximate (using BDDs to store $\mathcal{P}(\Sigma)$ efficiently)
Defining the Collecting Semantics

- We first define relations between the collecting semantics at different labels
  - We do it for unstructured CFGs (cf. HW5!)
  - Can do it for IMP with careful notion of program points
- Define a label on each edge in the CFG
- For assignment

\[ C_j = \{ \sigma[x := n] \mid \sigma \in C_i \land \text{[e]}\sigma = n \} \]
Defining the Collecting Semantics

• For conditionals

\[
\begin{align*}
C_{\text{else}} &= \{ \sigma \mid \sigma \in C_{\text{in}} \land \llbracket b \rrbracket \sigma = \text{false} \} \\
C_{\text{then}} &= \{ \sigma \mid \sigma \in C_{\text{in}} \land \llbracket b \rrbracket \sigma = \text{true} \}
\end{align*}
\]

• Assumes b has no side effects (as in IMP or HW5)
Defining the Collecting Semantics

• For a join

\[ C_{out} = C_i \cup C_j \]

• Verify that these relations are monotonic
  - If we increase a \( C_x \) all other \( C_y \) can only increase
Collecting Semantics: Example

- Assume $x \geq 0$ initially (explain this?)

$$C_1 = \{\sigma \mid \sigma(x) \geq 0\}$$
Collecting Semantics: Example

- Assume $x \geq 0$ initially

\[ C_1 = \{ \sigma \mid \sigma(x) \geq 0 \} \]
\[ C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \} \]
Collecting Semantics: Example

- Assume $x \geq 0$ initially

$C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$

$C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}$

$C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}$
Collecting Semantics: Example

- Assume $x \geq 0$ initially

$
\begin{align*}
C_1 &= \{ \sigma \mid \sigma(x) \geq 0 \} \\
C_2 &= \{ \sigma[y:=1] \mid \sigma \in C_1 \} \\
C_3 &= C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \} \\
C_4 &= \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}
\end{align*}
$
Collecting Semantics: Example

- Assume $x \geq 0$ initially

$$C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$$

$$C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \} \cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \}$$

$$C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}$$

$$C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}$$
Collecting Semantics: Example

- Assume $x \geq 0$ initially

C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}

C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}

\cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \}

C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}

C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}

C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}
Why Does This Work?

• We just made a system of recursive equations that are defined largely in terms of themselves
  - e.g., $C_2 = F(C_4)$, $C_4 = G(C_3)$, $C_3 = H(C_2)$

• Why do we have any reason to believe that this will get us what we want?
The Collecting Semantics

• We have an equation with the unknown $C$
  - The equation is defined by a **monotonic** and **continuous** function on domain $\text{Labels} \rightarrow \mathcal{P}(\Sigma)$

• We can use the **least fixed-point theorem**
  - Start with $C^0(L)=\emptyset$ (aka $C^0 = \lambda L.\emptyset$)
  - Apply the relations between $C_i$ and $C_j$ to get $C^1_i$ from $C^0_j$
  - Stop when all $C^k = C^{k-1}$
  - Problem: we’ll go on forever for most programs
  - But we know the **fixed point exists**
Collecting Semantics: Example

• (assume $x \geq 0$ initially)

$y := 1$

$x \equiv 0$

$y := y \times x$

$x := x - 1$

$C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$

$C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}$

$C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}$

$C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}$

$C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}$
Collecting Semantics: Example

- (assume \( x \geq 0 \) initially)

\[
\begin{align*}
C_1 &= \{ \sigma \mid \sigma(x) \geq 0 \} \\
C_2 &= \{ \sigma[y:=1] \mid \sigma \in C_1 \} \\
    &\quad \cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \} \\
C_3 &= C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \} \\
C_5 &= C_2 \cap \{ \sigma \mid \sigma(x) = 0 \} \\
C_4 &= \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}
\end{align*}
\]
Collecting Semantics: Example

- (assume $x \geq 0$ initially)

```
1 { $x \geq 0$ }

y := 1

2 { $x \geq 0, y = 1$ }

x == 0

C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}
C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}
\quad \cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \}
C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}
C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}
C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}

y := y \times x

3

x := x - 1

4

T

5 \emptyset

\emptyset

\emptyset

\emptyset

\emptyset
```
Collecting Semantics: Example

• (assume $x \geq 0$ initially)

$y := 1$

$x \geq 0$

$y := y \times x$

$x := x - 1$

$C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$

$C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}$

$\cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \}$

$C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}$

$C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}$

$C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}$
Collecting Semantics: Example

• (assume $x \geq 0$ initially)

\[
C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}
\]
\[
C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}
\]
\[
\cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4 \}
\]
\[
C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}
\]
\[
C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3 \}
\]
\[
C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}
\]
Collecting Semantics: Example

• (assume $x \geq 0$ initially)

```
\(y := 1\)
\(x == 0\)
\(y := y \times x\)
\(x := x - 1\)
```

- $C_1 = \{ \sigma \mid \sigma(x) \geq 0\}$
- $C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1\}$
  \[ \cup \{ \sigma[x:=\sigma(x)-1] \mid \sigma \in C_4\} \]
- $C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0\}$
- $C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0\}$
- $C_4 = \{ \sigma[y:=\sigma(y)\times\sigma(x)] \mid \sigma \in C_3\}$
Collecting Semantics: Example

• (assume $x \geq 0$ initially)

```
1 { x \geq 0 }
```

```
y := 1
```

```
2 { x \geq 0, y = 1 \lor y = x + 1 }
```

```
x == 0
```

```
3 { x > 0, y = x + 1 }
```

```
y := y \times x
```

```
4 { x > 0, y = x }
```

```
x := x - 1
```

```
5 { x = 0, y = 1 }
```

$C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$

$C_2 = \{ \sigma[y:=1] \mid \sigma \in C_1 \}$

$C_3 = C_2 \cap \{ \sigma \mid \sigma(x) \neq 0 \}$

$C_4 = \{ \sigma[y:=\sigma(y) \times \sigma(x)] \mid \sigma \in C_3 \}$

$C_5 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}$
Q: Theatre (006 / 842)

- Name the 1879 Gilbert & Sullivan operetta parodied by the following quote:
  - I am the very model of a Newsgroup personality.
  - I intersperse obscenity with tedious banality.
  - Addresses I have plenty of, both genuine and ghosted too,
  - On all the countless newsgroups that my drivel is cross-posted to.
Q: Movies (387 / 842)

• Name the movie quoted below and also name either character or either character's actor. In this 1987 Mel Brooks spoof, one character is revealed to be another character's "father's brother's nephew's cousin's former roommate."
Q: TV Music (040 / 842)

• Fill in the three blanks in this Flintstones theme song snippet:
  - Let's ride with the family down the street
  - Through the courtesy of blank blank blank
  - When you're with the Flintstones
  - Have a yabba dabba doo time
Abstract Interpretation

• Pick a complete lattice $A$ (abstractions for $\mathcal{P}(\Sigma)$)
  - Along with a monotonic abstraction $\alpha : \mathcal{P}(\Sigma) \rightarrow A$
  - Alternatively, pick $\beta : \Sigma \rightarrow A$
  - This uniquely defines its Galois connection $\gamma$

• Take the relations between $C_i$ and move them to the abstract domain:
  \[ a : \text{Label} \rightarrow A \]

• Assignment

  Concrete: $C_j = \{ \sigma[x := n] \mid \sigma \in C_i \land [e]\sigma = n \}$

  Abstract: $a_j = \alpha \{ \sigma[x := n] \mid \sigma \in \gamma(a_i) \land [e]\sigma = n \}$
Abstract Interpretation

- Conditional

**Concrete:** \( C_j = \{ \sigma \mid \sigma \in C_i \land \llbracket b \rrbracket \sigma = \text{false} \} \) and
\( C_k = \{ \sigma \mid \sigma \in C_i \land \llbracket b \rrbracket \sigma = \text{true} \} \)

**Abstract:** \( a_j = \alpha \{ \sigma \mid \sigma \in \gamma(a_i) \land \llbracket b \rrbracket \sigma = \text{false} \} \) and
\( a_k = \alpha \{ \sigma \mid \sigma \in \gamma(a_i) \land \llbracket b \rrbracket \sigma = \text{true} \} \)

- Join

**Concrete:** \( C_k = C_i \cup C_j \)

**Abstract:** \( a_k = \alpha (\gamma(a_i) \cup \gamma(a_j)) = \text{lub \{a_i, a_j\}} \)
Least Fixed Points
In The Abstract Domain

• We have a recursive equation with unknown “a”
  - Defined by a monotonic and continuous function on the domain Labels \( \rightarrow A \)

• We can use the least fixed-point theorem:
  - Start with \( a^0 = \lambda L. \perp \) (aka: \( a^0(L) = \perp \))
  - Apply the monotonic function to compute \( a^{k+1} \) from \( a^k \)
  - Stop when \( a^{k+1} = a^k \)

• Exactly the same computation as for the collecting semantics
  - What is new?
  - “There is nothing new under the sun but there are lots of old things we don't know.” - Ambrose Bierce
Least Fixed Points
In The Abstract Domain

• We have a hope of termination!

• Classic setup: A has only uninteresting chains (finite number of elements in each chain)
  - A has finite height $h$ (= “finite-height lattice”)

• The computation takes $O(h \times |Labels|^2)$ steps
  - At each step “a” makes progress on at least one label
  - We can only make progress $h$ times
  - And each time we must compute $|Labels|$ elements

• This is a quadratic analysis: good news
  - This is exactly the same as Kildall’s 1973 analysis of dataflow’s polynomial termination given a finite-height lattice and monotonic transfer functions.
Abstract Interpretation: Example

- Consider the following program, $x > 0$

```plaintext
y := 1

if $x == 0$
    y := y * x
else
    x := x - 1
```

We want to do the \textit{sign analysis} on it.
Abstract Domain for Sign Analysis

- **Invent** the complete sign lattice
  \[ S = \{ \bot, -, 0, +, \top \} \]

- Construct the complete lattice
  \[ A = \{ x, y \} \rightarrow S \]
  - With the usual point-wise ordering
  - Abstract state gives the sign for x and y

- We start with \( a^0 = \lambda L. \lambda v \in \{x, y\}. \bot \)
  - aka: \( a^0(L, v) = \bot \)
## Let’s Do It!

<table>
<thead>
<tr>
<th>Label</th>
<th>Iterations →</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>y</td>
<td>T</td>
</tr>
</tbody>
</table>
Notes, Weaknesses, Solutions

• We abstracted the state of each variable independently

\[ A = \{x, y \} \rightarrow \{\bot, -, 0, +, \top\} \]

• We lost relationships between variables
  - e.g., at a point \(x\) and \(y\) may always have the same sign
  - In the previous abstraction we get \(\{x := \top, y := \top\}\) at label 2 (when in fact \(y\) is always positive!)

• We can also abstract the state as a whole

\[ A = \mathcal{P}(\{\bot, -, 0, +, \top\} \times \{\bot, -, 0, +, \top\}) \]
Other Abstract Domains

- **Range analysis**
  - Lattice of ranges: \( R = \{ \bot, [n..m], (-\infty, m], [n, +\infty), \top \} \)
  - It is a complete lattice
    - \([n..m] \sqcup [n’..m’] = [\min(n, n’)..\max(m,m’)]\)
    - \([n..m] \sqcap [n’..m’] = [\max(n, n’)..\min(m, m’)]\)
    - With appropriate care in dealing with \( \infty \)
  - \( \beta : \mathbb{Z} \to R \) such that \( \beta(n) = [n..n] \)
  - \( \alpha : \mathcal{P}(\mathbb{Z}) \to R \) such that \( \alpha(S) = \text{lub} \{ \beta(n) \mid n \in S \} = [\min(S)..\max(S)] \)
  - \( \gamma : R \to \mathcal{P}(\mathbb{Z}) \) such that \( \gamma(r) = \{ n \mid n \in r \} \)

- This lattice has **infinite-height chains**
  - So the abstract interpretation **might not terminate**!
Example of Non-Termination

- Consider this (common) program fragment

```
z := 1
z
·
n
z := z + 1
```

We want to do range analysis on it.
Example of Non-Termination

- Consider the sequence of abstract states at point 2
  - [1..1], [1..2], [1..3], ...
  - The analysis never terminates
  - Or terminates very late if the loop bound is known statically

- It is time to approximate even more: widening

- We redefine the join (lub) operator of the lattice to ensure that from [1..1] upon union with [2..2] the result is [1..+∞) and not [1..2]

- Now the sequence of states is
  - [1..1], [1, +∞), [1, +∞) Done (no more infinite chains)
Formal Definition of Widening
(Cousot 16.399 “Abstract Interpretation”, 2005)

• A widening \( \sqcup : (P \times P) \rightarrow P \) on a poset \( \langle P, \sqsubseteq \rangle \) satisfies:
  - \( \forall x, y \in P \cdot x \sqsubseteq (x \sqcup y) \land y \sqsubseteq (x \sqcup y) \)
  - For all increasing chains \( x^0 \sqsubseteq x^1 \sqsubseteq \ldots \) the increasing chain \( y^0 = \text{def} \ x^0, \ldots, y^{n+1} = \text{def} \ y^n \sqcup x^{n+1}, \ldots \) is not strictly increasing.

• Two different main uses:
  - Approximate missing lubs.  \( \text{(Not for us.)} \)
  - Convergence acceleration.  \( \text{(This is the real use.)} \)
    - A widening operator can be used to effectively compute an upper approximation of the least fixpoint of \( F \in L \sqcup L \) starting from below when \( L \) is computer-representable but does not satisfy the ascending chain condition.
## Formal Widening Example

\[ [1, 1] \triangledown [1, 2] = [1, +\infty) \]

- **Range Analysis on z:**

\[
\begin{align*}
\text{L0:} & \quad z := 1 ; \\
\text{L1:} & \quad \text{while } z < 99 \text{ do} \\
\text{L2:} & \quad z := z + 1 \\
\text{L3:} & \quad \text{done /* } z \geq 99 */
\end{align*}
\]

### Widened vs. Original Values

<table>
<thead>
<tr>
<th>Original ( x^i )</th>
<th>Widened ( y^i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x^{L0}_0 = \perp )</td>
<td>( y^{L0}_0 = \perp )</td>
</tr>
<tr>
<td>( x^{L1}_0 = [1, 1] )</td>
<td>( y^{L1}_0 = [1, 1] )</td>
</tr>
<tr>
<td>( x^{L2}_0 = [1, 1] )</td>
<td>( y^{L2}_0 = [1, 1] )</td>
</tr>
<tr>
<td>( x^{L3}_0 = [2, 2] )</td>
<td>( y^{L3}_0 = [2, 2] )</td>
</tr>
<tr>
<td>( x^{L2}_1 = [1, 2] )</td>
<td>( y^{L2}_1 = [1, +\infty) )</td>
</tr>
<tr>
<td>( x^{L3}_1 = [2, +\infty) )</td>
<td>( y^{L3}_1 = [2, +\infty) )</td>
</tr>
<tr>
<td>( x^{L4}_0 = [99, +\infty) )</td>
<td>( y^{L4}_0 = [99, +\infty) )</td>
</tr>
</tbody>
</table>

\( x^{L_i}_j \) = \text{def the } j\text{th iterative attempt to compute an abstract value for } z \text{ at label } L_i \)

Recall \( \text{lub } S = [\min(S) .. \max(S)] \)

\( \text{lub } \{[2, +\infty), [1, +\infty)\} = \{(1, +\infty)\} \)
Other Abstract Domains

• Linear relationships between variables
  - A convex polyhedron is a subset of $\mathbb{Z}^k$ whose elements satisfy a number of inequalities:
    \[ a_1 x_1 + a_2 x_2 + \ldots + a_k x_k \geq c_i \]
  - This is a complete lattice; linear programming methods compute lubs

• Linear relationships with at most two variables
  - Convex polyhedra but with $\leq 2$ variables per constraint
  - Octagons $(x \pm y \geq c)$ have efficient algorithms

• Modulus constraints (e.g. even and odd)
Abstract Chatter

• **AI, Dataflow and Software Model Checking**
  - The big three (aside from flow-insensitive type systems) for program analyses

• Are in fact quite related:
  - David Schmidt. *Data flow analysis is model checking of abstract interpretation*. POPL ’98.

• **AI is usually flow-sensitive** (per-label answer)

• **AI can be path-sensitive** (if your abstract domain includes $\lor$, for example), which is just where model checking uses BDD’s

• **Metal, SLAM, ESP, ...** can all be viewed as AI
Abstract Interpretation

Conclusions

• AI is a very powerful technique that underlies a large number of program analyses
• AI can also be applied to functional and logic programming languages
• There are a few success stories
  - Strictness analysis for lazy functional languages
  - PolySpace for linear constraints
• In most other cases however AI is still slow
• When the lattices have infinite height and widening heuristics are used the result becomes unpredictable