Program Analysis
Lecture 5

Rayna Dimitrova

WS 2016/2017
Recap: Constant propagation analysis

**Goal:** For each program point, determine whether a variable has a constant value whenever an execution reaches the respective block.

**Applications:** Useful for the **constant folding** optimization: a use of the variable can be replaced by the respective constant.

**Classification:** This is a **forward must** analysis.
Recap: Constant propagation analysis (lattice)

\[(D, \leq) := ((\text{Vars} \rightarrow (\mathbb{Z} \cup \{?\})) \cup \{\bot\}, \leq)\]

- \text{Vars} is the set of variables appearing in the program
- an element \(d \in (\text{Vars} \rightarrow (\mathbb{Z} \cup \{?\}))\) is a function \(d : \text{Vars} \rightarrow (\mathbb{Z} \cup \{?\})\) such that for a variable \(v \in \text{Vars}\)
  - \(d(v) \in \mathbb{Z}\) indicates that the value of \(v\) is the constant \(d(v)\)
  - \(d(v) = ?\) indicates that the variable \(v\) is non-constant
- the least element \(\bot\) indicates that no information is available
- the partial ordering \(\leq\) is defined by
  - \(\bot \leq d\) for all \(d \in D\),
  - \(d_1 \leq d_2\) if and only if for every \(v \in \text{Vars}\) we have that either \(d_1(v), d_2(v) \in \mathbb{Z}\) and \(d_1(v) = d_2(v)\), or it holds that \(d_2(v) = ?\)
Recap : Constant propagation analysis (lattice)

$$(D, \leq) := ((\text{Vars} \rightarrow (\mathbb{Z} \cup \{?\})) \cup \{\bot\}, \leq)$$

Then, the binary join operation $\sqcup$ is such that

- $\bot \sqcup d = d \sqcup \bot = d$, for all $d \in D$
- for all $d_1, d_2 \in D \setminus \{\bot\}$ and every $v \in \text{Vars}$,

$$(d_1 \sqcup d_2)(v) = \begin{cases} d_1(v) & \text{if } d_1(v) = d_2(v), \\ ? & \text{otherwise} \end{cases}$$

It is easy to check that $(D, \leq)$ is a complete lattice.
Recap: Constant propagation analysis (transfer functions)

To define the transfer functions, we first define abstract evaluation of arithmetic expressions, which is a function `eval` that maps each arithmetic expression `a` and `d ∈ D \ {⊥}` to an element of \( \mathbb{Z} \cup \{?\} \).

- if `a` is a variable `v`, then `eval(a, d) := d(v)`
- if `a` is a constant `k`, then `eval(a, d) := k`
- if `a = a_1 \otimes a_2`, then
  - `eval(a, d) := ?`, if `eval(a_1, d) = ?` or `eval(a_2, d) = ?`,
  - `eval(a, d) := ?`, if `eval(a_1, d) \otimes eval(a_2, d)` is undefined,
  - `eval(a, d) := eval(a_1, d) \otimes eval(a_2, d)`, otherwise

For `d ∈ (\text{Vars} → (\mathbb{Z} \cup \{?\}))`, `x, v ∈ \text{Vars}`, and `n ∈ \mathbb{Z} \cup \{?\}`, define the function `d[x ↦ n] : \text{Vars} → (\mathbb{Z} \cup \{?\})`, such that

\[
d[x ↦ n](v) = \begin{cases} 
  n & \text{if } v = x, \\
  d(v) & \text{otherwise.}
\end{cases}
\]
Recap: Constant propagation analysis (transfer functions)

- for \([x := a]^b\) let
  
  \[ f_b(d) := \begin{cases} 
  \bot & \text{if } d = \bot, \\
  d[x \mapsto eval(a, d)] & \text{otherwise}
  \end{cases} \]

- for \([\text{read}(x)]^b\) let
  
  \[ f_b(d) := \begin{cases} 
  \bot & \text{if } d = \bot, \\
  d[x \mapsto ?] & \text{otherwise}
  \end{cases} \]

- for \([\text{print}(a)]^b\), \([\text{skip}]^b\), and for condition \([e]^b\) let
  
  \[ f_b(d) = d \text{ for all } d \in D \]
Recap: Constant propagation analysis (example)

Let's consider an example to illustrate constant propagation analysis.

\[ z > 0 \]
\[ x := 2 \]
\[ x := 3 \]
\[ y := 3 \]
\[ y := 2 \]
\[ z := x + y \]
\[ \text{[skip]} \]

**Initial Function Values:**
\[ f_b(\bot) = \bot \text{ for } b \in \{1, \ldots, 7\} \]

**Function Definitions:**
\[ f_1(d) = d \]
\[ f_2(d) = d[x \mapsto 2] \]
\[ f_3(d) = d[y \mapsto 3] \]
\[ f_4(d) = d[x \mapsto 3] \]
\[ f_5(d) = d[y \mapsto 2] \]
\[ f_6(d) = d[z \mapsto \text{eval}(x, d) + \text{eval}(y, d)] \]
\[ f_7(d) = d \]
Recap: Constant propagation analysis (example)

\((n_1, n_2, n_3)\) denotes the function \(d(x) = n_1, d(y) = n_2, d(z) = n_3\)

\[
\begin{align*}
X_1 & = (?, ?, ?) \\
X_2 & = f_1(X_1) \\
X_3 & = f_2(X_2) \\
X_4 & = f_1(X_1) \\
X_5 & = f_4(X_4) \\
X_6 & = f_3(X_3) \sqcup f_5(X_5) \\
X_7 & = f_6(X_6)
\end{align*}
\]

least fixpoint

\(((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), (?, ?, ?), (?, ?, ?))\)

From this solution we cannot infer the information that at the beginning of block 7 the value of \(z\) is 5 regardless of the execution path.
Example: precision of the least fixpoint solution

\[ z > 0 \]

\[ x := 2 \]

\[ y := 3 \]

\[ z := x + y \]

\[ [z := x + y] \]

\[ [x := 3] \]

\[ [y := 2] \]

\[ [skip] \]

\[ f_6(d) = d[z \mapsto eval(x, d) + eval(y, d)] \]

For the least fixpoint we compute

\[ f_6((2, 3, ?) \sqcup (3, 2, ?)) = f_6((?, ?, ?)) = (?, ?, ?) \]

On the other hand

\[ f_6((2, 3, ?)) \sqcup f_6((3, 2, ?)) = (2, 3, 5) \sqcup (3, 2, 5) = (?, ?, 5) \]
Join Over all Paths (JOP) solution

Let $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

A different solution method, called **Join Over all Paths (JOP)** propagates analysis information along paths in the program.

Analysis information for block $b$:

- join over all execution paths leading to $b$,
- most precise information for $b$ (reference solution)

**Remark:**
In the literature often you see Meet Over all Paths (MOP).
Join Over all Paths (JOP) solution

Let $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

A path to the entry of a block $b$ from the initial block is a list of blocks traversed from the initial block to $b$ (excluding $b$).

$$Paths(b) = \{ [b_1, \ldots, b_{n-1}] \mid n \geq 1, b_1 \text{ is initial, } b_n = b, \forall i < n : b_{i+1} \in \text{succ}(b_i) \}$$

We say that a path $[b_1, \ldots, b_{n-1}]$ has length $n - 1$.

Remark: For backward analysis we can consider paths to the exit of block $b$ that follow reverse edges starting from the final node.

$$Paths(b) = \{ [b_1, \ldots, b_{n-1}] \mid n \geq 1, b_1 \text{ is final, } b_n = b, \forall i < n : b_{i+1} \in \text{pred}(b_i) \}$$
Join Over all Paths (JOP) solution

Let $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

With a path $\pi = [b_1, \ldots, b_k]$ we associate the transfer function

$$f_{\pi} = f_{b_k} \circ f_{b_{k-1}} \circ \ldots \circ f_{b_1} \circ id.$$ 

Thus, for the empty path $\varepsilon$, the transfer function $f_{\varepsilon} = id$ is identity.

The JOP solution for $b \in B$ (up to but not including $b$) is defined as

$$X_{b}^{JOP} := \bigsqcup \{f_{\pi}(i) \mid \pi \in Paths(b)\}.$$ 

The JOP solution of $S$ is

$$JOP(S) := (X_{1}^{JOP}, \ldots, X_{|B|}^{JOP}).$$
JOP solution: example

For block 7 we have

\[ \text{Paths}(7) = \{[1, 2, 3, 6], [1, 4, 5, 6]\}. \]

\[
\begin{align*}
&f_{[1,2,3,6]}((?, ?, ?,)) = \\
&f_6(f_3(f_2(f_1(id((?, ?, ?,)))),)) = \\
&f_6(f_3(f_2(f_1((?, ?, ?,)))),) = \\
&f_6(f_3(f_2((?, ?, ?,))),) = \\
&f_6((2, ?, ?)) = \\
&(2, 3, 5)
\end{align*}
\]

Similarly we compute

\[
\begin{align*}
&f_{[1,4,5,6]}((?, ?, ?,)) = (3, 2, 5)
\end{align*}
\]
JOP solution: example

For block 7 we have

\[\text{Paths}(7) = \{[1, 2, 3, 6], [1, 4, 5, 6]\}.\]

\[X_7^{JOP} = f_{[1,2,3,6]}((?, ?, ?)) \sqcup f_{[1,4,5,6]}((?, ?, ?))\]

\[= (2, 3, 5) \sqcup (3, 2, 5)\]

\[= (?, ?, 5)\]

Recall that \(X_7^{LFP} = (?, ?, ?)\).

Thus, \(X_7^{JOP} \leq X_7^{LFP}\).

Note: In general, the set of paths in a program is infinite.
Least fixpoint as an approximation of JOP

Let $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system. Let $\text{lfp}(S) = (X_1^{LFP}, \ldots, X_{|B|}^{LFP})$ be the least fixpoint solution of $S$.

Then, $\text{lfp}(S)$ is a sound approximation of $\text{JOP}(S)$.

**Theorem**

For every dataflow system $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$

1. it holds for all $b \in B$ that $X_b^{\text{JOP}} \leq X_b^{LFP}$.
2. if all functions $f_b$ for $b \in B$ are distributive (that is, for all $d_1, d_2 \in D$, $f_b(d_1) \sqcup f_b(d_2) = f_b(d_1 \sqcup d_2)$), then for all $b \in B$ it holds that $X_b^{\text{JOP}} = X_b^{LFP}$, i.e., both solutions coincide.
Proof of (1)

By definition we have $X_b^{JOP} = \bigsqcup \{ f_\pi(i) \mid \pi \in \text{Paths}(b) \}$.

Let us define $X_b^{JOP,n} := \bigsqcup \{ f_\pi(i) \mid \pi \in \text{Paths}(b), |\pi| \leq n \}$, for the paths of length less than or equal to $n$ for $n \in \mathbb{N}$.

Then $X_b^{JOP} = \bigsqcup \{ X_b^{JOP,n} \mid n \in \mathbb{N} \}$, and so, we can show $X_b^{JOP} \leq X_b^{LFP}$ by proving that for all $n \in \mathbb{N}$, $X_b^{JOP,n} \leq X_b^{LFP}$.

We show by induction on $n$ that for all $n \in \mathbb{N}$, for all $b \in B$ it holds that $X_b^{JOP,n} \leq X_b^{LFP}$.

For $n = 0$ there are two possible cases

1) There is no path of length 0 in Paths$(b)$.
   Then $X_b^{JOP,0} = \bigsqcup \emptyset = \bot \leq X_b^{LFP}$.

2) There exists a path of length 0 in Paths$(b)$.
   Then $b$ is an extremal node, and thus, $X_b^{LFP} = i$.
   We have $X_b^{JOP,0} = f_\varepsilon(i) = id(i) = i$. Thus, $X_b^{JOP,0} = X_b^{LFP}$. 
Proof of (1) continued

Induction hypothesis: for some \( n \) and all \( b \in B \), \( X_{b}^{\text{JOP}, n} \leq X_{b}^{\text{LFP}} \).

If \( b \) is an extremal node, the only path is the empty path.
Now, suppose that \( b \) is not extremal. Then,

\[
X_{b}^{\text{LFP}} = \bigsqcup \{ f_{p}(X_{p}^{\text{LFP}}) \mid p \in \text{pred}(b) \} \quad (X^{\text{LFP}} \text{ is a solution})
\]
\[
\geq \bigsqcup \{ f_{p}(X_{p}^{\text{JOP}, n}) \mid p \in \text{pred}(b) \} \quad (IH, f_{p} \text{ monotonic})
\]
\[
= \bigsqcup \{ f_{p}(\bigsqcup \{ f_{\pi}(i) \mid \pi \in \text{Paths}(p), |\pi| \leq n \}) \mid p \in \text{pred}(b) \} \quad \text{(definition of } X_{p}^{\text{JOP}, n})
\]
\[
\geq \bigsqcup \{ \bigsqcup \{ f_{p}(f_{\pi}(i)) \mid \pi \in \text{Paths}(p), |\pi| \leq n \} \mid p \in \text{pred}(b) \} \quad (f_{p} \text{ is monotonic and thus, } f_{p}(d_{1}) \sqcup f_{p}(d_{2}) \leq f_{p}(d_{1} \sqcup d_{2}))
\]
\[
= \bigsqcup \{ f_{p}(f_{\pi}(i)) \mid \pi \in \text{Paths}(p), |\pi| \leq n, p \in \text{pred}(b) \}
\]
\[
= \bigsqcup \{ f_{\pi'}(i) \mid \pi' \in \text{Paths}(b), |\pi'| \leq n + 1 \} \quad (\pi' = \pi \cdot p \text{ is path of length } n + 1 \text{ to } b; f_{\pi'} = f_{p} \circ f_{\pi})
\]
\[
= X_{b}^{\text{JOP}, n+1} \quad \text{(definition of } X_{p}^{\text{JOP}, n+1})
\]
Distributive frameworks

A dataflow system $S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B})$ in which all $f_b$ are distributive are called a **distributive framework**.

The Reaching Definitions, Available Expressions, Live Variables, and Very Busy Expressions analyses are all distributive frameworks.

For them, the least fixpoint computation delivers the *JOP* solution.

Constant Propagation is **not** a distributive framework.
Theorem The problem of computing the $JOP$ solution for the Constant Propagation analysis is undecidable.

Proof reduction from modified Post Correspondence Problem
Modified Post Correspondence Problem

Let $\Gamma$ be a finite alphabet with at least 2 symbols.

given lists of finite words $u_1, \ldots, u_n \in \Gamma^+$ and $v_1, \ldots, v_n \in \Gamma^+$

determine whether there exists a sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $m \geq 1$, $i_1 = 1$, and

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

Example

$\Gamma = \{a, b\}$

$u_1 = bba$, $u_2 = ab$, $u_3 = a$

$u_1 = bb$, $u_2 = aa$, $u_3 = baa$

$1, 2, 1, 3$ is a solution: $bbaabbbaa = bbaabbbaa$
Modified Post Correspondence Problem

Let $\Gamma$ be a finite alphabet with at least 2 symbols.

given lists of finite words $u_1, \ldots, u_n \in \Gamma^+$ and $v_1, \ldots, v_n \in \Gamma^+$

determine whether there exists a sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $m \geq 1$, $i_1 = 1$, and

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

Remark: Fixing $i_1 = 1$ does not affect undecidability.

The (modified) Post Correspondence Problem is undecidable.
Proof of the Theorem

**Theorem** The problem of computing the $JOP$ solution for the Constant Propagation analysis is undecidable.

**Proof**

Let $\Gamma = \{1, \ldots, 9\}$ and for $u \in \Gamma^+$, let

- $|u|$ be the length of $u$ when interpreted as a string in $\Gamma^+$,
- $[u]$ be the value of $u$ when interpreted as a natural number,

Given lists of finite words $u_1, \ldots, u_n \in \Gamma^+$ and $v_1, \ldots, v_n \in \Gamma^+$ we construct a program for which computing the $JOP$ solution for Constant Propagation allows us to determine if the modified Post correspondence problem has a solution for this lists of words.
Proof of the Theorem (cont.)

\[ x := \lfloor u_1 \rfloor; \]
\[ y := \lfloor v_1 \rfloor; \]
\[ \textbf{while} \; [...] \; \textbf{do} \; \{ \]
\[ \quad \textbf{if} \; [...] \; \textbf{then} \; \{ \]
\[ \quad \quad x := x \times 10^{\lfloor u_1 \rfloor} + \lfloor u_1 \rfloor; \]
\[ \quad \quad y := y \times 10^{\lfloor v_1 \rfloor} + \lfloor v_1 \rfloor \} \]
\[ \quad \textbf{else if} \; [...] \; \textbf{then} \; \{ \]
\[ \quad \quad x := x \times 10^{\lfloor u_2 \rfloor} + \lfloor u_2 \rfloor; \]
\[ \quad \quad y := y \times 10^{\lfloor v_2 \rfloor} + \lfloor v_2 \rfloor \} \]
\[ \quad \ldots \]
\[ \quad \textbf{else if} \; [...] \; \textbf{then} \; \{ \]
\[ \quad \quad x := x \times 10^{\lfloor u_n \rfloor} + \lfloor u_n \rfloor; \]
\[ \quad \quad y := y \times 10^{\lfloor v_n \rfloor} + \lfloor v_n \rfloor \} \]
\[ \} \]
\[ [z := \text{sign}((x - y) \times (x - y))]^{l-1}; \]
\[ [\text{skip}]^l; \]

We assume that our programming language contains

- arithmetic expressions \( a_1^{a_2} \), for \( a_1, a_2 \) arithmetic expressions
- the function \text{sign}, where

\[
\text{sign}(z) = \begin{cases} 
-1 & \text{if } z < 0 \\
0 & \text{if } z = 0 \\
1 & \text{if } z > 0
\end{cases}
\]

Note: We omit the conditions of \textbf{while} and \textbf{if} statements, as they are ignored by Constant Propagation.
Proof of the Theorem (cont.)

The set $Paths(l)$ consists of execution paths corresponding to all possible pairs of words $x \in \Gamma^+$ and $y \in \Gamma^+$ of the form

$$x = u_{i_1} u_{i_2} \ldots u_{i_m} \text{ and } y = v_{i_1} v_{i_2} \ldots v_{i_m}$$

for sequences $i_1, \ldots, i_m$ of indices where $i_1 = 1$ and $m \geq 1$.

For each such path

- if $x = y$ then $z$ is set to 0 in block $l - 1$,
- if $x \neq y$ then $z$ is set to 1 in block $l - 1$.

Hence, the modified Post correspondence problem has no solution if and only if $z$ is set to 1 on all execution paths reaching block $l$.

If $X_{l}^{JOP} = d$ is the JOP solution for program location $l$, then $d(z) = 1$ if and only if the modified Post correspondence problem has no solution for the lists of words $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$. 
Thus, if there exists an algorithm for computing the $JOP$ solution for Constant Propagation, then we can construct an algorithm for solving the modified Post correspondence problem. This implies that the problem of computing the $JOP$ solution for Constant Propagation is undecidable, since the modified Post correspondence problem is undecidable.
Conditional branches

So far: values of conditions have been ignored by the analysis. Essentially, we have been treating if and while statements as nondeterministic choice between the two branches.

Solution: introduce transfer functions for branches

Possible approaches:

1. attach conditions as edge labels in the CFG
   - advantage: no language modification required
   - disadvantage: requires modification to the analysis framework

2. encode conditions as assert statements
   - advantage: no modification of the analysis framework
   - disadvantage: requires extension of the programming language
Taking into account conditional branches

approach 1:

\[
\begin{align*}
&[w = 0]^1 \\
&z := 0^2 \\
&w = 0 \\
&\neg(w = 0) \\
&z := 1^3 \\
&[\text{skip}]^4
\end{align*}
\]

approach 2:

\[
\begin{align*}
\text{if } [w = 0]^1 & \{ \\
&[\text{assert}(w = 0)]^2; \\
&z := 0^3; \\
&\} \\
\text{else } & \{ \\
&[\text{assert}(\neg(w = 0))]^4; \\
&z := 1^5; \\
&\} \\
&[\text{skip}]^6
\end{align*}
\]
While programs with assertions

**arithmetic expressions**, which denote integer values

\[ a ::= k | x | a_1 + a_2 | a_1 - a_2 | a_1 \times a_2 | a_1 / a_2 \]

\( k \) is an integer constant, \( x \) is a variable

**Boolean expressions**

\[ b ::= a_1 = a_2 | a_1 < a_2 | \neg b | b_1 \land b_2 | b_1 \lor b_2 | \text{true} \]

**programs**

\[ c ::= \langle \text{skip} \rangle | \langle x := a \rangle | \langle \text{read}(x) \rangle | \langle \text{print}(a) \rangle | c_1 ; c_2 \]

| if \( b \) then \{c_1\} else \{c_2\} |

| while \( b \) do \{c\} | \langle \text{assert}(b) \rangle |

\( x \) is a variable, \( a \) is arithmetic expression, \( b \) is a Boolean expression
assert \( (b) \) aborts the execution if \( b \) is false, and has no effect otherwise. We will only insert it where it is guaranteed to hold.

The definition of transfer functions for assert blocks depends on the analysis problem. Often, insight into the analysis problem is required to define non-trivial and sound constraints for assert.

Idea: use assertions as filters that only let valid information pass
Constant propagation analysis with assertions

extend the evaluation function $eval$ to boolean expressions: maps a boolean expression $b$ and $d \in D \setminus \{\bot\}$ to an element of $\mathbb{B}$

- if $b$ is true, then $eval(b, d) := true$,
- if $b$ is $a_1 = a_2$, then $eval(b, d) := true$, if $eval(a_1, d) = eval(a_2, d)$, and $eval(b, d) := false$, otherwise
- if $b$ is $\neg b'$, then $eval(b, d) := true$, if $eval(b', d) = false$, and $eval(b, d) := false$, otherwise
- if $b$ is $b_1 \land b_2$, then $eval(b, d) := true$, if $eval(b_1, d) = eval(b_2, d) = true$, and $eval(b, d) := false$, otherwise

Remark For simplicity, the definition of $eval(a_1 = a_2)$ assumes that the value of both expressions $a_1$ and $a_2$ is defined (this holds for expressions that do not contain division). To handle operations with undefined value (e.g., division by zero) the range of $eval$ should be extended with the value $\bot$, and the value $\bot$ should be taken into consideration in the transfer function for $assert$. 
Constant propagation analysis with assertions

for $[\text{assert}(b)]^l$ we define the transfer function $f_l : D \rightarrow D$

$$f_l(d) = \bot \quad \text{if } d = \bot$$

$$f_l(d) = \bot \quad \text{if } \neg (\exists d' \in \alpha(d) : \text{eval}(b, d') = \text{true})$$

$$f_l(d)(v) = n \quad \text{if } (\forall d' \in \alpha(d) : \text{eval}(b, d') = \text{true} \Rightarrow d'(v) = n)$$

$$f_l(d)(v) = ? \quad \text{otherwise}$$

where $\alpha(d)$ is the set of functions from $\text{Vars}$ to $\mathbb{Z}$ such that $d' \in \alpha(d)$ if and only if for all $x \in \text{Vars}$ it holds that

$$d'(x) \in \begin{cases} \{n\} & \text{if } d(x) = n \in \mathbb{Z} \\ \mathbb{Z} & \text{if } d(x) = ? \end{cases}$$
Example: Constant propagation analysis with assertions

\( (d_x, d_y) \): represents function \( d \) with \( d(x) = d_x \) and \( d(y) = d_y \)

consider \( d = (0, ?) \)

\( \alpha(d) = \{(0, m) \mid m \in \mathbb{Z}\} \)

\( d' \in \alpha(d) \) if and only if for all \( v \in \{x, y\} \) it holds that

\[
d'(v) \in \begin{cases} \{n\} & \text{if } d(v) = n \in \mathbb{Z} \\ \mathbb{Z} & \text{if } d(v) = ? \end{cases}
\]
Example: Constant propagation analysis with assertions

\[(d_x, d_y): \text{ represents function } d \text{ with } d(x) = d_x \text{ and } d(y) = d_y\]

consider \(d = (0, ?)\)

\[\alpha(d) = \{(0, m) \mid m \in \mathbb{Z}\}\]

\[\forall d' \in \alpha(d): \text{eval}(x \geq 0, d') = \text{true}\]
\[\forall d' \in \alpha(d): \text{eval}(x < 0, d') = \text{false}\]

\[f_3(d)(x) = 0, \ f_3(d)(y) = ?\]
\[f_4(d) = \bot\]

\[f_5(f_3(d)) = f_5((0, ?)) = (0, 0)\]
\[f_6(f_4(d)) = f_6(\bot) = \bot\]