Data-flow Analysis / Abstract Interpretation

Deepak D’Souza and K. V. Raghavan

IISc
What is data-flow analysis

“Computing ‘safe’ approximations to the set of values / behaviours arising dynamically at run time, statically or at compile time.”

Typically used by compiler writers to optimize running time of compiled code.
  - Constant propagation: Is the value of a variable constant at a particular program location.
  - Replace $x := y + z$ by $x := 17$ during compilation.

More recently, used for verifying properties of programs.
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path Concrete states
I \{(i,j)\} (given)
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path  Concrete states
I  \{ (i, j) \} (given)
IA  \{ (i, j) | i \text{ is odd}, j \text{ is even} \}

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path       Concrete states
I          \{(i,j)\} (given)
IA         \{(i,j)|i \text{ is odd, } j \text{ is even}\}
IAB        \{(i,j)|i \text{ is odd, } j \text{ is even}\}

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path | Concrete states
--- | ---
I | \{(i,j)\} (given)
IA | \{(i,j)|i is odd, j is even\}
IAB | \{(i,j)|i is odd, j is even\}
IABC | \{(i,j)|i is odd, j is even, i > j\}

Deepak D'Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting semantics — example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path | Concrete states
-----|------------------
I    | \{(i,j)\} (given)
IA   | \{(i,j)|i is odd, j is even\}
IAB  | \{(i,j)|i is odd, j is even\}
IABC | \{(i,j)|i is odd, j is even, i > j\}
IABCD| \{(i,j)|i is even, j is even, i > j + 1\}

Deepak D'Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path    Concrete states
I       \{ (i, j) \} (given)
IA      \{ (i, j) \mid i \text{ is odd, } j \text{ is even} \}
IAB     \{ (i, j) \mid i \text{ is odd, } j \text{ is even} \}
IABC    \{ (i, j) \mid i \text{ is odd, } j \text{ is even, } i > j \}
IABCD   \{ (i, j) \mid i \text{ is even, } j \text{ is even, } i > j + 1 \}
IABCDE  \{ (i, j) \mid i \text{ is even, } j \text{ is even, } i \geq j \}

Diagram:

- Path I
- Path IA
- Path IAB
- Path IABC
- Path IABCD
- Path IABCDE

States:
- p = odd()
- q = even()
- p > q
- p := p + 1
- q := q + 2
- print p, q
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

```
Path                   Concrete states
I                      \{(i,j)\} (given)
IA                     \{(i,j)|i is odd, j is even\}
IAB                    \{(i,j)|i is odd, j is even\}
IABC                   \{(i,j)|i is odd, j is even, i > j\}
IABCD                  \{(i,j)|i is even, j is even, i > j + 1\}
IABCDE                 \{(i,j)|i is even, j is even, i ≥ j\}
IABCDEB                \{(i,j)|i is even, j is even, i ≥ j\}
```
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path                  Concrete states
I                     \{ (i, j) \} (given)
IA                    \{ (i, j) | i is odd, j is even \}
IAB                   \{ (i, j) | i is odd, j is even \}
IABC                  \{ (i, j) | i is odd, j is even, i > j \}
IABCD                 \{ (i, j) | i is even, j is even, i > j + 1 \}
IABCDE                \{ (i, j) | i is even, j is even, i ≥ j \}
IABCDEB               \{ (i, j) | i is even, j is even, i ≥ j \}
IABCDEB                   \{ (i, j) | i is even, j is even, i > j \}

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path     Concrete states
I       \{(i,j)\} (given)
IA      \{(i,j)|i is odd, j is even\}
IAB     \{(i,j)|i is odd, j is even\}
IABC    \{(i,j)|i is odd, j is even, i > j\}
IABCD   \{(i,j)|i is even, j is even, i > j + 1\}
IABCDEFG \{(i,j)|i is even, j is even, i ≥ j\}
IABCDEFGB \{(i,j)|i is even, j is even, i ≥ j\}
IABCDEFGBC \{(i,j)|i is even, j is even, i > j\}
IABCDEFGBCD \{(i,j)|i is odd, j is even, i > j + 1\}

...
Collecting semantics – example

Collecting semantics of a program = set of (concrete) states occurring at each program point.

Path          Concrete states
I             \{(i,j)\} (given)
IA            \{(i,j)|i \text{ is odd, } j \text{ is even}\}
IAB           \{(i,j)|i \text{ is odd, } j \text{ is even}\}
IABC          \{(i,j)|i \text{ is odd, } j \text{ is even, } i > j\}
IABCD         \{(i,j)|i \text{ is even, } j \text{ is even, } i > j + 1\}
IABCDE        \{(i,j)|i \text{ is even, } j \text{ is even, } i \geq j\}
IABCDEB       \{(i,j)|i \text{ is even, } j \text{ is even, } i \geq j\}
IABCDEBC      \{(i,j)|i \text{ is even, } j \text{ is even, } i > j\}
IABCDEBCD     \{(i,j)|i \text{ is odd, } j \text{ is even, } i > j + 1\}

Therefore, collecting semantics:
A             \{(i,j)|i \text{ odd, } j \text{ even}\}
B             \{(i,j)|i \text{ odd, } j \text{ even}\} \cup \{(i,j)|i \text{ even, } j \text{ even, } i \geq j\}
C             \{(i,j)|j \text{ even, } i > j\}
D             \{(i,j)|j \text{ even, } i > j + 1\}
E             \{(i,j)|j \text{ even, } i \geq j\}
F             \{(i,j)|i \text{ odd, } j \text{ even, } i < j\} \cup \{(i,j)|i \text{ even, } j \text{ even, } i = j\}
An abstract interpretation

Components of an abstract interpretation:

- **Set of abstract states** $D$, forming a complete lattice.
- **“Concretization”** function $\gamma : D \rightarrow 2^{State}$, which associates a set of concrete states with each abstract state.
- **Transfer function** $f_n : D \rightarrow D$ for each type of node $n$, which “interprets” each program statement using the abstract states.
Abstract interpretation – example

- Abstract lattice $D$

- Transfer functions for an assignment node $n$ of the form $p := \text{exp}$

$$f_n(s) = \begin{cases} 
\perp & \text{if } s \text{ is } \perp \\
(o, s[2]) & \text{if } \text{exp} \text{ evaluates to } o \text{ in state } s, \\
(e, s[2]) & \text{if } \text{exp} \text{ evaluates to } e \text{ in state } s \\
(oe, s[2]) & \text{if } \text{exp} \text{ evaluates to } oe \text{ in state } s 
\end{cases}$$

- The concretization function $\gamma$

$$\gamma((oe, oe)) =$$
Abstract interpretation – example

- Abstract lattice $D$

- Transfer functions for an assignment node $n$ of the form $p := \text{exp}$

\[
f_n(s) = \begin{cases} 
\bot & \text{if } s \text{ is } \bot \\
(o, s[2]) & \text{if } \text{exp} \text{ evaluates to } o \text{ in state } s, \\
(e, s[2]) & \text{if } \text{exp} \text{ evaluates to } e \text{ in state } s, \\
(oe, s[2]) & \text{if } \text{exp} \text{ evaluates to } oe \text{ in state } s 
\end{cases}
\]

- The concretization function $\gamma$
  - $\gamma((oe, oe)) = \text{State}$, $\gamma(\bot) =$
Abstract interpretation – example

- Abstract lattice $D$

- Transfer functions for an assignment node $n$ of the form $p := \exp$

$$f_n(s) = \begin{cases} 
\bot & \text{if } s \text{ is } \bot \\
(o, s[2]) & \text{if } \exp \text{ evaluates to } o \text{ in state } s, \\
(e, s[2]) & \text{if } \exp \text{ evaluates to } e \text{ in state } s \\
(oe, s[2]) & \text{if } \exp \text{ evaluates to } oe \text{ in state } s
\end{cases}$$

- The concretization function $\gamma$
  - $\gamma((oe, oe)) = State$, $\gamma(\bot) = \emptyset$, $\gamma((o, oe)) =$
Abstract interpretation – example

- Abstract lattice $D$

- Transfer functions for an assignment node $n$ of the form $p := \text{exp}$

\[
 f_n(s) = \begin{cases} 
 \bot & \text{if } s \text{ is } \bot \\
 (o, s[2]) & \text{if exp evaluates to } o \text{ in state } s, \\
 (e, s[2]) & \text{if exp evaluates to } e \text{ in state } s \\
 (oe, s[2]) & \text{if exp evaluates to } oe \text{ in state } s 
\end{cases}
\]

- The concretization function $\gamma$
  - $\gamma((oe, oe)) = \text{State}$, $\gamma(\bot) = \emptyset$, $\gamma((o, oe)) = \{(m, n) \mid m \text{ is odd}\}$
  - $\gamma((o, e)) = \emptyset$
Abstract interpretation – example

- Abstract lattice $D$

![Diagram of an abstract lattice]

- Transfer functions for an assignment node $n$ of the form $p := \exp$

$$f_n(s) = \begin{cases} 
\bot & \text{if } s \text{ is } \bot \\
(o, s[2]) & \text{if } \exp \text{ evaluates to } o \text{ in state } s, \\
& \text{where } s[2] \text{ is the 2nd component of the pair } s \\
(e, s[2]) & \text{if } \exp \text{ evaluates to } e \text{ in state } s \\
(oe, s[2]) & \text{if } \exp \text{ evaluates to } oe \text{ in state } s
\end{cases}$$

- The concretization function $\gamma$
  - $\gamma((oe, oe)) = \text{State}$, $\gamma(\bot) = \emptyset$, $\gamma((o, oe)) = \{(m, n) \mid m \text{ is odd}\}$
  - $\gamma((o, e)) = \{(m, n) \mid m \text{ is odd and } n \text{ is even}\}, \ldots$
Collecting abstract values – example

Path

Abstract value

I

(o, e)

(given)

p = odd()
q = even()

A

p > q

C

p := p+1

D

q := q+2

E

print p, q

G

F

Therefore, joining all abstract values at each point:

A (o, e) ⊔ (e, e) = (o, e)

B (o, e) ⊔ (e, e) = (o, e)

C (e, e) ⊔ (o, e) = (o, e)

D (e, e) ⊔ (o, e) = (o, e)

E (o, e) ⊔ (e, e) = (o, e)

F (o, e) ⊔ (e, e) = (o, e)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path Abstract value

I (oe, oe) (given)
IA (o, e)

Therefore, joining all abstract values at each point:

A (o, e) ⊔ (e, e) = (oe, e)
B (o, e) ⊔ (e, e) = (oe, e)
C (e, e) ⊔ (o, e) = (oe, e)
D (e, e) ⊔ (o, e) = (oe, e)
E (o, e) ⊔ (e, e) = (oe, e)
F (o, e) ⊔ (e, e) = (oe, e)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path                  Abstract value

A                  (oe, oe)  (given)
IAB                 (o, e)
IABCD               (o, e)
IABCDE              (o, e)
IABCDEBC             (o, e)
IABCDEBCD            (o, e)
IABF                (o, e)
IABCDEBF             (e, e)
This is abstract join-over-all-paths (JOP) solution.

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting abstract values – example

Path Abstract value
I \((oe, oe)\)
IA \((o, e)\)
IAB \((o, e)\)
IABC \((o, e)\)

Therefore, joining all abstract values at each point:
A \((o, e)\) \(\sqcup\) \((e, e)\) = \((oe, e)\)
B \((o, e)\) \(\sqcup\) \((e, e)\) = \((oe, e)\)
C \((e, e)\) \(\sqcup\) \((o, e)\) = \((oe, e)\)
D \((e, e)\) \(\sqcup\) \((o, e)\) = \((oe, e)\)
E \((o, e)\) \(\sqcup\) \((e, e)\) = \((oe, e)\)
F \((o, e)\) \(\sqcup\) \((e, e)\) = \((oe, e)\)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path                  Abstract value
I                     (oe, oe)
                      (given)
IA                    (o, e)
IAB                   (o, e)
IABC                  (o, e)
IABCD                 (e, e)

Therefore, joining all abstract values at each point:
A (o, e) ⊔ (e, e) = (oe, e)
B (o, e) ⊔ (e, e) = (oe, e)
C (o, e) ⊔ (e, e) = (oe, e)
D (e, e) ⊔ (o, e) = (oe, e)
E (o, e) ⊔ (e, e) = (oe, e)
F (o, e) ⊔ (e, e) = (oe, e)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path  Abstract value
I  (oe, oe)  (given)
IA  (o, e)
IAB  (o, e)
IABC  (o, e)
IABCD  (e, e)
IABCDE  (e, e)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path Abstract value
I (oe, oe) 
A (given) 
B 
P > q 
C 
I 
D 
E 
F 
G 
print p,q 
IABCDEBCD (o, e) 
IABCDEBCDE (e, e) 
IABCDEBF (o, e) 
IABCDEDE (e, e) 
IABCD (e, e) 
IABCDE (e, e) 
IABCD (o, e) 
IAÇÃO (o, e) 
IA (o, e) 
I (given) 
p := p+1 
q := q+2

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting abstract values – example

Path | Abstract value
---|---
I | (oe, oe) (given)
IA | (o, e)
IAB | (o, e)
IABC | (o, e)
IABCD | (e, e)
IABCDE | (e, e)
IABCDEB | (e, e)
IABCDEBC | (e, e)
IABCDEBCD | (e, e)
IABF | (o, e)
IABCDEBF | (e, e)

Therefore, joining all abstract values at each point:

A (o, e) ⊔ (e, e) = (oe, e)
B (o, e) ⊔ (e, e) = (oe, e)
C (o, e) ⊔ (e, e) = (oe, e)
D (e, e) ⊔ (o, e) = (oe, e)
E (o, e) ⊔ (e, e) = (oe, e)
F (o, e) ⊔ (e, e) = (oe, e)

This is abstract join-over-all-paths (JOP) solution.
Collecting abstract values – example

Path | Abstract value
--- | ---
I | \( (oe, oe) \) (given)
IA | \( (o, e) \)
IAB | \( (o, e) \)
IABC | \( (o, e) \)
IABCD | \( (e, e) \)
IABCDE | \( (e, e) \)
IABCDEB | \( (e, e) \)
IABCDEBC | \( (e, e) \)
IABCDEBCD | \( (o, e) \)
Collecting abstract values – example

Path | Abstract value
-----|-----------------
I | (oe, oe) (given)
IA | (o, e)
IAB | (o, e)
IABC | (o, e)
IABCD | (e, e)
IABCDE | (e, e)
IABCDEB | (e, e)
IABCDEBC | (e, e)
IABCDEBCD | (o, e)
IABCDEBCDE | (o, e)
Collecting abstract values – example

Path | Abstract value
--- | ---
I | $(oe, oe)$ (given)
IA | $(o, e)$
IAB | $(o, e)$
IABC | $(o, e)$
IABCD | $(e, e)$
IABCDDE | $(e, e)$
IABCDDEB | $(e, e)$
IABCDDEBC | $(e, e)$
IABCDDEBCD | $(o, e)$
IABCDDEBCE | $(o, e)$
IABE | $(o, e)$

Deepak D’Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
Collecting abstract values – example

Path Abstract value
I (oe, oe) (given)
IA (o, e)
IAB (o, e)
IABC (o, e)
IABCD (e, e)
IABCDE (e, e)
IABCDEB (e, e)
IABCDEBC (e, e)
IABCDEBCD (o, e)
IABCDEBCDE (o, e)
IABF (o, e)
IABCDEBF (e, e)
Collecting abstract values – example

Path | Abstract value
--- | ---
I | \((oe, oe)\)
IA | \((o, e)\)
IAB | \((o, e)\)
IABC | \((o, e)\)
IABCD | \((e, e)\)
IABCDE | \((e, e)\)
IABCDEB | \((e, e)\)
IABCDEBC | \((e, e)\)
IABCDEBCD | \((o, e)\)
IABCDEFG | \((o, e)\)
IABF | \((o, e)\)
IABCDEFGF | \((e, e)\)

Therefore, joining all abstract values at each point:

A | \((o, e)\)
B | \((o, e) \sqcup (e, e) = (oe, e)\)
C | \((o, e) \sqcup (e, e) = (oe, e)\)
D | \((e, e) \sqcup (o, e) = (oe, e)\)
E | \((e, e) \sqcup (o, e) = (oe, e)\)
F | \((o, e) \sqcup (e, e) = (oe, e)\)
Collecting abstract values – example

Path Abstract value
I \( (oe, oe) \) (given)
IA \( (o, e) \)
IAB \( (o, e) \)
IABC \( (o, e) \)
IABCD \( (e, e) \)
IABCDE \( (e, e) \)
IABCD DE \( (e, e) \)
IABCDE \( (e, e) \)
IABF \( (o, e) \)
IABCDEF \( (e, e) \)

Therefore, joining all abstract values at each point:
A \( (o, e) \)
B \( (o, e) \sqcup (e, e) = (oe, e) \)
C \( (o, e) \sqcup (e, e) = (oe, e) \)
D \( (e, e) \sqcup (o, e) = (oe, e) \)
E \( (e, e) \sqcup (o, e) = (oe, e) \)
F \( (o, e) \sqcup (e, e) = (oe, e) \)

This is abstract join-over-all-paths (JOP) solution.
Comparison of abstract JOP states and collecting states

<table>
<thead>
<tr>
<th>Abstract JOP:</th>
<th>Collecting states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ((o, e))</td>
<td>A {((i, j))</td>
</tr>
<tr>
<td>B ((oe, e))</td>
<td>B {((i, j))</td>
</tr>
<tr>
<td>C ((oe, e))</td>
<td>C {((i, j))</td>
</tr>
<tr>
<td>D ((oe, e))</td>
<td>D {((i, j))</td>
</tr>
<tr>
<td>E ((oe, e))</td>
<td>E {((i, j))</td>
</tr>
<tr>
<td>F ((oe, e))</td>
<td>F {((i, j))</td>
</tr>
</tbody>
</table>
Comparison of abstract JOP states and collecting states

<table>
<thead>
<tr>
<th>Abstract JOP:</th>
<th>Collecting states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ((o, e))</td>
<td>A (((i, j)</td>
</tr>
<tr>
<td>B ((oe, e))</td>
<td>B (((i, j)</td>
</tr>
<tr>
<td>C ((oe, e))</td>
<td>C (((i, j)</td>
</tr>
<tr>
<td>D ((oe, e))</td>
<td>D (((i, j)</td>
</tr>
<tr>
<td>E ((oe, e))</td>
<td>E (((i, j)</td>
</tr>
<tr>
<td>F ((oe, e))</td>
<td>F (((i, j)</td>
</tr>
</tbody>
</table>

Note that at each point \(\gamma\) image of abstract solution is over-approximation of collecting states.
A given abstract interpretation is said to be *correct* if, for all abstract states $d_0 \in D$, for all programs $P$ and for all program points $p$ in $P$,

$\gamma$ image of join of all abstract states arising at $p$ (i.e., abstract JOP solution at $p$), with $d_0$ as the initial abstract value at $P$’s entry

$\supseteq$

collecting semantics at $p$, with $\gamma(d_0)$ as the initial set of concrete states at $P$’s entry
A given abstract interpretation is said to be *correct* if, for all abstract states $d_0 \in D$, for all programs $P$ and for all program points $p$ in $P$,

$\gamma$ image of join of all abstract states arising at $p$ (i.e., abstract JOP solution at $p$), with $d_0$ as the initial abstract value at $P$’s entry

$\supseteq$

collecting semantics at $p$, with $\gamma(d_0)$ as the initial set of concrete states at $P$’s entry

We will study later certain sufficient conditions for a given abstract interpretation to be correct.
Another example program

Path Characterization of concrete states

I $true$ (given)
IB $x = 1$
IBC $x = 1$
IBCD $x = 1 \land y = 1$
IBCDE $x = -1 \land y = 1$
IBCDEC $x = -1 \land y = 1$
IBCDECD $x = -1 \land y = 1$
... $x = -1 \land y = 1$

Therefore, collecting semantics:

Point Characterization of concrete states

I $true$
B $x = 1$
C $(x = 1) \lor (x = -1 \land y = 1)$
D $(y = 1) \land (x = -1 \lor x = 1)$
E $x = -1 \land y = 1$
Abstract interpretation for constant propagation

- **Abstract lattice** $D$

- **Concretization function**: What is $\gamma(d)$?
Abstract interpretation for constant propagation

- Abstract lattice $D$

- Concretization function: What is $\gamma(d)$?

\[
\begin{align*}
&\bot \quad \mapsto \quad \{\} \\
&\emptyset \quad \mapsto \quad \text{State} \\
&\{(v, c)\} \quad \mapsto \quad \{s | s \in \text{State} \land s[v] = c\} \\
&\{(x, c), (y, d)\} \quad \mapsto \quad \{(c, d)\}
\end{align*}
\]
Transfer function for assignment node $n$ of the form $x := \text{exp}$. 

$$f_n(P) = \bot, \quad \text{if } P \text{ is } \bot$$

$$= \left\{ (y, c) \in P \mid y \neq x \right\} \cup \{(x, d)\}, \quad \text{else if } [\text{exp}]_P = d$$

$$= \left\{ (y, c) \in P \mid y \neq x \right\}, \quad \text{otherwise}$$
Path | Abstract value at end of path
--- | ---
I | ∅
JOP using abstract lattice

Path | Abstract value at end of path
---|---
l | ∅
IB | {(x, 1)}
<table>
<thead>
<tr>
<th>Path</th>
<th>Abstract value at end of path</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>I</code></td>
<td>∅</td>
</tr>
<tr>
<td><code>IB</code></td>
<td><code>{(x, 1)}</code></td>
</tr>
<tr>
<td><code>IBC</code></td>
<td><code>{(x, 1)}</code></td>
</tr>
</tbody>
</table>

```
x := 1
y := x*x
x := -1
```
JOP using abstract lattice

Path | Abstract value at end of path
--- | ---
I | ∅
IB | {(x, 1)}
IBC | {(x, 1)}
IBCD | {(x, 1), (y, 1)}
Path                        Abstract value at end of path
I                           ∅
IB                          {(x, 1)}
IBC                         {(x, 1)}
IBCD                        {(x, 1), (y, 1)}
IBCDE                       {(x, −1), (y, 1)}
Path | Abstract value at end of path
---|---
I  | ∅
IB | \{(x, 1)\}
IBC | \{(x, 1)\}
IBCD | \{(x, 1), (y, 1)\}
IBCDE | \{(x, -1), (y, 1)\}
IBCDEC | \{(x, -1), (y, 1)\}

Deepak D'Souza and K. V. Raghavan (IISc) Data-flow Analysis / Abstract Interpretation
JOP using abstract lattice

Path | Abstract value at end of path
---|---
I | ∅
IB | \{(x, 1)\}
IBC | \{(x, 1)\}
IBCD | \{(x, 1), (y, 1)\}
IBCDE | \{(x, -1), (y, 1)\}
IBCDEC | \{(x, -1), (y, 1)\}
IBCDECD | \{(x, -1), (y, 1)\}
JOP using abstract lattice

Path | Abstract value at end of path
--- | ---
I | ∅
IB | {(x, 1)}
IBC | {(x, 1)}
IBCD | {(x, 1), (y, 1)}
IBCDE | {(x, −1), (y, 1)}
IBCDEC | {(x, −1), (y, 1)}
IBCDECD | {(x, −1), (y, 1)}
... | {(x, −1), (y, 1)}
JOP using abstract lattice

Path                        Abstract value at end of path
I                           \emptyset
IB                          \{(x, 1)\}
IBC                         \{(x, 1)\}
IBCD                       \{(x, 1), (y, 1)\}
IBCDE                      \{(x, -1), (y, 1)\}
IBCDEC                     \{(x, -1), (y, 1)\}
IBCDECD                    \{(x, -1), (y, 1)\}
...                        \{(x, -1), (y, 1)\}

Point                        Abstract JOP value
I                           \emptyset
JOP using abstract lattice

Path | Abstract value at end of path
--- | ---
I | ∅
IB | {(x, 1)}
IBC | {(x, 1)}
IBCD | {(x, 1), (y, 1)}
IBCDE | {(x, −1), (y, 1)}
IBCDEC | {(x, −1), (y, 1)}
IBCDECD | {(x, −1), (y, 1)}
... | {(x, −1), (y, 1)}

Point | Abstract JOP value
--- | ---
I | ∅
B | {(x, 1)}
JOP using abstract lattice

Path  Abstract value at end of path
I  $\emptyset$
IB  $\{(x, 1)\}$
IBC  $\{(x, 1)\}$
IBCD  $\{(x, 1), (y, 1)\}$
IBCDE  $\{(x, -1), (y, 1)\}$
IBCDEC  $\{(x, -1), (y, 1)\}$
...  $\{(x, -1), (y, 1)\}$

Point  Abstract JOP value
I  $\emptyset$
B  $\{(x, 1)\}$
C  $\emptyset$
JOP using abstract lattice

Path                  Abstract value at end of path
I                    ∅
IB                   {(x, 1)}
IBC                  {(x, 1)}
IBCD                 {(x, 1), (y, 1)}
IBCDE                {(x, −1), (y, 1)}
IBCDECD              {(x, −1), (y, 1)}
...                  {(x, −1), (y, 1)}

Point                  Abstract JOP value
I                    ∅
B                   {(x, 1)}
C                    ∅
D                   {(y, 1)}
Path | Abstract value at end of path
---|---
I | ∅
IB | {(x, 1)}
IBC | {(x, 1)}
IBCD | {(x, 1), (y, 1)}
IBCDE | {(x, −1), (y, 1)}
IBCDEC | {(x, −1), (y, 1)}
IBCDECD | {(x, −1), (y, 1)}
... | {(x, −1), (y, 1)}

Point | Abstract JOP value
---|---
I | ∅
B | {(x, 1)}
C | ∅
D | {(y, 1)}
E | {(x, −1), (y, 1)}
Correctness in previous example

Verify that

- at points I, B and E
  \( \gamma(\text{abstract JOP value}) = \text{collecting semantics.} \)
- at points C and D
  \( \gamma(\text{abstract JOP value}) \supset \text{collecting semantics.} \)
- the abstract transfer functions given are the best possible for the given lattice \( L \). That is, imprecision is due to the lattice, not the transfer functions.
Formal definition of control-flow graphs

Programs are finite directed graphs with following nodes (statements):

**Nodes or statements in a program**

- **Expressions:**
  
  $e ::= c | x | e + e | e - e | e \times e.$

- **Boolean expressions:**
  
  $be ::= tt | ff | e \leq e | e = e | \neg be | be \lor be | be \land be.$

- Assume unique initial program point $I$. 
Formal definition of an abstract interpretation

- Complete join semi-lattice \((D, \leq)\), with a least element \(\perp\).
- Concretization function \(\gamma : D \to 2^{\text{State}}\)
- \(\perp \in D\) represents unreachability of the program point (i.e., \(\gamma(\perp) = \emptyset\))
- Transfer function \(f_{LM} : D \to D\) for each node \(n\) and incoming edge \(L\) into \(n\) and outgoing edge \(M\) from \(n\).

Junction nodes have identity transfer function.
What we want to compute for a given program

- Path in a program: Sequence of connected edges or program points.
- Transfer functions extend to paths in program:
  \[ f_{ABCD} = f_{CD} \circ f_{BC} \circ f_{AB}. \]
  where \((f_a \circ f_b)(x)\) is defined as \(f_a(f_b(x))\).
- \(f_p\) is \(\lambda d. \bot \Rightarrow \) path \(p\) is infeasible.
- Join over all paths (JOP) definition: For each program point \(N\)
  \[ d_N = \bigcup \{ f_p(d_0) \mid \text{paths } p \text{ from } I \text{ to } N \}. \]
  where \(d_0\) is a given initial abstract value at entry node.
Formalization of collecting semantics

- Let $Val$ be the set of all concrete values; e.g., $Integer \cup Boolean$.
- $State$ is normally the domain $Var \rightarrow Val$. However, in general, it can be any semantic domain.

Program semantics is given by the functions $\text{nstate}_{MN} : State \rightarrow 2^{State}$

These induce the functions $\text{nstate}' : 2^{State} \rightarrow 2^{State}$

$$\text{nstate}'_{MN}(S_1 \in 2^{State}) = \bigcup_{s_1 \in S_1} \text{nstate}_{MN}(s_1)$$
Collecting semantics SS is a map \( \text{ProgramPoints} \rightarrow 2^{\text{State}} \).

At each program point \( N \),

\[
SS(N) = \bigcup_{p \text{ path from } I \text{ to } N} nstate'_p(S_0).
\]

where \( I \) is entry point of CFG, \( S_0 \) is the given initial set of states, and \( nstate'_p \) is composition of \( nstate' \) functions of edges that constitute \( p \).