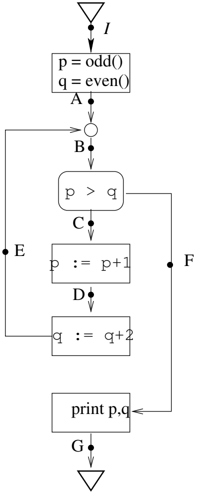
What is data-flow analysis

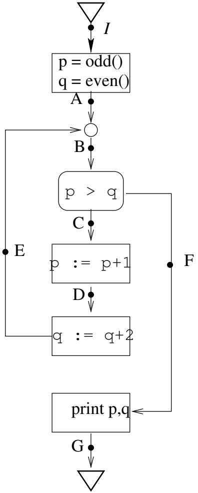
- "Computing 'safe' approximations to the set of values / behaviours arising dynamically at run time, statically or at compile time."
- Typically used by compiler writers to optimize running time of compiled code.
  - Constant propagation: Is the value of a variable constant at a particular program location.
  - Replace `x := y + z` by `x := 17` during compilation.
- More recently, used for verifying properties of programs.

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



Path	Concrete states
I	$\{(i,j)\}$ (given)

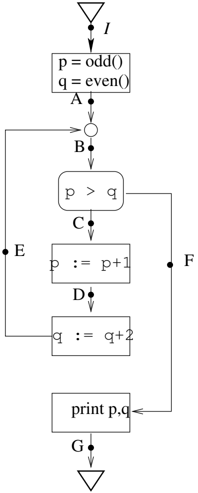
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

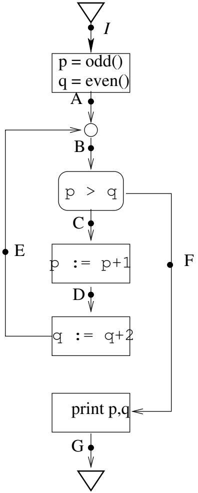
Collecting semantics – example

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



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

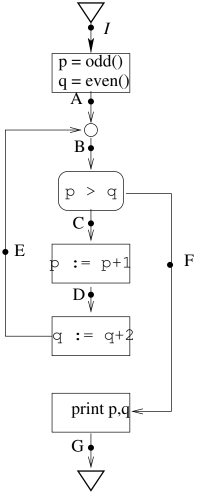
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

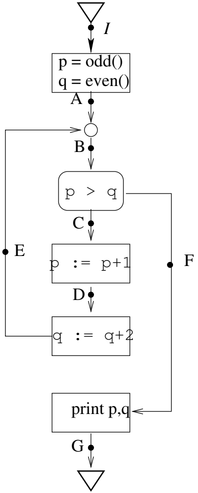
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

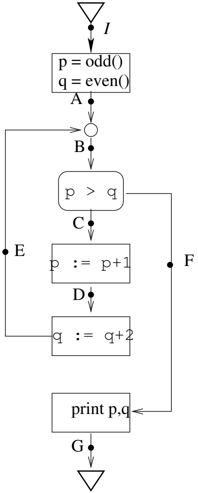
Collecting semantics of a program = set of (concrete) states occurring at each program point.



Path	Concrete states
I	$\{(i,j)\}$ (given)
IA	$\{(i,j) \| i$ 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, $i \geq j\}$

Collecting semantics – example

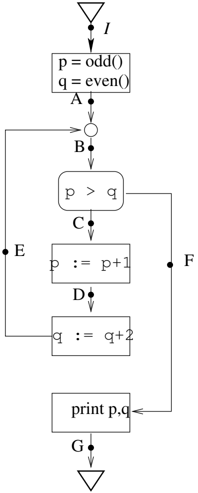
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

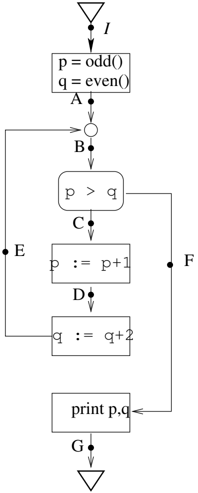
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

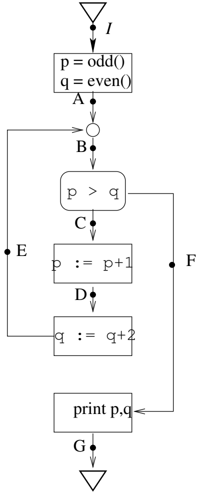
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

...

Therefore, collecting semantics:

I	$\{(i,j)\}$
A	$\{(i,j) \mid i \text{ odd}, j \text{ even}\}$
B	$\{(i,j) \mid i \text{ odd}, j \text{ even}\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i \geq j\}$
C	$\{(i,j) \mid j \text{ even}, i > j\}$
D	$\{(i,j) \mid j \text{ even}, i > j+1\}$
E	$\{(i,j) \mid j \text{ even}, i \geq j\}$
F	$\{(i,j) \mid i \text{ odd}, j \text{ even}, i < j\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i = j\}$

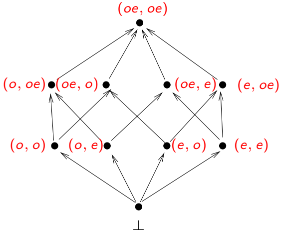
An abstract interpretation

Components of an abstract interpretation:

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

Abstract interpretation – example

- Abstract lattice D



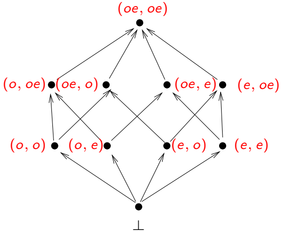
- Transfer function for an assignment node n: `p := p+q`

$$f_n(s) = \begin{cases} \bot & \text{if } s \text{ is } \bot \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

- The concretization function γ
  - $\gamma((oe, oe)) =$

Abstract interpretation – example

- Abstract lattice D



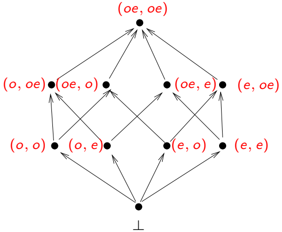
- Transfer function for an assignment node n: `p := p+q`

$$f_n(s) = \begin{cases} \perp & \text{if } s \text{ is } \perp \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

- The concretization function γ
  - $\gamma((oe, oe)) = \mathit{State}$, $\gamma(\perp) =$

Abstract interpretation – example

- Abstract lattice D



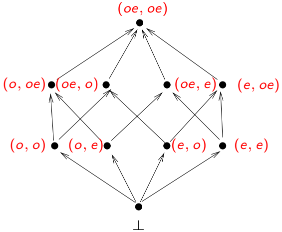
- Transfer function for an assignment node n: `p := p+q`

$$f_n(s) = \begin{cases} \bot & \text{if } s \text{ is } \bot \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

- The concretization function γ
  - $\gamma((oe, oe)) = \mathit{State}$, $\gamma(\bot) = \emptyset$, $\gamma((o, oe)) =$

Abstract interpretation – example

- Abstract lattice D



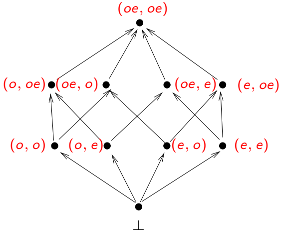
- Transfer function for an assignment node n: `p := p+q`

$$f_n(s) = \begin{cases} \perp & \text{if } s \text{ is } \perp \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

- The concretization function γ
  - $\gamma((oe, oe)) = \mathit{State}$, $\gamma(\perp) = \emptyset$, $\gamma((o, oe)) = \{(m, n) \mid m \text{ is odd}\}$
  - $\gamma((o, e)) =$

Abstract interpretation – example

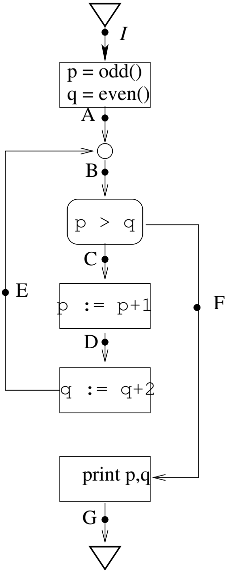
- Abstract lattice D



- Transfer function for an assignment node n: `p := p+q`

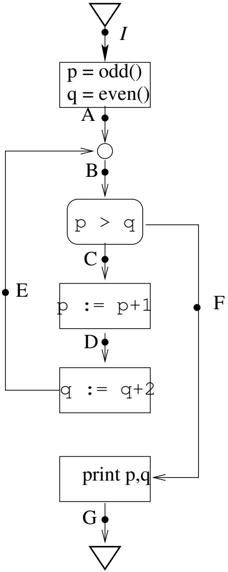
$$f_n(s) = \begin{cases} \bot & \text{if } s \text{ is } \bot \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

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

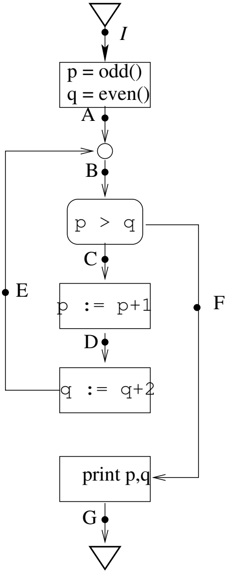


Path	Abstract value
I	(oe, oe) (given)

Collecting abstract values – example

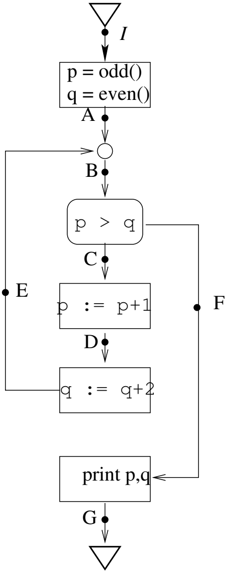


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)



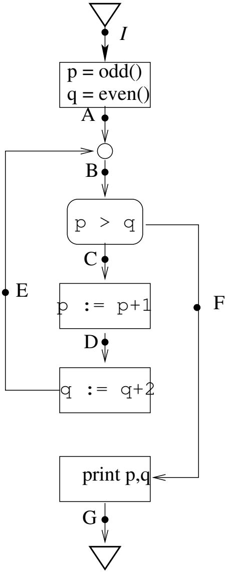
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)

Collecting abstract values – example

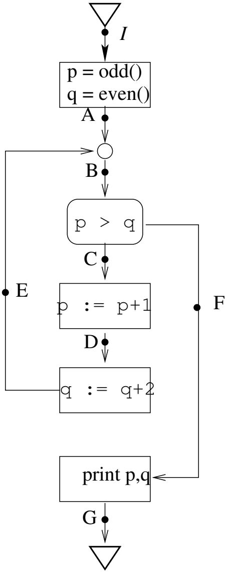


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)

Collecting abstract values – example

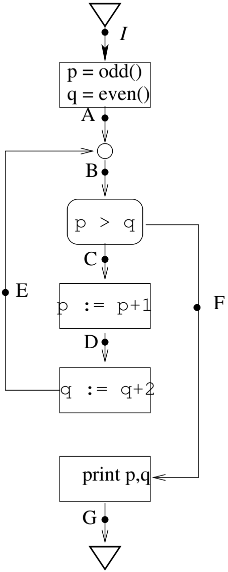


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)



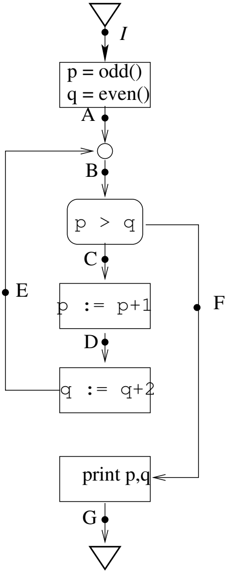
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)

Collecting abstract values – example



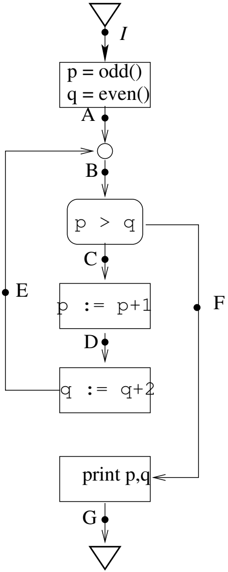
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)

Collecting abstract values – example



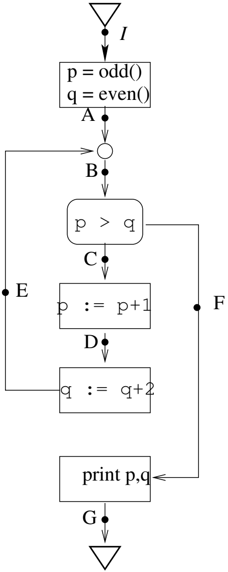
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)

Collecting abstract values – example



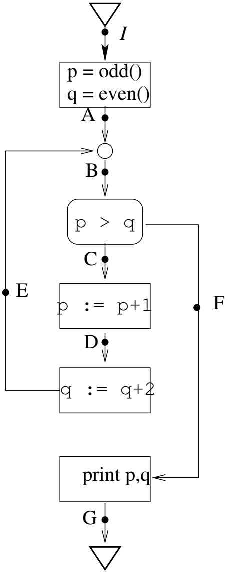
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)

Collecting abstract values – example



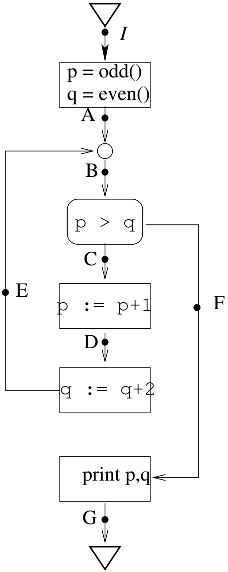
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)

Collecting abstract values – example



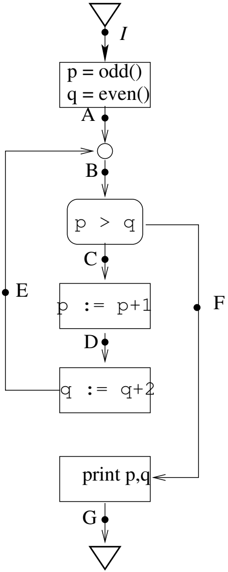
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)
IABF	(o, e)

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)
IABF	(o, e)
IABCDEBF	(e, e)

Collecting abstract values – example

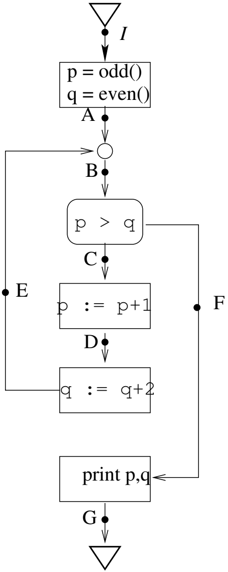


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)
IABF	(o, e)
IABCDEBF	(e, e)

Therefore, joining all abstract values at each point:

I	(oe, oe)		
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)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)
IABF	(o, e)
IABCDEBF	(e, e)

Therefore, joining all abstract values at each point:

I	(oe, oe)		
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

Abstract JOP:

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

Collecting states:

A	$\{(i,j) \mid i \text{ odd}, j \text{ even}\}$
B	$\{(i,j) \mid i \text{ odd}, j \text{ even}\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i \geq j\}$
C	$\{(i,j) \mid j \text{ even}, i > j\}$
D	$\{(i,j) \mid j \text{ even}, i > j+1\}$
E	$\{(i,j) \mid j \text{ even}, i \geq j\}$
F	$\{(i,j) \mid i \text{ odd}, j \text{ even}, i < j\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i = j\}$

Comparison of abstract JOP states and collecting states

Abstract JOP:		Collecting states:	
A	(o, e)	A	$\{(i,j) \mid i \text{ odd}, j \text{ even}\}$
B	(oe, e)	B	$\{(i,j) \mid i \text{ odd}, j \text{ even}\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i \geq j\}$
C	(oe, e)	C	$\{(i,j) \mid j \text{ even}, i > j\}$
D	(oe, e)	D	$\{(i,j) \mid j \text{ even}, i > j + 1\}$
E	(oe, e)	E	$\{(i,j) \mid j \text{ even}, i \geq j\}$
F	(oe, e)	F	$\{(i,j) \mid i \text{ odd}, j \text{ even}, i < j\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i = j\}$

Note that at each point γ 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,

γ 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,

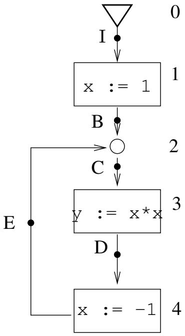
γ image of join of all abstract states arising at p (i.e., abstract JOP solution at p), with d_0 as the initial abstract value at P's entry

$\supseteq$

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



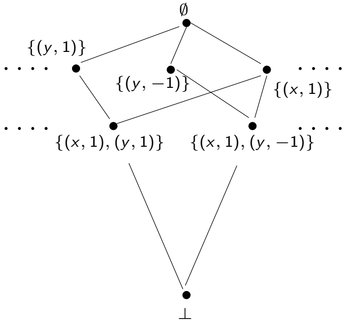
Path	Characterization of concrete states
I	*true* (given)
IB	x = 1
IBC	x = 1
IBCD	x = 1 $\wedge$ y = 1
IBCDE	x = -1 $\wedge$ y = 1
IBCDEC	x = -1 $\wedge$ y = 1
IBCDECD	x = -1 $\wedge$ y = 1
...	x = -1 $\wedge$ y = 1

Therefore, collecting semantics:

Point	Characterization of concrete states
I	*true*
B	x = 1
C	(x = 1) $\vee$ (x = -1 $\wedge$ y = 1)
D	(y = 1) $\wedge$ (x = -1 $\vee$ x = 1)
E	x = -1 $\wedge$ y = 1

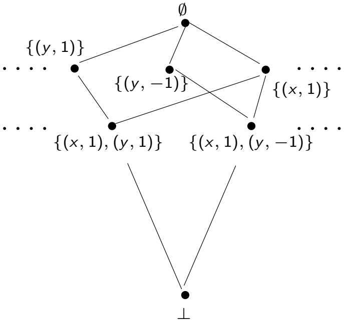
Abstract interpretation for constant propagation

- Abstract lattice D



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

- Abstract lattice D



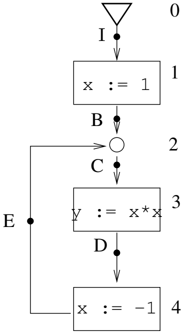
- Concretization function: What is $\gamma(d)$?

$$
\begin{array}{lcl}
\perp & \mapsto & \{\} \\
\emptyset & \mapsto & \mathit{State} \\
\{(x,c)\} & \mapsto & \{(c,j) \mid j \text{ is any value}\} \\
\{(x,c),(y,d)\} & \mapsto & \{(c,d)\}
\end{array}
$$

Transfer function for assignment node n of the form $x := exp$.

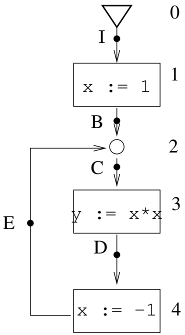
$$
\begin{aligned}
f_n(P) &= \bot, \text{ if } P \text{ is } \bot \\
&= \{(y,c) \in P \mid y \neq x\} \cup \{(x,d)\}, \\
&\quad \text{if all variables in } exp \text{ have constant values in } P, \text{ and if} \\
&\quad \text{exp evaluates to } d \text{ with these constant values} \\
&= \{(y,c) \in P \mid y \neq x\}, \text{ otherwise}
\end{aligned}
$$

JOP using abstract lattice



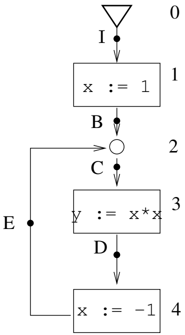
Path	Abstract value at end of path
I	$\emptyset$

JOP using abstract lattice



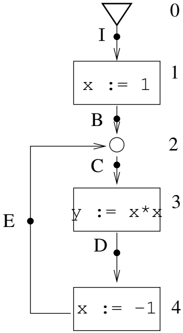
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$

JOP using abstract lattice



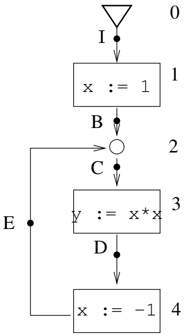
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$

JOP using abstract lattice



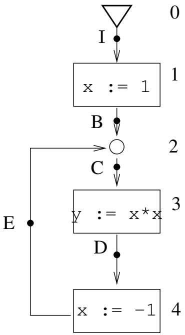
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$

JOP using abstract lattice



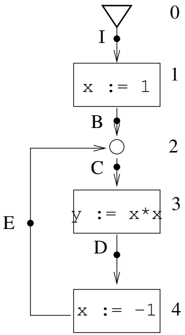
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$

JOP using abstract lattice



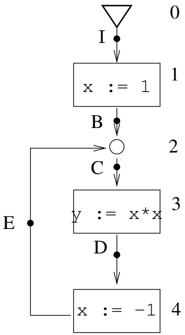
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$

JOP using abstract lattice



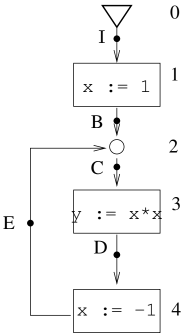
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$

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	$\{(x, -1), (y, 1)\}$
...	$\{(x, -1), (y, 1)\}$

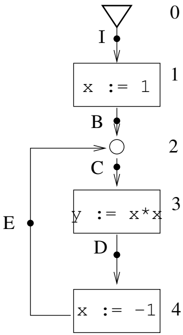
JOP using abstract lattice



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	$\emptyset$

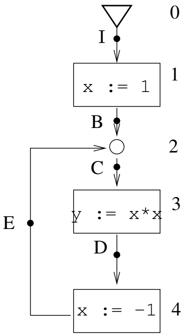
JOP using abstract lattice



Path	Abstract value at end of path
I	$\emptyset$
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	$\emptyset$
B	$\{(x,1)\}$

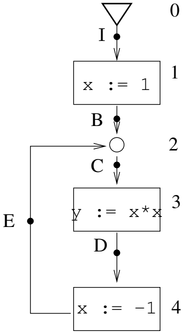
JOP using abstract lattice



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	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$
...	$\{(x, -1), (y, 1)\}$

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

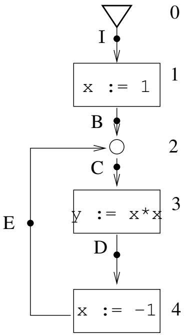
JOP using abstract lattice



Path	Abstract value at end of path
I	$\emptyset$
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	$\emptyset$
B	$\{(x,1)\}$
C	$\emptyset$
D	$\{(y,1)\}$

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	$\emptyset$
B	$\{(x, 1)\}$
C	$\emptyset$
D	$\{(y, 1)\}$
E	$\{(x, -1), (y, 1)\}$

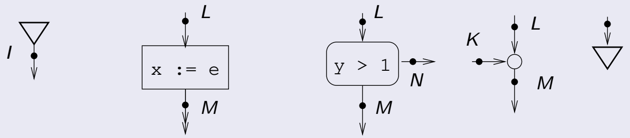
Correctness in previous example

Verify that

- at points I, B and E
  γ(abstract JOP value) $=$ collecting semantics.
- at points C and D
  γ(abstract JOP value) $\supset$ 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.

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

Nodes or statements in a program



- Expressions:

$$e ::= c \mid x \mid e + e \mid e - e \mid e * e.$$

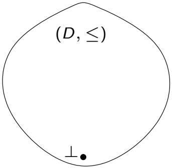
- Boolean expressions:

$$be ::= tt \mid ff \mid e \leq e \mid e = e \mid \neg be \mid be \vee be \mid be \wedge be.$$

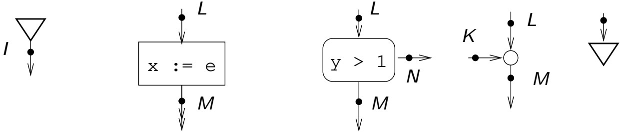
- Assume unique initial program point I.

Formal definition of an abstract interpretation

- Complete join semi-lattice $(D, \leq)$, with a least element $\bot$.
- Concretization function $\gamma : D \rightarrow 2^{State}$
- $\bot \in D$ represents unreachability of the program point (i.e., $\gamma(\bot)$ should be equal to $\emptyset$). Also, $\gamma(\top)$ should be *State*.



- We require transfer functions f_{LM}, f_{LN}, f_{KM} for all scenarios below:



- We assume transfer functions are monotonic, and satisfy $f(\bot) = \bot$.
- For junction nodes, both transfer functions should be *identity*

What we want to compute for a given program

- Path in a program: Sequence of connected edges or program points, beginning at initial point I
- Transfer functions extend to paths in program:

$$f_{IBCD} = f_{CD} \circ f_{BC} \circ f_{IB}.$$

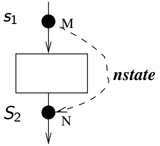
  where $(f_a \circ f_b)(x)$ is defined as $f_a(f_b(x))$.
- f_p is $\lambda d.\bot \;\Rightarrow\;$ path p is *infeasible*.
- Join over all paths (JOP) definition: For each program point N

$$d_N = \bigsqcup_{\text{paths } p \text{ from } I \text{ to } N} f_p(d_0).$$

  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 $nstate_{MN} : State \rightarrow 2^{State}$



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

$$nstate'_{MN}(S_1 \in 2^{State}) = \bigcup_{s_1 \in S_1} nstate_{MN}(s_1)$$

Formalization of collecting semantics – contd.

- Collecting semantics SS is a map $ProgramPoints \rightarrow 2^{State}$
- At each program point N,

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

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.

Recollection of Abstract Interpretation

It is a tuple (D, F_D, γ), such that

- $(D, \leq)$ is a complete join semi-lattice (aka the abstract lattice), with a least element $\perp$.
- Concretization function $\gamma : D \rightarrow 2^{State}$
- Monotone transfer function $(f_{LM} : D \rightarrow D) \in F_D$ for each node n and incoming edge L into n and outgoing edge M from n.
    - Junction nodes have identity transfer function.

An aside: Collecting semantics stated as an abstract interpretation

- Concrete lattice $C : (2^{State}, \subseteq)$, $\bot = \emptyset$, $\top = State$, $\sqcup = \cup$.
- Transfer function $f_{LM} = nstate'_{LM}$ for each node n and incoming edge L into n and outgoing edge M from n.
- $\gamma : C \to C$ is identity

An aside: Collecting semantics stated as an abstract interpretation

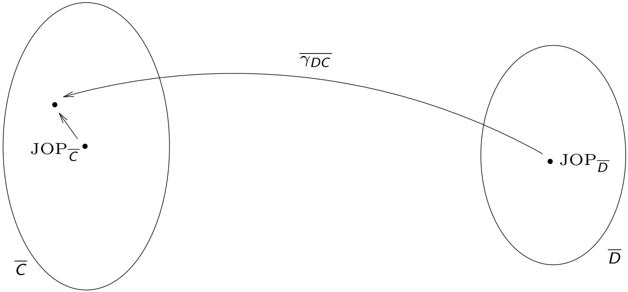
- Concrete lattice $C : (2^{State}, \subseteq)$, $\bot = \emptyset$, $\top = State$, $\sqcup = \cup$.
- Transfer function $f_{LM} = nstate'_{LM}$ for each node n and incoming edge L into n and outgoing edge M from n.
- $\gamma : C \rightarrow C$ is identity
- Therefore, collecting states at any point $N =$ JOP at this point using this interpretation
- This particular abstract interpretation is also known as the concrete interpretation.

Definition: consistent abstractions

An A.I. $(D, F_D, \gamma_D : D \to 2^{State})$ is said to be a consistent abstraction of (or, be correct wrt) another A.I. $(C, F_C, \gamma_C : C \to 2^{State})$ *under* a pair of monotone functions $\gamma_{DC} : D \to C$ and $\alpha_{CD} : C \to D$ iff:
(a) $(\alpha_{CD}, \gamma_{DC})$ form a Galois connection, and

(b) for all programs, and for all $d_0 \in D$ and $c_0 \in C$ such that $\gamma_{DC}(d_0) \geq c_0$:

$$\mathrm{JOP}_{\overline{C}} \;\leq\; \overline{\gamma_{DC}}(\mathrm{JOP}_{\overline{D}})$$



Definition – contd.

where

- $\mathrm{JOP}_{\overline{C}}$ is obtained by using (C, f_C), with c_0 as the initial state,
- $\mathrm{JOP}_{\overline{D}}$ is by obtained using (D, f_D), with d_0 as the initial state, and
- $\overline{x}$ is the "vectorized" form of x, i.e., x for all points in a program.

Note: Throughout remaining slides we use γ to mean γ_{DC} and α to mean α_{CD}.

Definition: (α, γ) form Galois Connection

- α and γ are monotonic
- $\gamma(\alpha(e)) \geq e$, for all $e \in C$
- $\alpha(\gamma(d)) = d$, for all $d \in D$

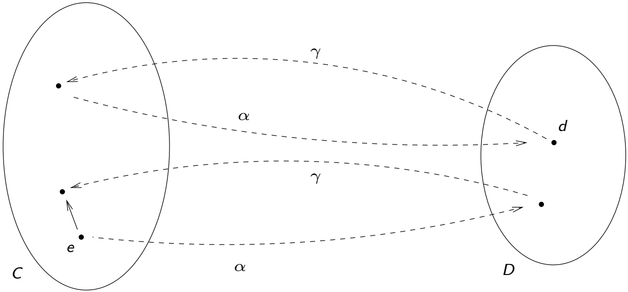


Illustration of consistent abstraction

- Consider the lattices L_1 and L_2 from the introduction slides.
- L_1 is a consistent abstraction of L_2 under the following (α, γ):

$$
\begin{array}{lcl}
\alpha(S \in L_2) & = & \bot, \text{ if } S = \emptyset \\
 & = & (coll(\{x \mid (x,y) \in S\}), coll(\{y \mid (x,y) \in S\})), \\
 & & \text{ otherwise} \\
\gamma((c,d) \in L_1) & = & \{(x,y) \mid \text{if } c \text{ is } oe \text{ then } x = o \vee x = e \text{ else } x = c, \\
 & & \quad \text{if } d \text{ is } oe \text{ then } y = o \vee y = e \text{ else } y = d\}
\end{array}
$$

where

$$
\begin{array}{lcl}
coll(W) & = & o, \text{ if } W = \{o\} \\
 & = & e, \text{ if } W = \{e\} \\
 & = & oe, \text{ if } W = \{o,e\}
\end{array}
$$

Another illustration of consistent abstraction

Constant propagation (CP) is a consistent abstraction of the concrete interpretation, under the following (α, γ):

$$
\begin{aligned}
\alpha(S \in 2^{State}) &= \bot, \\
&\quad \text{if } S \text{ is empty} \\
&= \{(x, c) \mid \forall e \in S : \; e(x) = c\}, \\
&\quad \text{otherwise} \\
\gamma(p) &= \emptyset, \\
&\quad \text{if } p = \bot \\
&= \{e \in State \mid \text{for each } (x, c) \in p : e(x) = c\}, \\
&\quad \text{if } p \text{ is any other element of the lattice}
\end{aligned}
$$

Properties of consistent abstractions

- Note: If an abstract interpretation $(D, F_D, \gamma : D \rightarrow 2^{State})$ is a consistent abstraction of $(2^{State}, nstate', identity)$, then we say that $(D, F_D, \gamma : D \rightarrow 2^{State})$ is correct.
- Consistent-abstraction-of is a transitive property. That is, if $(D, F_D, \gamma_D : D \rightarrow 2^{State})$ is a consistent abstraction of $(C, F_C, \gamma_C : C \rightarrow 2^{State})$ under $\gamma_{DC} : D \rightarrow C$, and $(C, F_C, \gamma_C : C \rightarrow 2^{State})$ is a consistent abstraction of $(B, F_B, \gamma_B : B \rightarrow 2^{State})$ under $\gamma_{CB} : C \rightarrow B$, then $(D, F_D, \gamma_D : D \rightarrow 2^{State})$ is a consistent abstraction of $(B, F_B, \gamma_B : B \rightarrow 2^{State})$ under $\gamma_{CB} \circ \gamma_{DC}$.

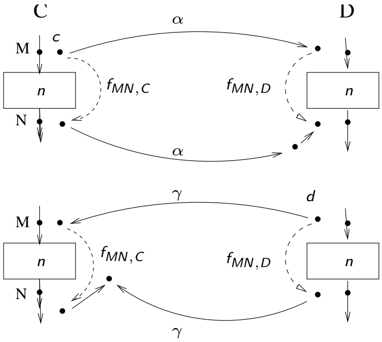
A sufficient condition for correctness

Theorem 1: An abstract interpretation (D, F_D, γ_D) is a consistent abstraction of another abstract interpretation (C, F_C, γ_C) under a pair (α, γ) if

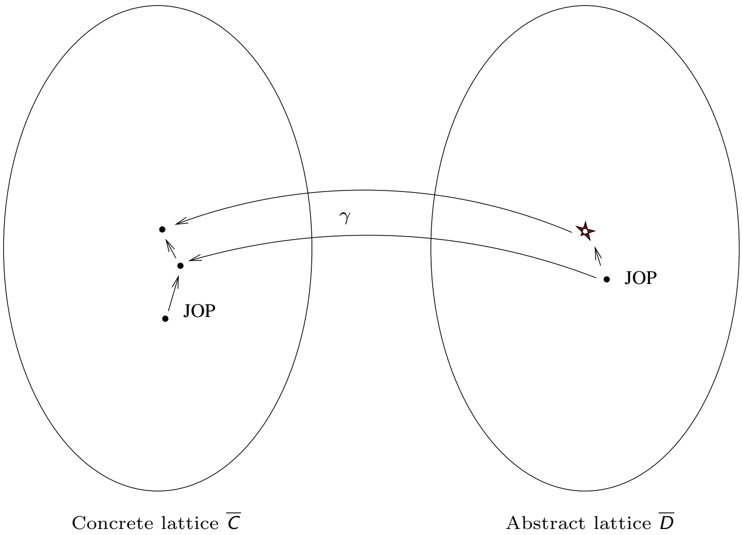
- (α, γ) form a Galois connection, and
- Each transfer function $f_{LM,D} \in F_D$ is an abstraction of the corresponding function $f_{LM,C} \in F_C$.

Definition: $f_{n,D}$ is an abstraction of $f_{n,C}$

$f_{MN,C}$ and $f_{MN,D}$ satisfy *one* of the following (each of them implies the other):



Why over-approximation of JOP in abstract lattice is useful



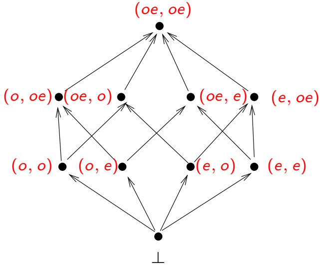
Kildall's algorithm to compute over-approximation of JOP

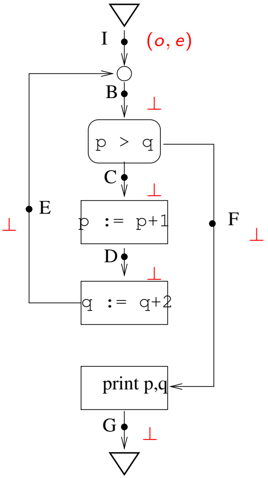
Input: An instance (P, d_0) of a monotone data-flow framework $((D, \leq), F)$.
Output: For each program point N in P, a data-value d_N such that $\mathrm{JOP}_N^{d_0} \leq d_N$.

- Initialize data value at each program point to $\perp$, entry point to d_0.
- Mark all points.
- Repeat while there is a marked point:
  - Choose a marked point M with value d_M, unmark it, and "propagate" it to successor points (i.e. for each successor N, replace value at N by $f_{MN}(d_M) \sqcup d_N$).
  - Mark successor point if old value was marked, or new value strictly dominates than old value.
- Return data values at each point as over-approx of JOP.

Kildall's algo on parity interpretation example

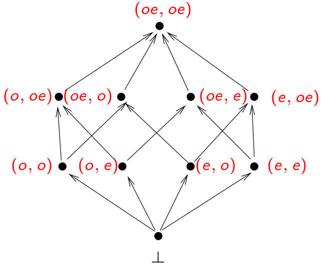
Underlying lattice

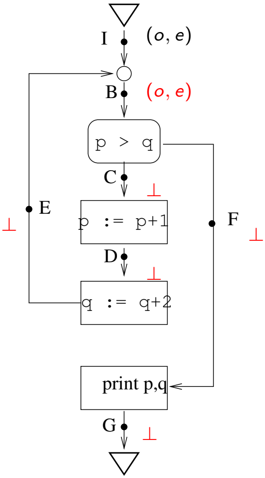




Kildall's algo on parity interpretation example

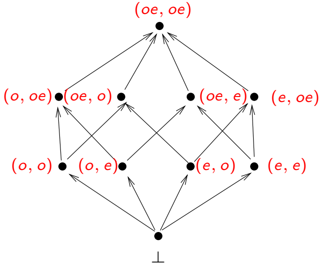
Underlying lattice

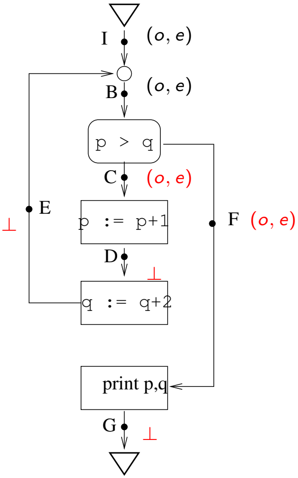




Kildall's algo on parity interpretation example

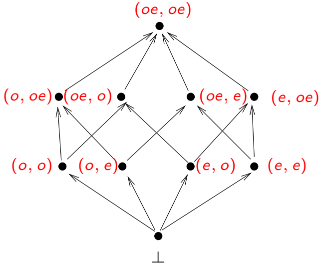
Underlying lattice

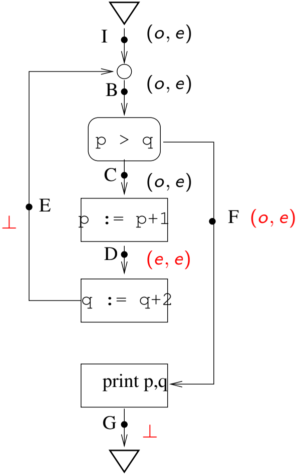




Kildall's algo on parity interpretation example

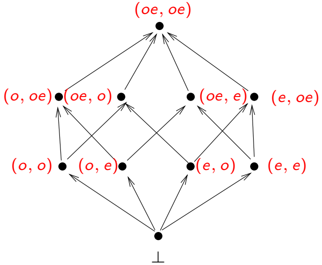
Underlying lattice

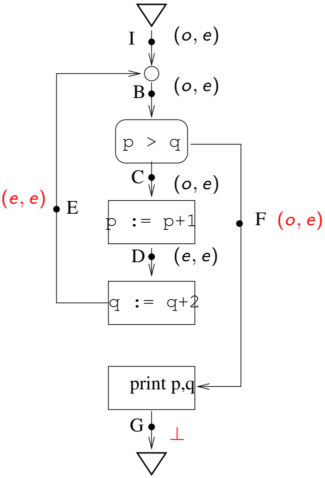




Kildall's algo on parity interpretation example

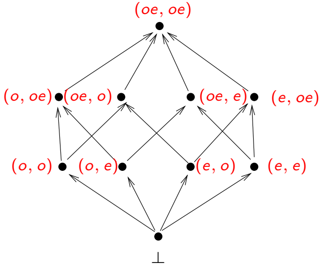
Underlying lattice

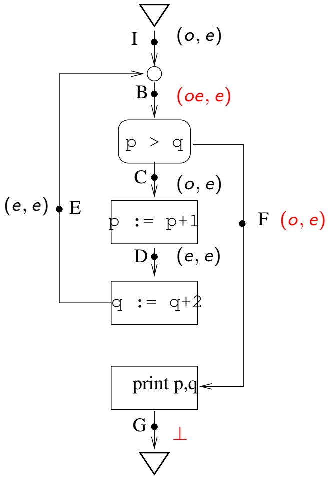




Kildall's algo on parity interpretation example

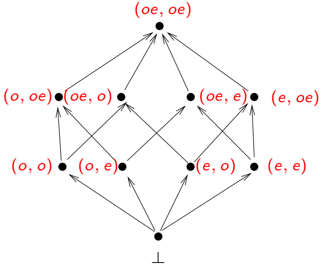
Underlying lattice

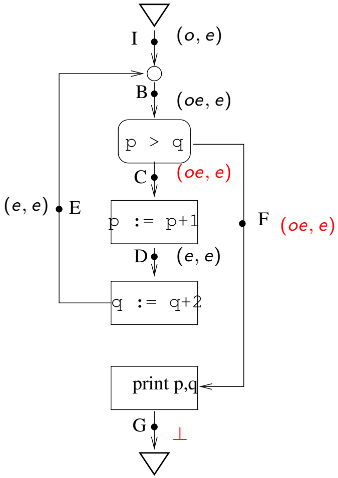




Kildall's algo on parity interpretation example

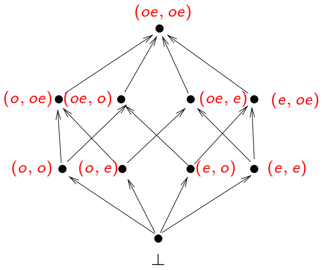
Underlying lattice

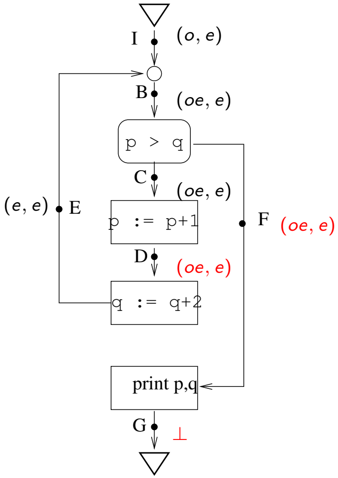




Kildall's algo on parity interpretation example

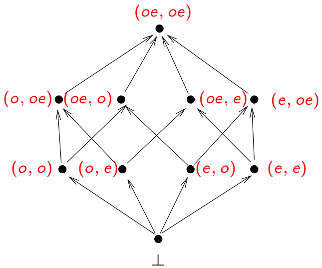
Underlying lattice

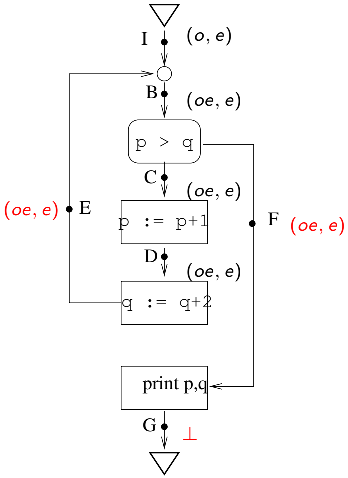




Kildall's algo on parity interpretation example

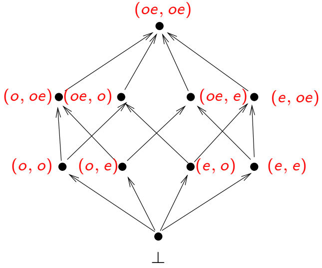
Underlying lattice

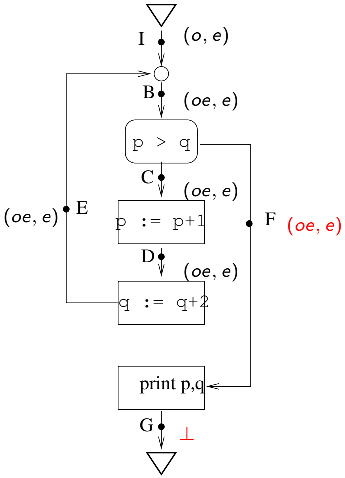




Kildall's algo on parity interpretation example

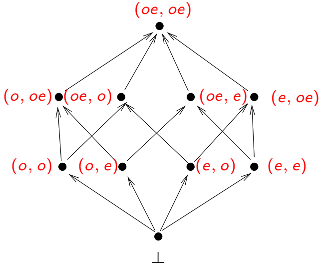
Underlying lattice

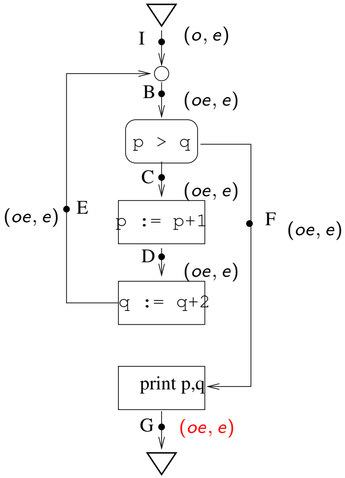




Kildall's algo on parity interpretation example

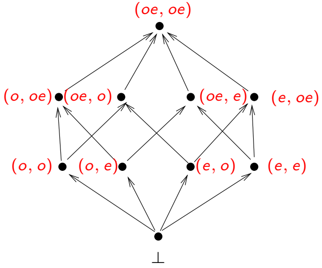
Underlying lattice

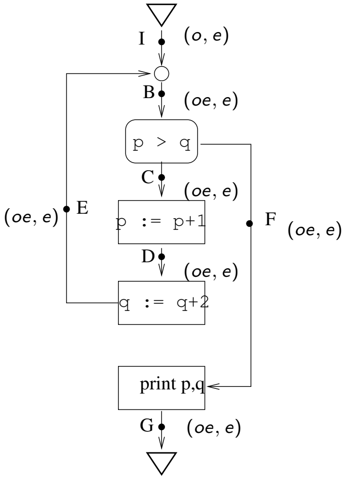




Kildall's algo on parity interpretation example

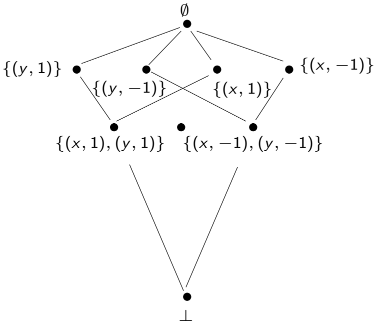
Underlying lattice

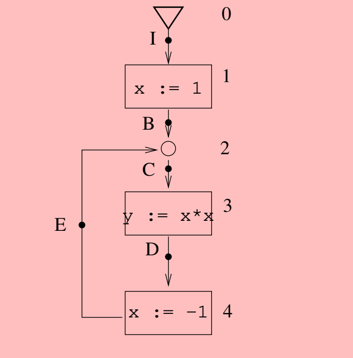




Values computed coincide with JOP values.

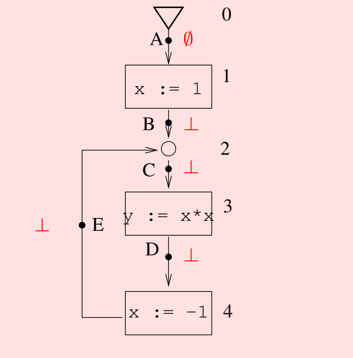
Constant propagation example



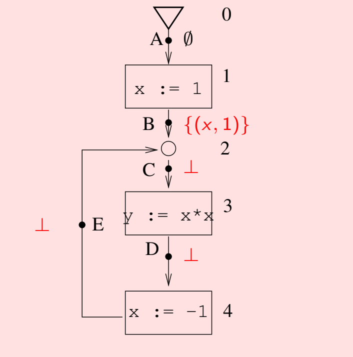


ProgPt	JOP values
A	$\emptyset$
B	$\{(x, 1)\}$
C	$\emptyset$
D	$\{(y, 1)\}$
E	$\{(x, -1), (y, 1)\}$

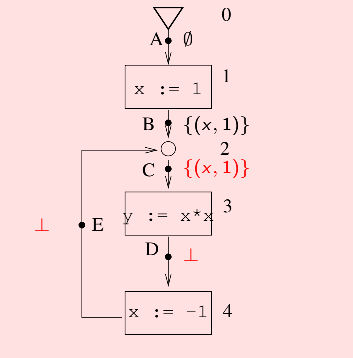
Kildall's algo on CP example: 1



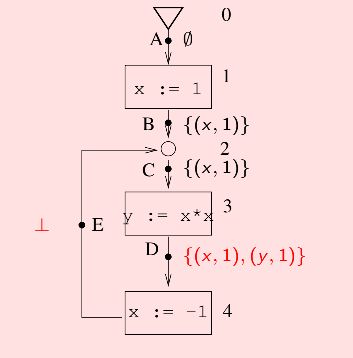
Kildall's algo on CP example: 2



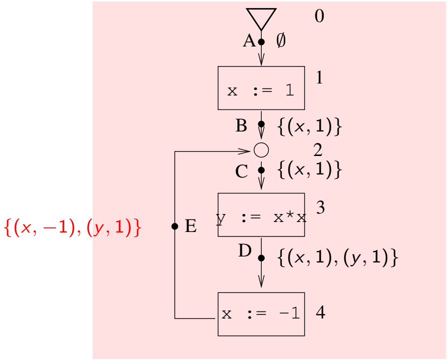
Kildall's algo on CP example: 3



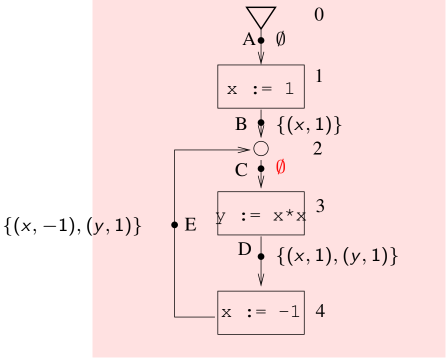
Kildall's algo on CP example: 4



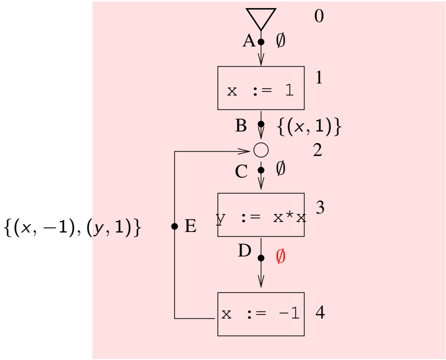
Kildall's algo on CP example: 5



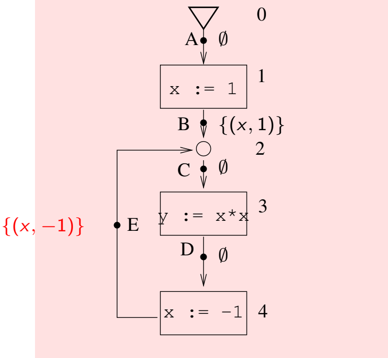
Kildall's algo on CP example: 6



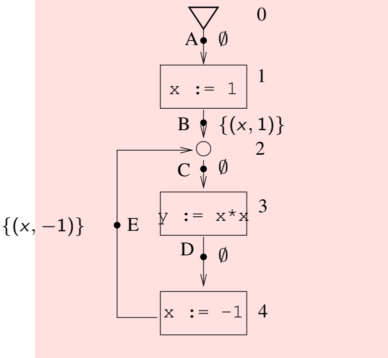
Kildall's algo on CP example: 7



Kildall's algo on CP example: 8

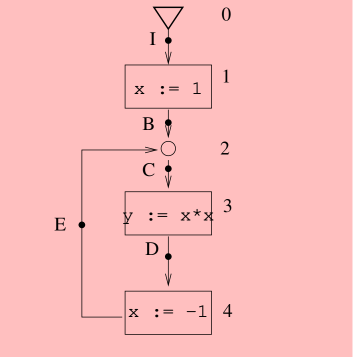


Kildall's algo on CP example: 9



Kildall's algo vs Actual Constant data

ProgPt	Actual JOP values	Kildall's data
A	$\emptyset$	$\emptyset$
B	$\{(x, 1)\}$	$\{(x, 1)\}$
C	$\emptyset$	$\emptyset$
D	$\{(y, 1)\}$	$\emptyset$
E	$\{(x, -1), (y, 1)\}$	$\{(x, -1)\}$



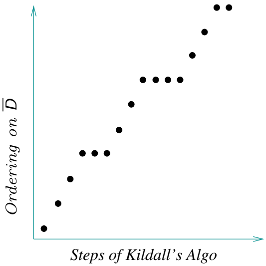
Note that Kildall's values are $\geq$ the actual JOP values at all points.

What Kildall's algo computes

- Always terminates if lattice has no infinite ascending chains.
- In general, computes the least solution to a system of equations induced by the given instance of the analysis.
- This value is always an over-approximation of the JOP for the given instance.

Termination of Kildall's algo

- Let $\overline{d}_i$ be the vector of values after the i-th step of algo.
- At step $i+1$ either $\overline{d}_{i+1}$ strictly dominates $\overline{d}_i$, or $\overline{d}_{i+1} = \overline{d}_i$. In the latter case number of marks *decreases*.
- The maximum length of any contiguous non-"climbing" sequence is equal to the number of program points.
- Moreover, the maximum number of "climbing" steps in algorithm is at most the length of any chain in the lattice $\overline{D}$.
- Therefore, the algorithm is guaranteed to terminate on finite-height lattices.



Induced Equations

The program induces a set of data-flow equations:

x_I	$=$	d_0	for entry point I
x_N	$=$	$f_{MN}(x_M)$	for an assignment or conditional node n with with incoming point M and outgoing point N
x_N	$=$	$x_L \sqcup x_M$	for a junction node with incoming points L,M and outgoing N.
	$\cdots$		etc.

Induced Equations

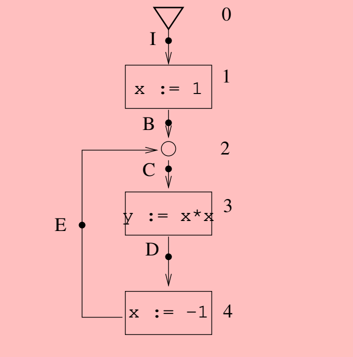
The program induces a set of data-flow equations:

x_I	$=$	d_0	for entry point I
x_N	$=$	$f_{MN}(x_M)$	for an assignment or conditional node n with with incoming point M and outgoing point N
x_N	$=$	$x_L \sqcup x_M$	for a junction node with incoming points L,M and outgoing N.
	$\cdots$		etc.

Note: The collecting semantics is a solution to the above equations.

Example equations

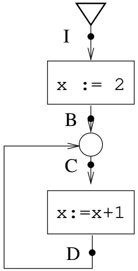
$$
\begin{aligned}
x_I &= d_0 \\
x_B &= f_1(x_I) \\
x_C &= x_B \sqcup x_E \\
x_D &= f_3(x_C) \\
x_E &= f_4(x_D).
\end{aligned}
$$



Equations can have multiple solutions

Exercise: Give two solutions to equations induced for this program

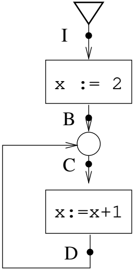
- Use lattice of subsets of concrete stores, with integer values for `x`.
- Write down induced equations.
- Give two different solutions to the equations. Let $d_0 = \mathit{State}$.



Equations can have multiple solutions

Exercise: Give two solutions to equations induced for this program

- Use lattice of subsets of concrete stores, with integer values for `x`.
- Write down induced equations.
- Give two different solutions to the equations. Let $d_0 = \mathit{State}$.

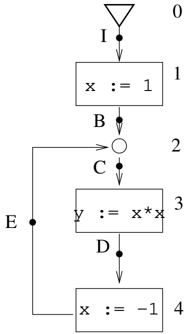


Note: collecting semantics of any program is the least solution to its data-flow equations using the concrete lattice (to be shown).

Function $\overline{f}$ induced by equations

Equations:

$$
\begin{aligned}
x_I &= d_0 \\
x_B &= f_1(x_I) \\
x_C &= x_B \sqcup x_E \\
x_D &= f_3(x_C) \\
x_E &= f_4(x_D).
\end{aligned}
$$



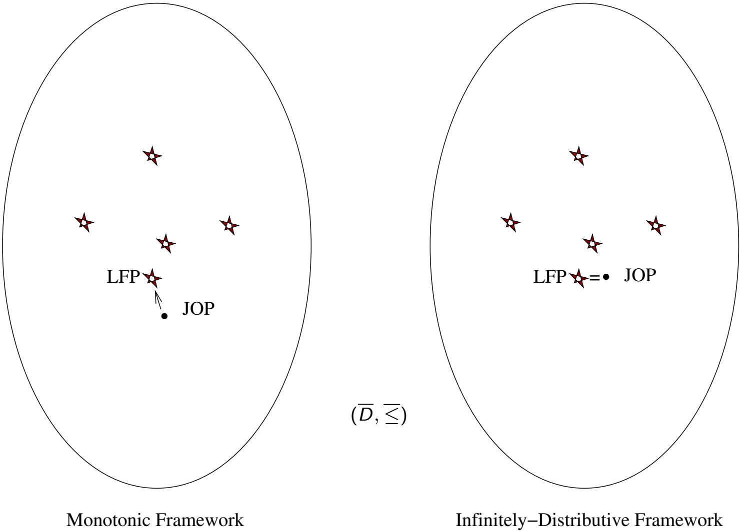
Corresponding $\overline{f}$ function:

$$\overline{f}(d_I, d_B, d_C, d_D, d_E) = (d_0, f_1(d_I), d_B \sqcup d_E, f_3(d_C), f_4(d_D)).$$

Natural ordering on solutions to Eq

- Consider "vectorised" lattice $\overline{D} = (D^k, \overline{\leq})$, where D is the underlying lattice.
- Each solution to the equations is a point in this vectorised lattice.
- The solutions are precisely the fix-points of the function $\overline{f}: \overline{D} \to \overline{D}$.
- If D is a complete lattice and f_i's are monotone, then $\overline{D}$ is complete and $\overline{f}$ is monotone.
  - Note: Concrete analysis satisfies these properties. So do many abstract interpretations.
- Therefore, Knaster-Tarski theorem applies. Therefore, there exists a least solution to $\overline{f}$.
- Kildall's algorithm computes this lfp (if it terminates).
  - So does the Kleene iteration $\perp_{\overline{D}}, \overline{f}(\perp_{\overline{D}}), \overline{f}^2(\perp_{\overline{D}}), \ldots$ if it reaches a stable value.

Correctness



Kildall's algo always computes LFP of $\overline{f}$.