Data-flow Analysis / Abstract Interpretation

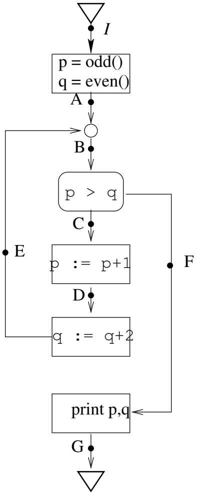
Deepak D'Souza and K. V. Raghavan

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.

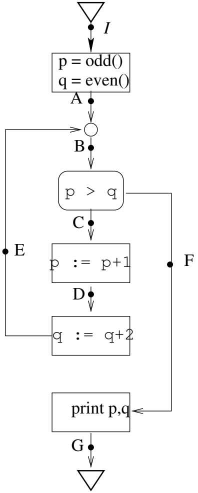
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

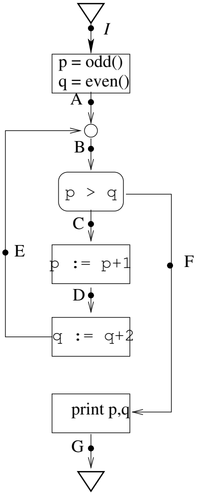
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

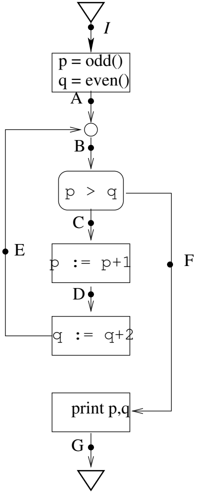
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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 – example

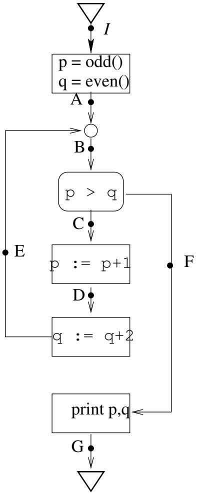
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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, $i > j\}$

Collecting semantics – example

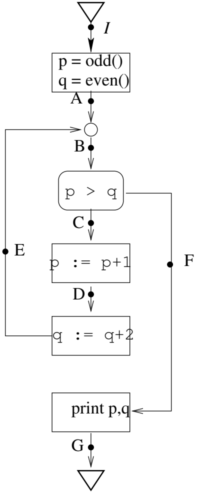
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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, $i > j+1\}$

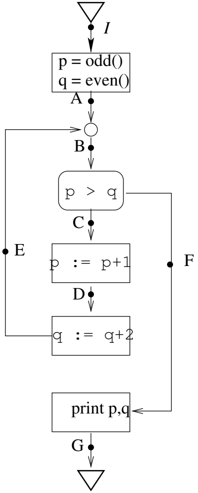
Collecting semantics – example

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



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

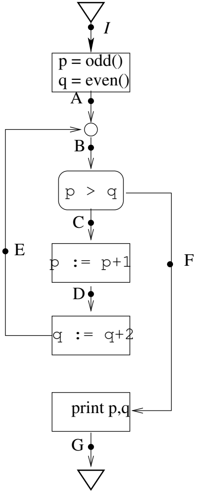
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

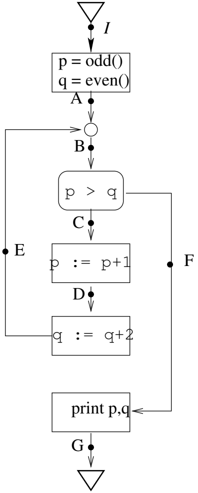
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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, $i > j\}$

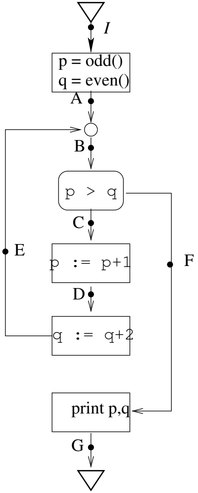
Collecting semantics – example

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 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, $i > j+1\}$
...	

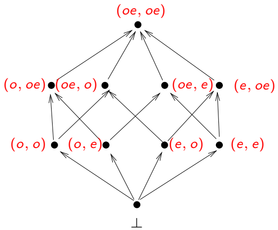
Therefore, collecting semantics:

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

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 lattice D

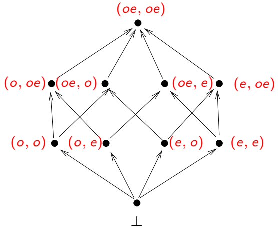


- Transfer functions for an assignment node n of the form `p := exp`

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

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

- Abstract lattice D

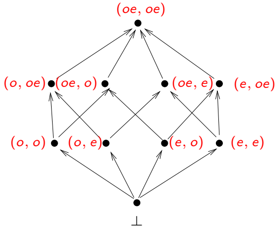


- Transfer functions for an assignment node n of the form `p := exp`

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

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

- Abstract lattice D



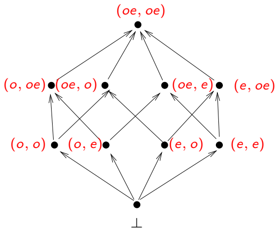
- Transfer functions for an assignment node n of the form `p := exp`

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

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

Abstract interpretation – example

- Abstract lattice D

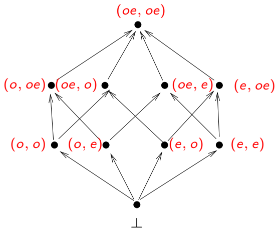


- Transfer functions for an assignment node n of the form `p := exp`

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

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

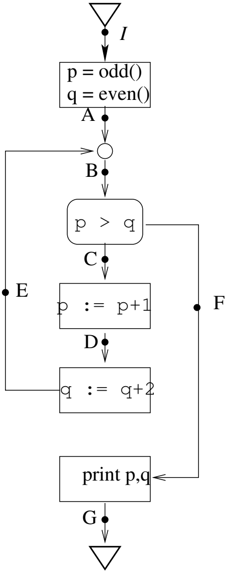
- Abstract lattice D



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

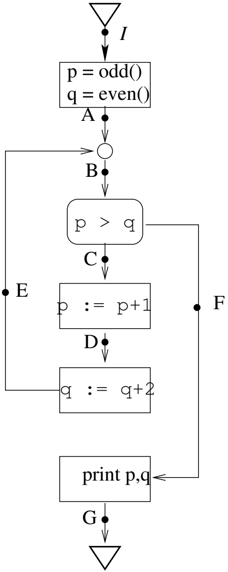
$$f_n(s) = \begin{cases} \perp & \text{if } s \text{ is } \perp \\ (o, s[2]) & \text{if exp evaluates to } o \text{ in state } s, \\ & \text{where } s[2] \text{ is the 2nd component} \\ & \text{of the pair } s \\ (e, s[2]) & \text{if exp evaluates to } e \text{ in state } s \\ (oe, s[2]) & \text{if exp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

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

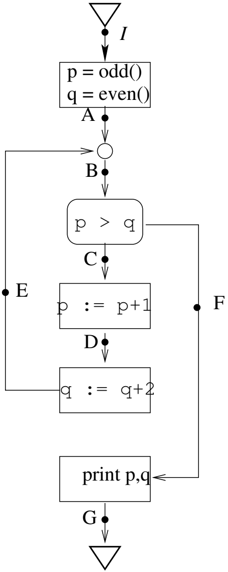


Path	Abstract value
I	(oe, oe) (given)

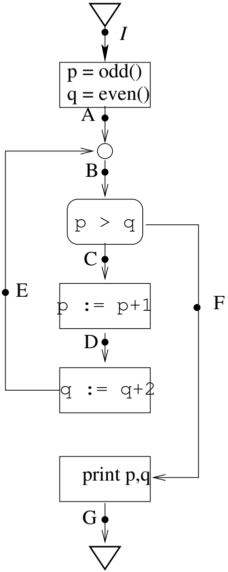
Collecting abstract values – example



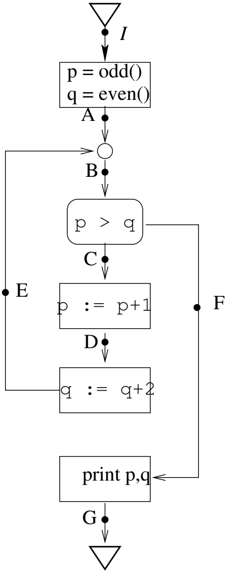
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)



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

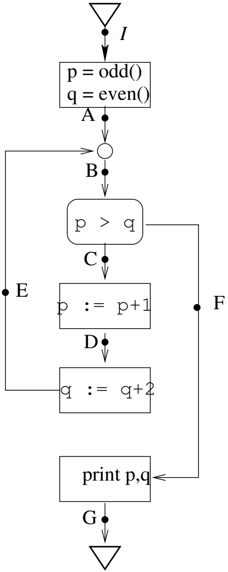


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)

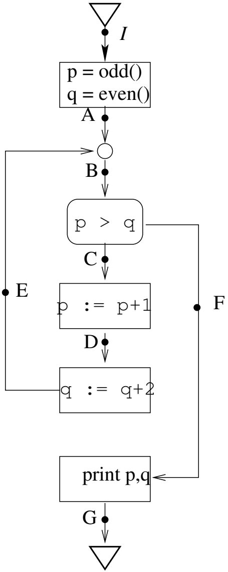


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)

Collecting abstract values – example

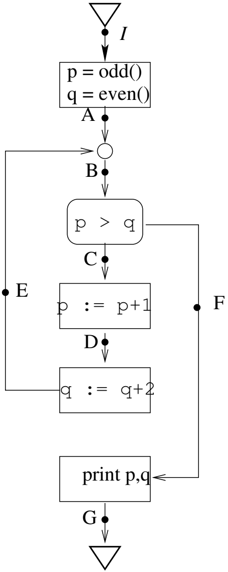


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)



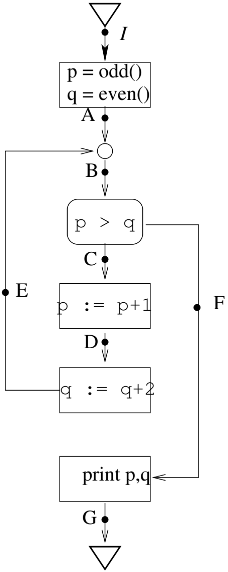
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)

Collecting abstract values – example



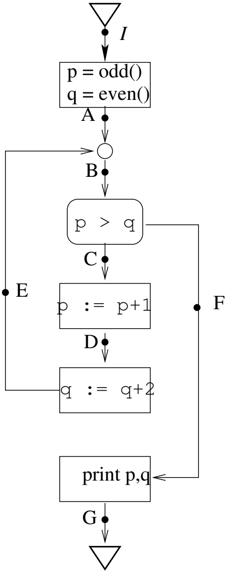
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)

Collecting abstract values – example

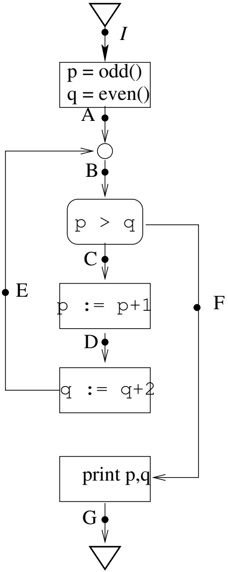


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)

Collecting abstract values – example

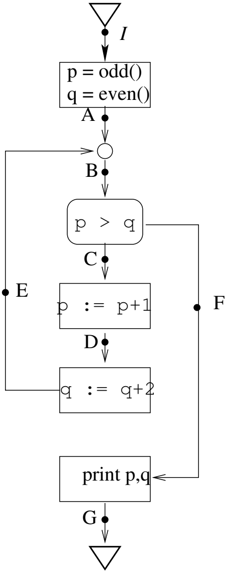


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)



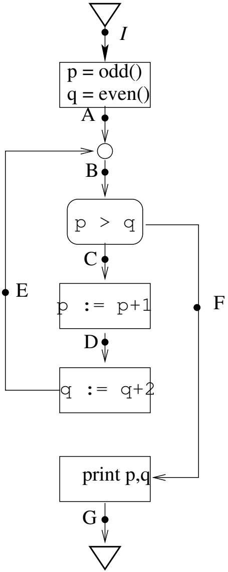
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)
IABF	(o, e)

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)

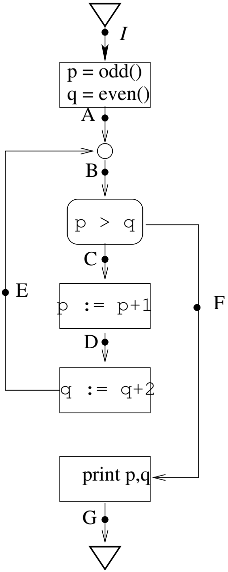
Collecting abstract values – example



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:

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)



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:

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

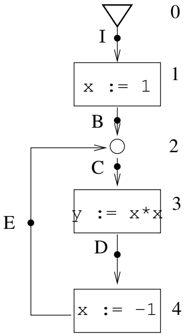
A given abstract interpretation is said to be *correct* if, for all abstract states $d_0 \in D$, for all programs P and for all program points p in P,

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.

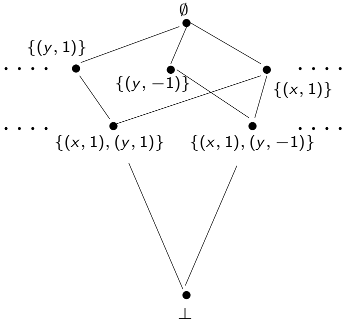


Path	Characterization of concrete states
I	*true* (given)
IB	x = 1
IBC	x = 1
IBCD	x = 1 $\wedge$ y = 1
IBCDE	x = -1 $\wedge$ y = 1
IBCDEC	x = -1 $\wedge$ y = 1
IBCDECD	x = -1 $\wedge$ y = 1
...	x = -1 $\wedge$ y = 1

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

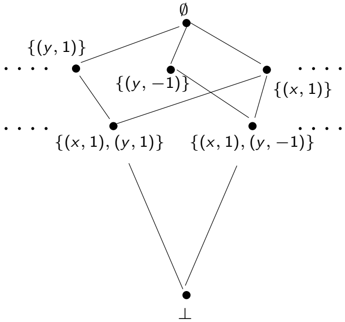
- Abstract lattice D



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

Abstract interpretation for constant propagation

- Abstract lattice D



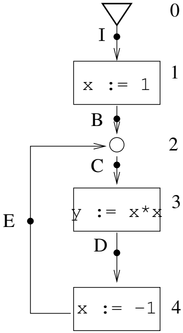
- Concretization function: What is $\gamma(d)$?

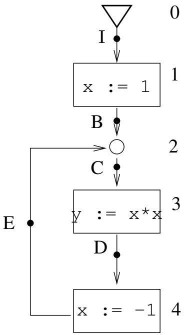
$$
\begin{array}{lcl}
\perp & \mapsto & \{\} \\
\emptyset & \mapsto & State \\
\{(v,c)\} & \mapsto & \{s \mid s \in State \wedge s[v] = c\} \\
\{(x,c),(y,d)\} & \mapsto & \{(c,d)\}
\end{array}
$$

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

$$
\begin{array}{lcll}
f_n(P) & = & \bot, & \text{if } P \text{ is } \bot \\
& = & \{(y,c) \in P \mid y \neq x\} \cup \{(x,d)\}, & \text{else if } [exp]_P = d \\
& = & \{(y,c) \in P \mid y \neq x\}, & \text{otherwise}
\end{array}
$$

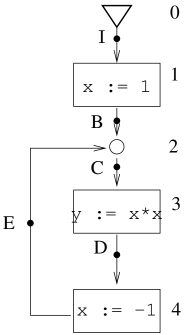
Path	Abstract value at end of path
I	$\emptyset$





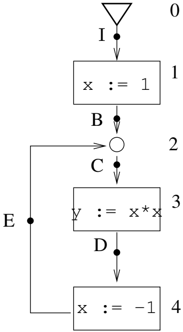
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$

JOP using abstract lattice



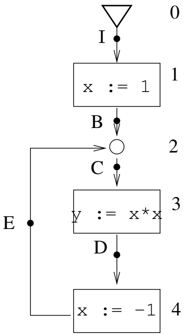
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$

JOP using abstract lattice



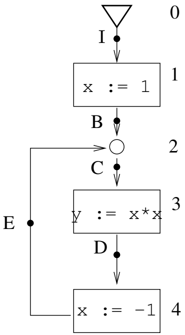
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$

JOP using abstract lattice



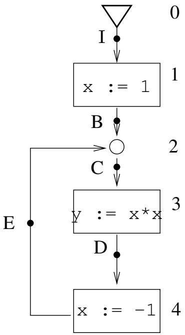
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$

JOP using abstract lattice

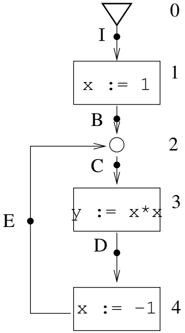


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$

JOP using abstract lattice

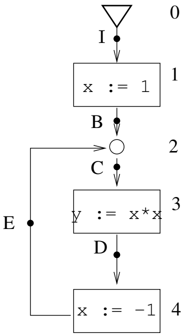


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$



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

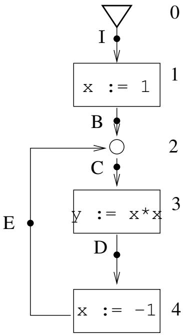
JOP using abstract lattice



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	$\emptyset$

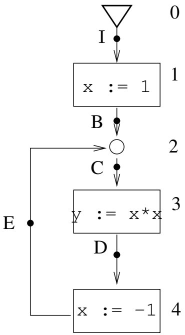
JOP using abstract lattice



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

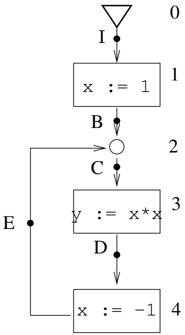
Point	Abstract JOP value
I	$\emptyset$
B	$\{(x, 1)\}$

JOP using abstract lattice



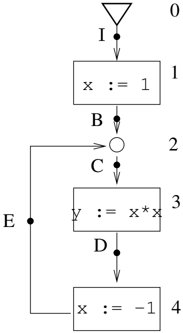
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$
...	$\{(x, -1), (y, 1)\}$

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



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	$\{(y, 1)\}$



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

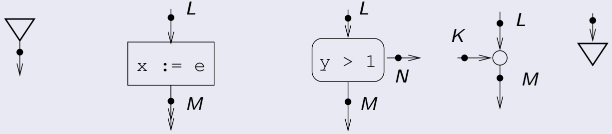
Verify that

- 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.

Formal definition of control-flow graphs

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

Nodes or statements in a program



- Expressions:

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

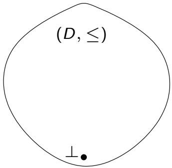
- Boolean expressions:

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

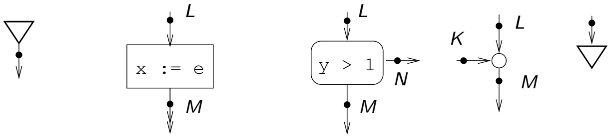
- Assume unique initial program point I.

Formal definition of an abstract interpretation

- Complete join semi-lattice $(D, \leq)$, with a least element $\perp$.
- Concretization function $\gamma : D \to 2^{State}$
- $\perp \in D$ 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:

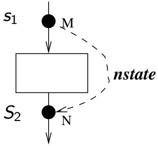
$$f_{ABCD} = f_{CD} \circ f_{BC} \circ f_{AB}.$$

  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

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

  where d_0 is a given initial abstract value at entry node.

- 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 $\mathit{ProgramPoints} \rightarrow 2^{\mathit{State}}$
- At each program point N,

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

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.