Abstract Interpretation

CS6410 Software Verification

Ashish Mishra

Credits: KV Raghavan and Deepak D'souza, IISc

Program verification

The algorithmic discovery of properties of a program by inspection of the source text.

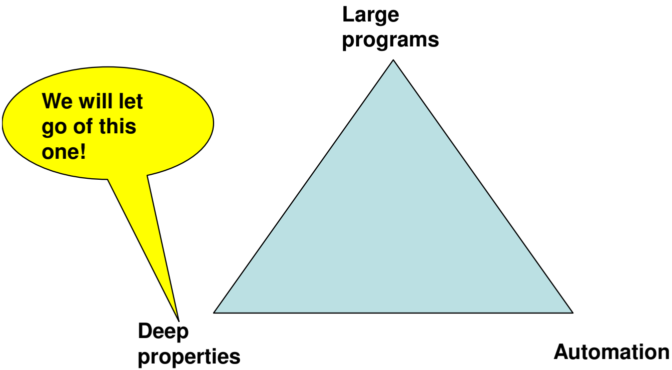
– *Manna and Pneuli, "Algorithmic Verification"*

Also known as: static analysis, static program analysis, formal methods, . . .

Difficulty of program verification

- What will we prove?
  - "Deep" specifications of complex software are as complex as the software itself
  - Are difficult to prove
  - State of the art tools and automation are not good enough
- We will focus on "shallow" properties
  - That is, we will prove "partial correctness", or absence of certain classes of low-level errors (e.g., null pointer dereferences)

Elusive triangle



Credit: Sriram Rajamani, Microsoft Research India

Example: Determining whether variables are odd (o) or even (e)

p = oddNatInput()	(p,*o*)	
q = evenNatInput()	(p,*o*)	(q,*e*)
if (p > q)	(p,*o*)	(q,*e*)
p = p*2 + q	(p,*e*)	(q,*e*)
write(p)	(p,*oe*)	(q,*e*)
if (p <= q)	(p,*o*)	(q,*e*)
p = p+1	(p,*e*)	(q,*e*)
write(p)	(p,*e*)	(q,*e*)
q = q+p	(p,*e*)	(q,*e*)

A verification approach: abstract interpretation

- A kind of program execution in which variables store *abstract* values from bounded domains, not concrete values
- Input values are also from the abstract domains
- Program statement semantics are modified to work on abstract variable values
- We execute the program on *all* (abstract) inputs and observe the program properties from these runs

Example: An abstraction

- Abstract value domain V_1 for a single variable: $\{o, e, oe\}$.
- Abstract domain:

$$L_1 = Var \rightarrow V_1$$

where Var is the set of variables in the program.

- Modified operator semantics:

$+$	o	e	oe
o	e	o	oe
e	o	e	oe
oe	oe	oe	oe

$*$	o	e	oe
o	o	e	oe
e	e	e	e
oe	oe	e	oe

- From the operator semantics, we can construct an abstract transfer function, with signature $L_1 \rightarrow L_1$, for each possible statement in the language.

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	$<(p,oe), (q,oe)>$
p=oddNatInput()	
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	$<$(p,oe), (q,oe)$>$
p=oddNatInput()	$<$(p,o), (q,oe)$>$
q=evenNatInput()	$<$(p,o), (q,e)$>$
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	<(p,*o*), (q,*e*)>
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	<(p,*o*), (q,*e*)>
p = p*2 + q	<(p,*e*), (q,*e*)>
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	$<$(p,oe), (q,oe)$>$	
p=oddNatInput()	$<$(p,o), (q,oe)$>$	
q=evenNatInput()	$<$(p,o), (q,e)$>$	
if (p $>$ q)	$<$(p,o), (q,e)$>$	
p = p*2 + q		$<$(p,e), (q,e)$>$
write(p)	$<$(p,oe), (q,e)$>$	
if (p $<=$ q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	<(p,*o*), (q,*e*)>
p = p*2 + q	<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>
if (p <= q)	<(p,*oe*), (q,*e*)>
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	<(p,*o*), (q,*e*)>
p = p*2 + q	<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>
if (p <= q)	<(p,*oe*), (q,*e*)>
p = p+1	<(p,*oe*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>
q = q+p	<(p,*oe*), (q,*oe*)>

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
q = q+p	<(p,*oe*), (q,*oe*)>	

Ideal results

(p,*o*)	
(p,*o*)	(q,*e*)
(p,*o*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*oe*)	(q,*e*)
(p,*o*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*e*)	(q,*e*)

Example: The abstract interpretation

	Abstract interpretation, using domain L_1		Ideal results	
	<(p,*oe*), (q,*oe*)>			
p=oddNatInput()	<(p,*o*), (q,*oe*)>		(p,*o*)	
q=evenNatInput()	<(p,*o*), (q,*e*)>		(p,*o*)	(q,*e*)
if (p > q)	<(p,*o*), (q,*e*)>		(p,*o*)	(q,*e*)
p = p*2 + q		<(p,*e*), (q,*e*)>	(p,*e*)	(q,*e*)
write(p)	<(p,*oe*), (q,*e*)>		(p,*oe*)	(q,*e*)
if (p <= q)	X <(p,*oe*), (q,*e*)>		(p,*o*)	(q,*e*)
p = p+1		X <(p,*oe*), (q,*e*)>	(p,*e*)	(q,*e*)
write(p)	X <(p,*oe*), (q,*e*)>		(p,*e*)	(q,*e*)
q = q+p	X <(p,*oe*), (q,*oe*)>		(p,*e*)	(q,*e*)

Example: Another abstraction

- Abstract value domain V_2 for a single variable: $\{o, e\}$.
- The alternative domain:

  $$L_2 \ = \ 2^{Var \to V_2}$$

  where *Var* is the set of variables in the program.
- Same operator tables as before.
- From the operator semantics, we can construct an abstract transfer function, $L_2 \to L_2$, for each possible statement in the language.

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,o), (q,o)>, <(p,o), (q,e)>
	<(p,e), (q,o)>, <(p,e), (q,e)>}
p=oddNatInput()	,
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{<(p,o), (q,o)>, <(p,o), (q,e)>$
	$<(p,e), (q,o)>, <(p,e), (q,e)>\}$
p=oddNatInput()	$\{<(p,o), (q,o)>, <(p,o), (q,e)>\}$
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{<(\mathrm{p},o),\ (\mathrm{q},o)>,\ <(\mathrm{p},o),\ (\mathrm{q},e)>$ $<(\mathrm{p},e),\ (\mathrm{q},o)>,\ <(\mathrm{p},e),\ (\mathrm{q},e)>\}$
p=oddNatInput()	$\{<(\mathrm{p},o),\ (\mathrm{q},o)>,\ <(\mathrm{p},o),\ (\mathrm{q},e)>\}$
q=evenNatInput()	$\{<(\mathrm{p},o),\ (\mathrm{q},e)>\}$
if (p > q)	$\{<(\mathrm{p},o),\ (\mathrm{q},e)>\}$
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{\langle(p,o), (q,o)\rangle,\ \langle(p,o), (q,e)\rangle$ $\langle(p,e), (q,o)\rangle,\ \langle(p,e), (q,e)\rangle\}$
p=oddNatInput()	$\{\langle(p,o), (q,o)\rangle,\ \langle(p,o), (q,e)\rangle\}$
q=evenNatInput()	$\{\langle(p,o), (q,e)\rangle\}$
if (p > q)	$\{\langle(p,o), (q,e)\rangle\}$
p = p*2 + q	$\{\langle(p,e), (q,e)\rangle\}$
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{<(p,o), (q,o)>,$	$<(p,o), (q,e)>$
	$<(p,e), (q,o)>,$	$<(p,e), (q,e)>\}$
p=oddNatInput()	$\{<(p,o), (q,o)>,$	$<(p,o), (q,e)>\}$
q=evenNatInput()	$\{<(p,o), (q,e)>\}$	
if (p > q)	$\{<(p,o), (q,e)>\}$	
p = p*2 + q		$\{<(p,e), (q,e)>\}$
write(p)	$\{<(p,o), (q,e)>,$	
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}
if (p > q)	{<(p,*o*), (q,*e*)>}
p = p*2 + q	{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}
if (p > q)	{<(p,*o*), (q,*e*)>}
p = p*2 + q	{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}
if (p <= q)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,o), (q,o)>, <(p,e), (q,o)>,	<(p,o), (q,e)> <(p,e), (q,e)>}
p=oddNatInput()	{<(p,o), (q,o)>,	<(p,o), (q,e)>}
q=evenNatInput()	{<(p,o), (q,e)>}	
if (p > q)	{<(p,o), (q,e)>}	
p = p*2 + q		{<(p,e), (q,e)>}
write(p)	{<(p,o), (q,e)>,	<(p,e), (q,e)>}
if (p <= q)	{<(p,o), (q,e)>,	<(p,e), (q,e)>}
p = p+1	{<(p,e), (q,e)>,	<(p,o), (q,e)>}
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{<(p,o), (q,o)>, \ <(p,o), (q,e)>$
	$<(p,e), (q,o)>, \ <(p,e), (q,e)>\}$
p=oddNatInput()	$\{<(p,o), (q,o)>, \ <(p,o), (q,e)>\}$
q=evenNatInput()	$\{<(p,o), (q,e)>\}$
if (p > q)	$\{<(p,o), (q,e)>\}$
p = p*2 + q	$\{<(p,e), (q,e)>\}$
write(p)	$\{<(p,o), (q,e)>, \ <(p,e), (q,e)>\}$
if (p <= q)	$\{<(p,o), (q,e)>, \ <(p,e), (q,e)>\}$
p = p+1	$\{<(p,e), (q,e)>, \ <(p,o), (q,e)>\}$
write(p)	$\{<(p,e), (q,e)>, \ <(p,o), (q,e)>\}$
q = q+p	

Example: The abstract interpretation

	Abstract interpretation, using domain L_2	Ideal results
	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}	
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}	(p,*o*)
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	(p,*o*) (q,*e*)
if (p > q)	{<(p,*o*), (q,*e*)>}	(p,*o*) (q,*e*)
p = p*2 + q	{<(p,*e*), (q,*e*)>}	(p,*e*) (q,*e*)
write(p)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}	(p,*oe*) (q,*e*)
if (p <= q)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}	(p,*o*) (q,*e*)
p = p+1	{<(p,*e*), (q,*e*)>, <(p,*o*), (q,*e*)>}	(p,*e*) (q,*e*)
write(p)	{<(p,*e*), (q,*e*)>, <(p,*o*), (q,*e*)>}	(p,*e*) (q,*e*)
q = q+p	{<(p,*e*), (q,*e*)>, <(p,*o*), (q,*o*)>}	(p,*e*) (q,*e*)

Example: The abstract interpretation

Abstract interpretation, using domain L_2		Ideal results
	{<(p,o), (q,o)>, <(p,o), (q,e)> <(p,e), (q,o)>, <(p,e), (q,e)>}	
p=oddNatInput()	{<(p,o), (q,o)>, <(p,o), (q,e)>}	(p,o)
q=evenNatInput()	{<(p,o), (q,e)>}	(p,o) (q,e)
if (p > q)	{<(p,o), (q,e)>}	(p,o) (q,e)
p = p*2 + q	{<(p,e), (q,e)>}	(p,e) (q,e)
write(p)	{<(p,o), (q,e)>, <(p,e), (q,e)>}	(p,oe) (q,e)
if (p <= q)	{<(p,o), (q,e)>, X<(p,e), (q,e)>}	(p,o) (q,e)
p = p+1	{<(p,e), (q,e)>, X<(p,o), (q,e)>}	(p,e) (q,e)
write(p)	{<(p,e), (q,e)>, X<(p,o), (q,e)>}	(p,e) (q,e)
q = q+p	{<(p,e), (q,e)>, X<(p,o), (q,o)>}	(p,e) (q,e)

Example: The abstract interpretation

Program	Abstract interpretation, using domain L_2	Ideal results
	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}	
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}	(p,*o*)
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	(p,*o*) (q,*e*)
if (p > q)	{<(p,*o*), (q,*e*)>}	(p,*o*) (q,*e*)
p = p*2 + q	{<(p,*e*), (q,*e*)>}	(p,*e*) (q,*e*)
write(p)	{<(p,*o*), (q,*e*)>, <(p,*e*), (q,*e*)>}	(p,*oe*) (q,*e*)
if (p <= q)	{<(p,*o*), (q,*e*)>, **X**<(p,*e*), (q,*e*)>}	(p,*o*) (q,*e*)
p = p+1	{<(p,*e*), (q,*e*)>, **X**<(p,*o*), (q,*e*)>}	(p,*e*) (q,*e*)
write(p)	{<(p,*e*), (q,*e*)>, **X**<(p,*o*), (q,*e*)>}	(p,*e*) (q,*e*)
q = q+p	{<(p,*e*), (q,*e*)>, **X**<(p,*o*), (q,*o*)>}	(p,*e*) (q,*e*)

In comparison to the L_1 domain

- L_2 is a *more precise* domain. Result at the end of the program was <(p,*oe*),(q,*oe*)> with L_1, which *over-approximates* $\{<(p,e),(q,e)>,<(p,o),(q,o)>\}$.
- However, L_1 is *more efficient*.
- Both are less precise than ideal!

Other examples of verification problems

Analysis	*Abstract domain*
Null-pointer deref.	$\mathit{Var} \rightarrow \{\mathit{not\text{-}pointer}, \mathit{null}, \mathit{non\text{-}null}\} \times 2^{\mathit{Var}}$
Array overruns	$\mathit{Var} \rightarrow \mathit{IntegerRanges}$
File IO	$\mathit{File\text{-}handles} \rightarrow \mathit{Files}, \mathit{Files} \rightarrow \{\mathit{open}, \mathit{closed}\},$
Reachability	Reachability condition
Mutual exclusion	set of locks taken

Other applications of program analysis

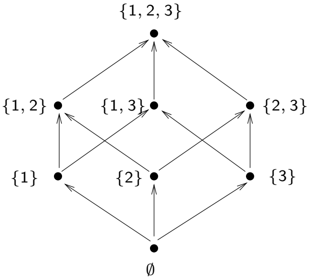
- Compilers
  - Live variables analysis
    - Useful, e.g., for register allocation
  - Side-effect analysis of functions
    - Useful, e.g., for code motion
  - Interaction between statements
    - Useful, e.g., for separating sequential code into independent threads
- Code development tools
  - Refactoring; e.g., rename method, extract method
  - Generating code automatically from specifications
  - Automated generation of test cases

Lattice Theory

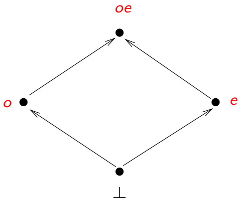
Outline

1. Why study lattices
2. Partial Orders
3. Lattices
4. Knaster-Tarski Theorem
5. Computing LFP

What a lattice looks like



Subsets of $\{1,2,3\}$, "subset"



Odd/even, "contained in"

Why study lattices in program analysis?

Why lattices?

- Natural way to obtain the "collecting state" at a point is to take union of states reached along each path leading to the point.
- With abstract states also we want a "union" or "join" over all paths (JOP).

Why fixpoints?

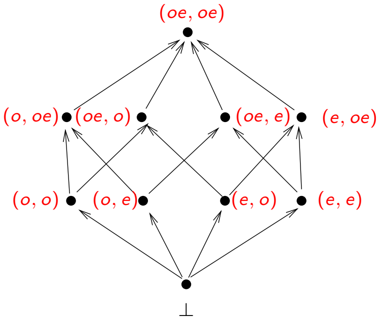
- Guaranteed to safely approximate JOP (* Conditions apply).
- Easier to compute than JOP.
- Knaster-Tarski theorem tells us about the existence of fixpoints and their structure in a lattice.

Motivation: Interpreting a program with even/odd abstract values

```
1:  p := 5;
2:  q := 2;
3:  while (p > q) {
4:    p := p+1;
5:    q := q+2;
    }
6:  print p;
```

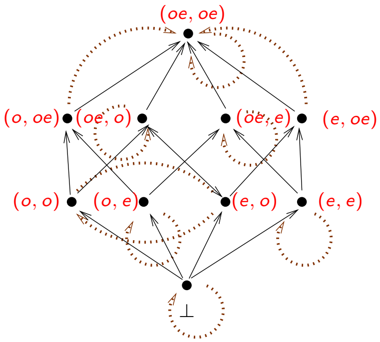
Motivation: Interpreting a program with even/odd abstract values

```
1:  p := 5;
2:  q := 2;
3:  while (p > q) {
4:    p := p+1;
5:    q := q+2;
    }
6:  print p;
```



Why Fixed Points?

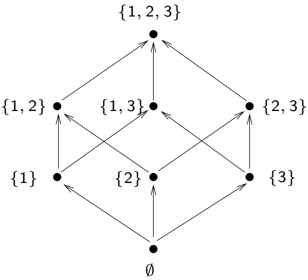
- JOP not always possible to compute
- LFP guaranteed to conservatively approximate JOP
- More efficient to compute LFP



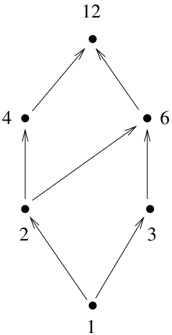
Transfer function for `p:=p+q`

Partial Orders

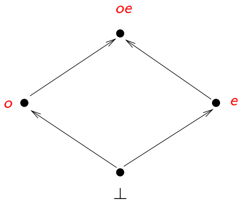
- Usual order (or total order) on numbers: $1 \leq 2 \leq 3$.
- Some domains are naturally "partially" ordered:



Subsets of $\{1, 2, 3\}$, "subset"



Divisors of 12, "divides"



Odd/even, "contained in"

Partial orders: definition

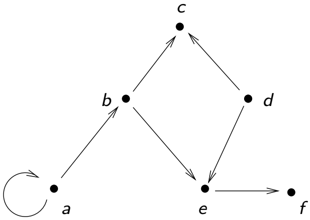
- A partially ordered set is a non-empty set D along with a partial order $\leq$ on D. Thus $\leq$ is a binary relation on D satisfying:
  - $\leq$ is reflexive ($d \leq d$ for each $d \in D$)
  - $\leq$ is transitive ($d \leq d'$ and $d' \leq d''$ implies $d \leq d''$)
  - $\leq$ is anti-symmetric ($d \leq d'$ and $d' \leq d$ implies $d = d'$).

Binary relations as Graphs

We can view a binary relation on a set as a directed graph.
For example, the binary relation

$$\{(a,a),(a,b),(b,c),(b,e),(d,e),(d,c),(e,f)\}$$

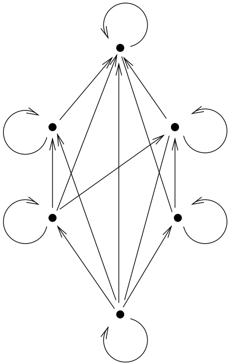
can be represented as the graph:



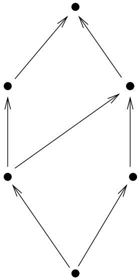
Partial Order as a graph

A partial order is then a special kind of directed graph:

- Reflexive = self-loop on each node
- Antisymmetric = no 2-length cycles
- Transitive = "transitivity" of edges.



Graph representation

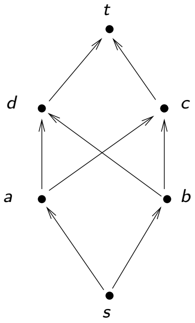


Hasse-diagram representation

Upper bounds etc.

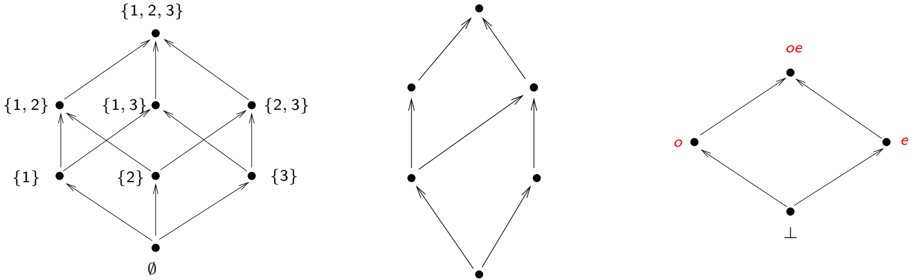
In a partially ordered set $(D, \leq)$:

- An element $u \in D$ is an upper bound of a set of elements $X \subseteq D$, if $x \leq u$ for all $x \in X$.
- u is the least upper bound (or lub or join) of X if u is an upper bound for X, and for every upper bound y of X, we have $u \leq y$. We write $u = \bigsqcup X$.
- Similarly, $v = \bigsqcap X$ (v is the greatest lower bound or glb or meet of X).



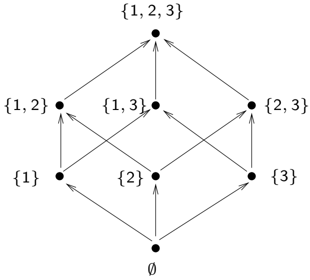
Lattices

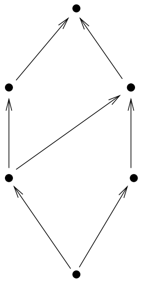
- A lattice is a partially order set in which every pair of elements has an lub and a glb.
- A complete lattice is a lattice in which every subset of elements has a lub and glb.

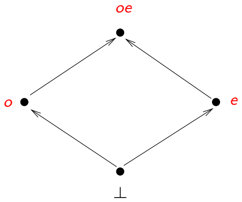


Lattices

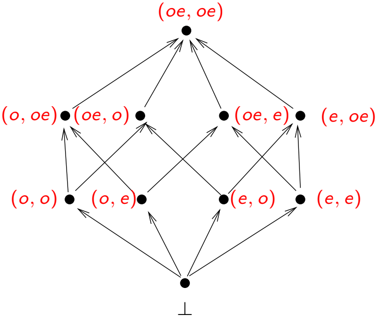
- A lattice is a partially order set in which every pair of elements has an lub and a glb.
- A complete lattice is a lattice in which every subset of elements has a lub and glb.
- Examples below are all complete lattices.







More lattices

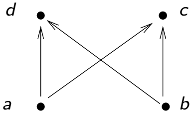


Exercise

1. Example of a partial order that is not a lattice?

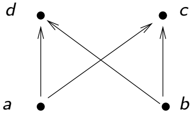
Exercise

1. Example of a partial order that is not a lattice?



Exercise

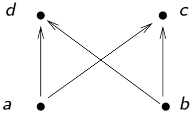
1. Example of a partial order that is not a lattice?



2. "Simplest" example of a partial order that is not a lattice?

Exercise

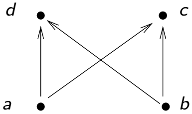
1. Example of a partial order that is not a lattice?



2. "Simplest" example of a partial order that is not a lattice?

Exercise

1. Example of a partial order that is not a lattice?

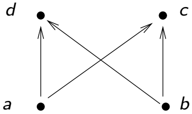


2. "Simplest" example of a partial order that is not a lattice?

3. Example of a lattice which is not complete?

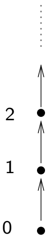
Exercise

1. Example of a partial order that is not a lattice?



2. "Simplest" example of a partial order that is not a lattice?

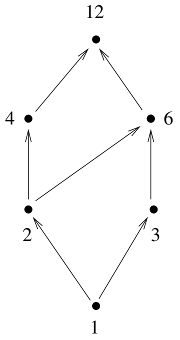
3. Example of a lattice which is not complete?

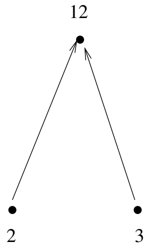


Partial order induced by a subset of elements

Let $(D, \leq)$ be a partially ordered set, and X be a non-empty subset of D. Then X induces a partial order, which we call the partial order *induced by* X in $(D, \leq)$, and defined to be $(X, \leq \cap (X \times X))$.

Example: the partial order induced by the set of elements $X = \{2, 3, 12\}$.

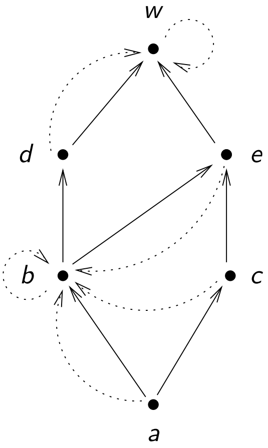




Monotonic functions

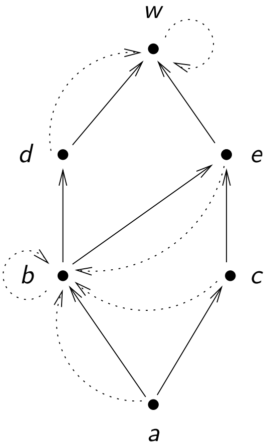
Let $(D, \leq)$ be a partially ordered set.

- A function $f : D \to D$ is monotonic or order-preserving if whenever $x \leq y$ we have $f(x) \leq f(y)$.



Fixpoints

- A fixpoint of a function $f : D \to D$ is an element $x \in D$ such that $f(x) = x$.
- A pre-fixpoint of f is an element x such that $x \leq f(x)$.
- A post-fixpoint of f is an element x such that $f(x) \leq x$.



Knaster-Tarski Fixpoint Theorem

Theorem (Knaster-Tarski)

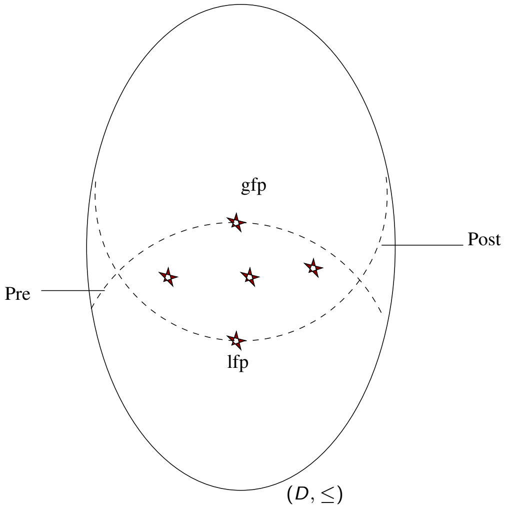
Let $(D, \leq)$ be a complete lattice, and $f : D \to D$ a monotonic function on $(D, \leq)$. Then:

(a) *f has at least one fixpoint.*

(b) *f has a least fixpoint which coincides with the glb of the set of postfixpoints of f, and a greatest fixpoint which coincides with the lub of the prefixpoints of f.*

(c) *The set of fixpoints P of f itself forms a complete lattice under $\leq$.*

Fixpoints of f



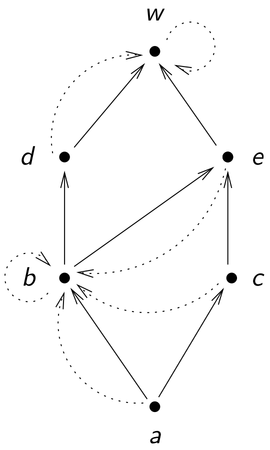
Stars denote fixpoints.

Exercise

Consider the complete lattice and monotone function f below.

1. Mark the pre-fixpoints with up-triangles ($\triangle$).
2. What is the lub of the pre-fixpoints?
3. Mark post-fixpoints with down-triangles ($\triangledown$).
4. Fixpoints are the stars (✡).

Check that claims of K-T theorem hold here.



Exercise

If you drop one of the conditions of the K-T theorem

- Monotonicity of the function f
- Completeness of the lattice

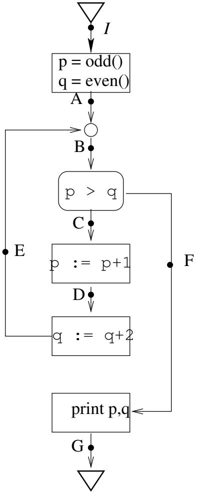
does the conclusion of the theorem still hold?

Abstract Interpretation: Formally

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.

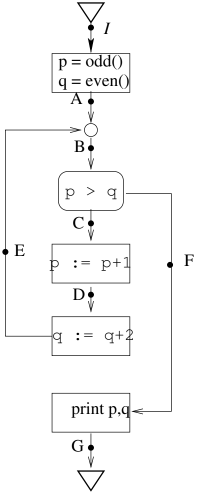
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

Collecting semantics – example

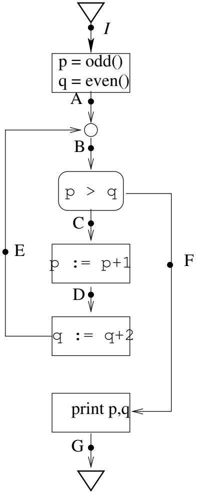
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

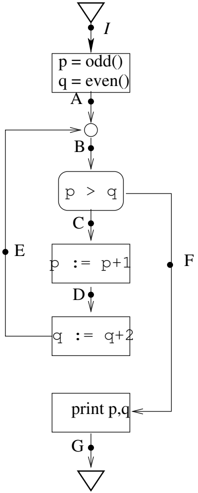
Collecting semantics – example

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$\}$

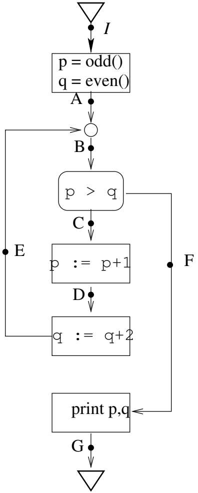
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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

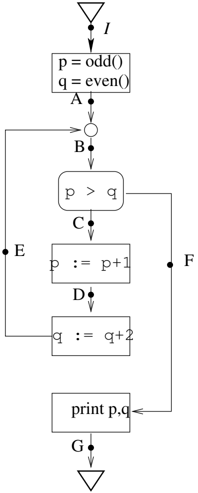
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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, $i > j+1\}$

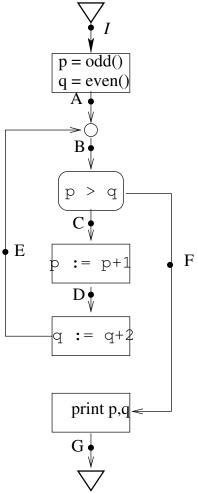
Collecting semantics – example

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



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

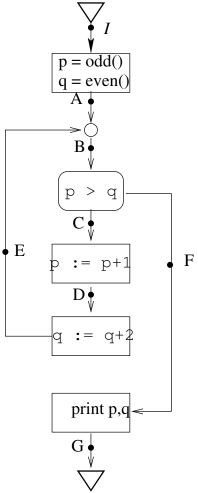
Collecting semantics of a program = set of (concrete) states occurring at each program point.



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, $i \geq j\}$

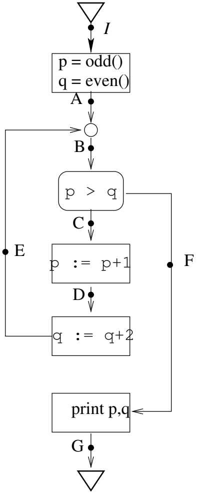
Collecting semantics – example

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



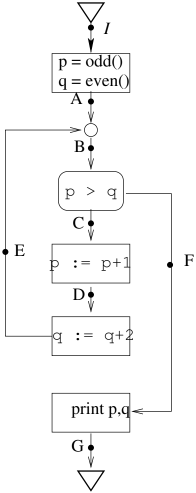
Path	Concrete states
I	$\{(i,j)\}$ (given)
IA	$\{(i,j) \| i$ is odd, j is even$\}$
IAB	$\{(i,j) \| i$ is odd, j is even$\}$
IABC	$\{(i,j) \| i$ is odd, j is even, $i > j\}$
IABCD	$\{(i,j) \| i$ is even, j is even, $i > j+1\}$
IABCDE	$\{(i,j) \| i$ is even, j is even, $i \geq j\}$
IABCDEB	$\{(i,j) \|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

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, $i \geq j\}$
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	$\{(i,j) \mid i \text{ is odd}, j \text{ is even}\}$
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	$\{(i,j) \mid i \text{ is even}, j \text{ is even}, i \geq j\}$
IABCDEB	$\{(i,j) \mid i \text{ is even}, j \text{ is even}, i \geq j\}$
IABCDEBC	$\{(i,j) \mid i \text{ is even}, j \text{ is even}, i > j\}$
IABCDEBCD	$\{(i,j) \mid i \text{ is odd}, j \text{ is even}, i > j+1\}$

...

Therefore, collecting semantics:

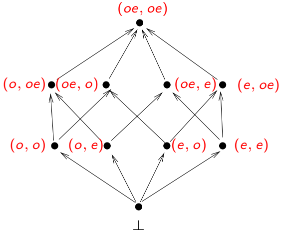
I	$\{(i,j)\}$
A	$\{(i,j) \mid i \text{ odd}, j \text{ even}\}$
B	$\{(i,j) \mid i \text{ odd}, j \text{ even}\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i \geq j\}$
C	$\{(i,j) \mid j \text{ even}, i > j\}$
D	$\{(i,j) \mid j \text{ even}, i > j+1\}$
E	$\{(i,j) \mid j \text{ even}, i \geq j\}$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

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

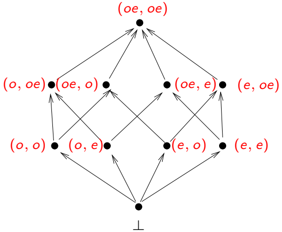


- Transfer function for an assignment node n: `p := p+q`

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

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

- Abstract lattice D



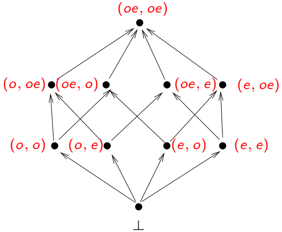
- Transfer function for an assignment node n: `p := p+q`

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

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

Abstract interpretation – example

- Abstract lattice D



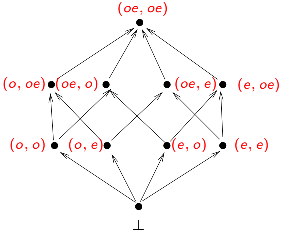
- Transfer function for an assignment node n: `p := p+q`

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

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

Abstract interpretation – example

- Abstract lattice D



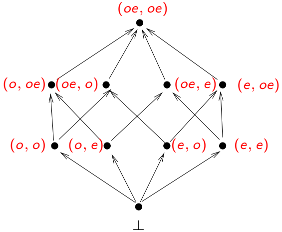
- Transfer function for an assignment node n: `p := p+q`

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

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

Abstract interpretation – example

- Abstract lattice D

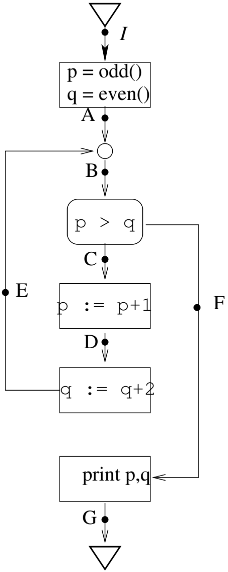


- Transfer function for an assignment node n: `p := p+q`

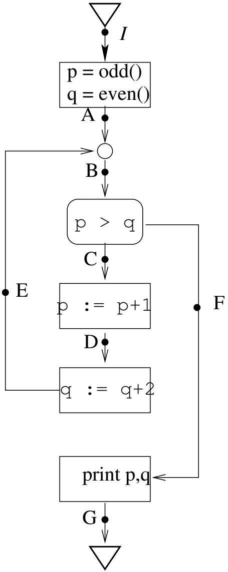
$$f_n(s) = \begin{cases} \bot & \text{if } s \text{ is } \bot \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o}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

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

Collecting abstract values – example

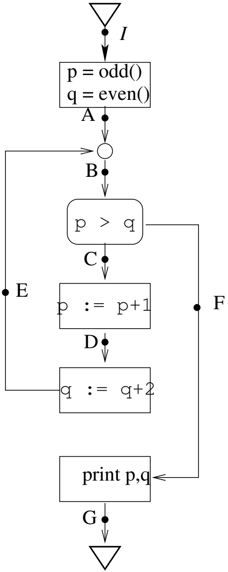


Path	Abstract value
I	(oe, oe) (given)

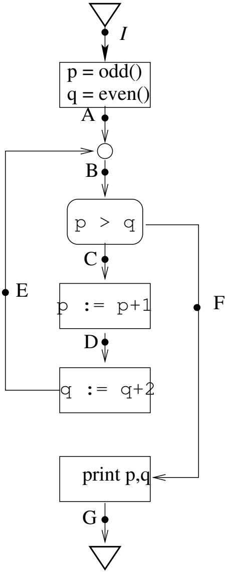


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)

Collecting abstract values – example

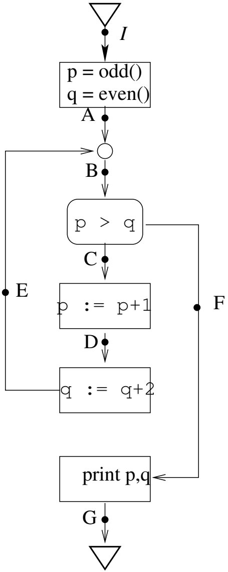


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)

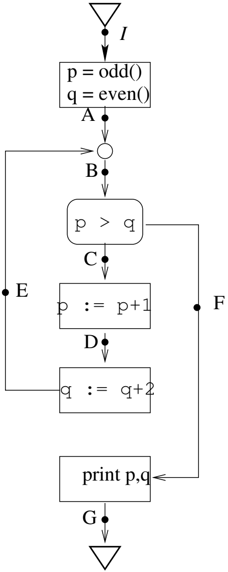


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)

Collecting abstract values – example

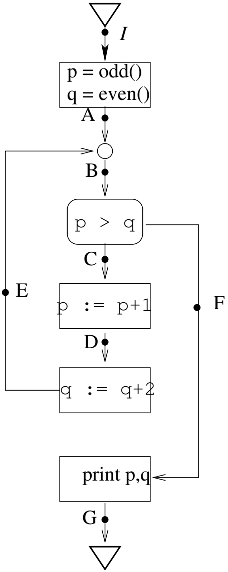


Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)



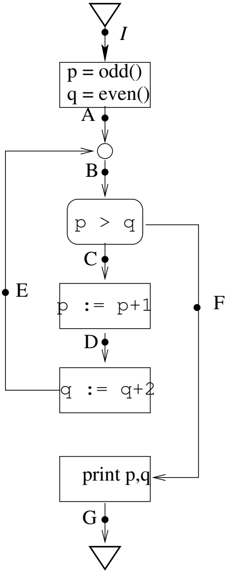
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)

Collecting abstract values – example



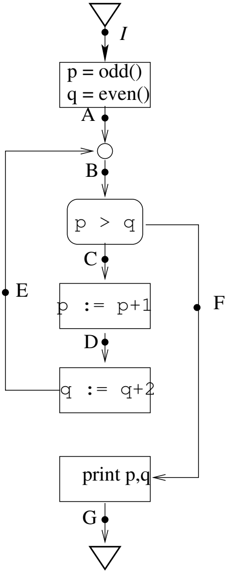
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)

Collecting abstract values – example



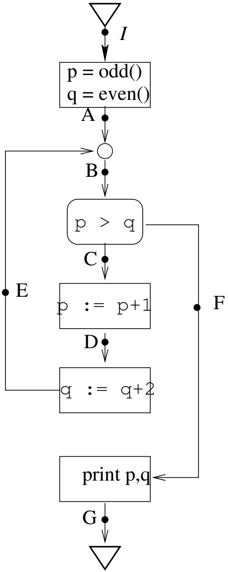
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)

Collecting abstract values – example



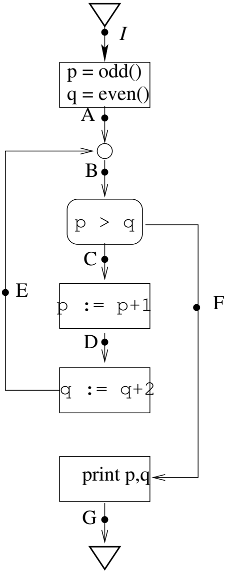
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)

Collecting abstract values – example



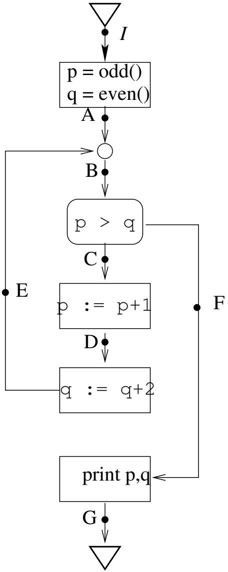
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)

Collecting abstract values – example



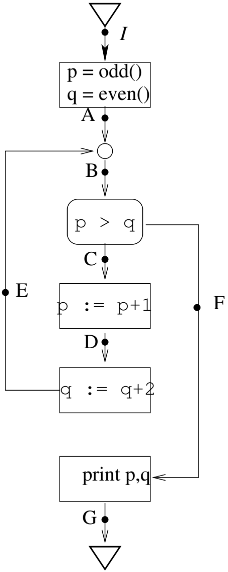
Path	Abstract value
I	(oe, oe) (given)
IA	(o, e)
IAB	(o, e)
IABC	(o, e)
IABCD	(e, e)
IABCDE	(e, e)
IABCDEB	(e, e)
IABCDEBC	(e, e)
IABCDEBCD	(o, e)
IABCDEBCDE	(o, e)
IABF	(o, e)

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)

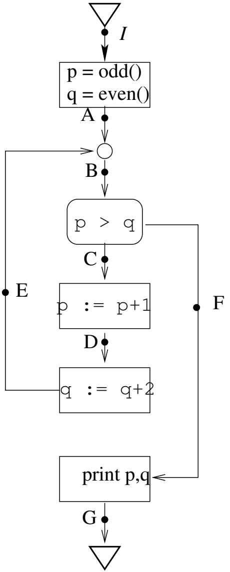
Collecting abstract values – example



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)



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.

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:

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	$\{(i,j) \mid i \text{ odd}, j \text{ even}, i < j\} \cup \{(i,j) \mid i \text{ even}, j \text{ even}, i = j\}$

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,

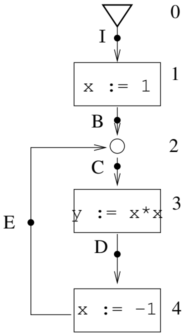
γ image of join of all abstract states arising at p (i.e., abstract JOP solution at p), with d_0 as the initial abstract value at P's entry

$$\supseteq$$

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



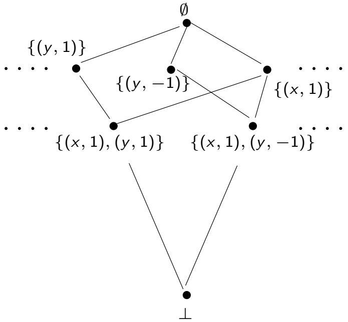
Path	Characterization of concrete states
I	*true* (given)
IB	x = 1
IBC	x = 1
IBCD	x = 1 $\wedge$ y = 1
IBCDE	x = -1 $\wedge$ y = 1
IBCDEC	x = -1 $\wedge$ y = 1
IBCDECD	x = -1 $\wedge$ y = 1
...	x = -1 $\wedge$ y = 1

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

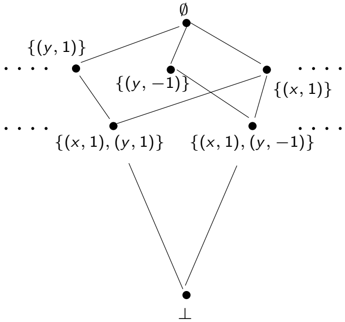
Abstract interpretation for constant propagation

- Abstract lattice D



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

- Abstract lattice D



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

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

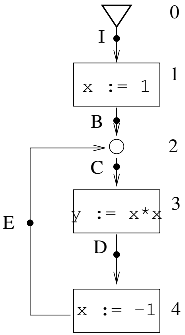
Abstract interpretation for constant propagation – contd.

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

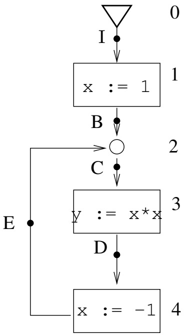
$$
\begin{array}{rcl}
f_n(P) & = & \bot, \text{ if } P \text{ is } \bot \\
& = & \{(y,c) \in P \mid y \neq x\} \cup \{(x,d)\}, \\
& & \quad \text{if all variables in } exp \text{ have constant values in } P\text{, and if} \\
& & \quad \text{exp evaluates to } d \text{ with these constant values} \\
&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

JOP using abstract lattice

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

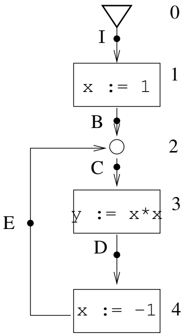


JOP using abstract lattice



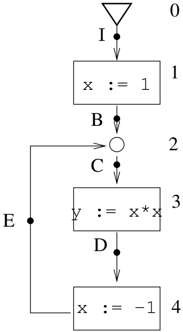
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$

JOP using abstract lattice



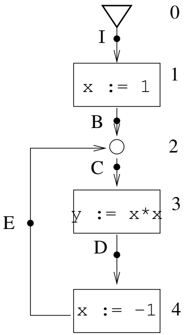
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$

JOP using abstract lattice



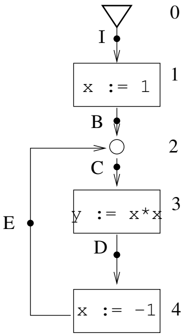
Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x,1)\}$
IBC	$\{(x,1)\}$
IBCD	$\{(x,1),(y,1)\}$

JOP using abstract lattice

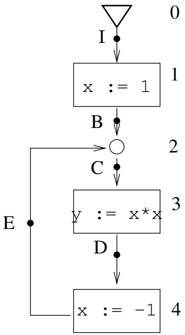


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x,1)\}$
IBC	$\{(x,1)\}$
IBCD	$\{(x,1),(y,1)\}$
IBCDE	$\{(x,-1),(y,1)\}$

JOP using abstract lattice

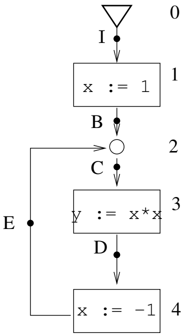


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$

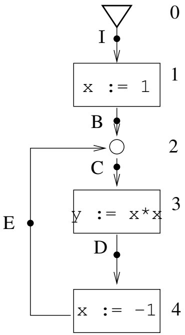


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$

JOP using abstract lattice

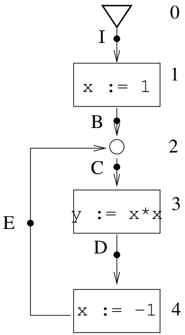


Path	Abstract value at end of path
I	$\emptyset$
IB	$\{(x, 1)\}$
IBC	$\{(x, 1)\}$
IBCD	$\{(x, 1), (y, 1)\}$
IBCDE	$\{(x, -1), (y, 1)\}$
IBCDEC	$\{(x, -1), (y, 1)\}$
IBCDECD	$\{(x, -1), (y, 1)\}$
...	$\{(x, -1), (y, 1)\}$



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

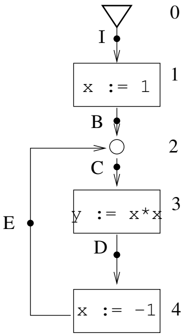
Point	Abstract JOP value
I	$\emptyset$



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	$\{(x,1)\}$

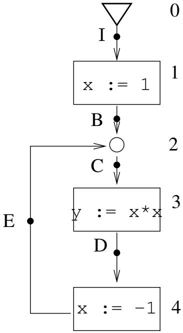
JOP using abstract lattice



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

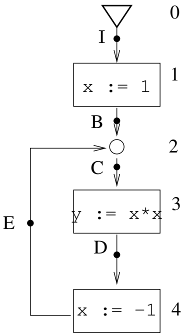
Point	Abstract JOP value
I	$\emptyset$
B	$\{(x, 1)\}$
C	$\emptyset$

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	$\{(y,1)\}$



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

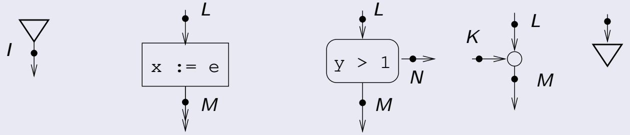
Verify that

- 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:

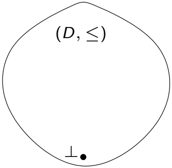
$$e ::= c \mid x \mid e + e \mid e - e \mid e * e.$$

- Boolean expressions:

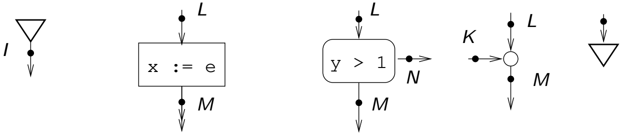
$$be ::= tt \mid ff \mid e \leq e \mid e = e \mid \neg be \mid be \vee be \mid be \wedge be.$$

- Assume unique initial program point I.

- Complete join semi-lattice $(D, \leq)$, with a least element $\perp$.
- Concretization function $\gamma : D \rightarrow 2^{State}$
- $\perp \in D$ represents unreachability of the program point (i.e., $\gamma(\perp)$ 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 $f(\perp) = \perp$.
- 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:

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

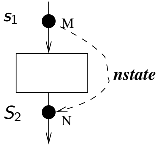
  where $(f_a \circ f_b)(x)$ is defined as $f_a(f_b(x))$.
- f_p is $\lambda d.\bot \;\Rightarrow\;$ path p is *infeasible*.
- Join over all paths (JOP) definition: For each program point N

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

  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.