Data-flow Analysis / Abstract Interpretation

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

16 January 2009

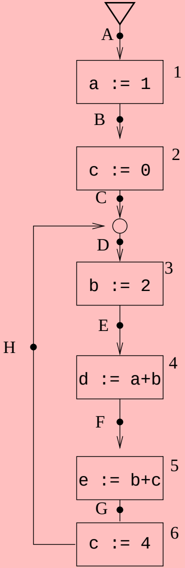
What is data-flow analysis

- "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 propogation: Is the value of a variable constant at a particular program location.
  - Replace `x := y + z` by `x := 17` during compilation.
- More recent interest by verification community (starting with Cousot-Cousot 1977).
- Ideas used in SLAM tool to verify properties ("lock-unlock protocol is respected") of device driver code.

Constant Propogation Example

A variable x has constant value c at a program point N if along every execution the value of x at N is c.

Example: At program point G, constants are $R_G = \{(a,1),(b,2),(d,3)\}$.

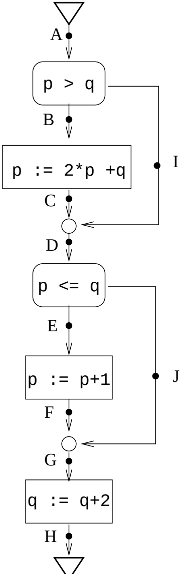


Overview of data-flow analysis

- Informal intro and motivation
- Lattices
- Data-flow analysis more formally
- Kildall's algo for computing over-approximation of JOP.
- Knaster-Tarski Fixpoint Theorem
- Correctness of Kildall's algo (computes the least solution to equations).

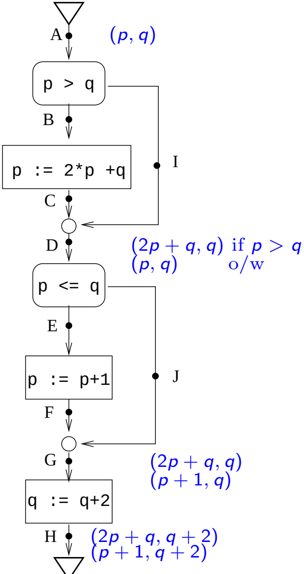
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.



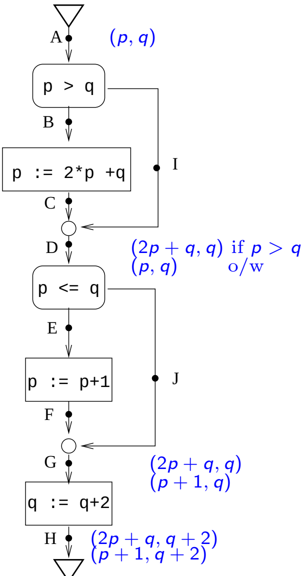
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.



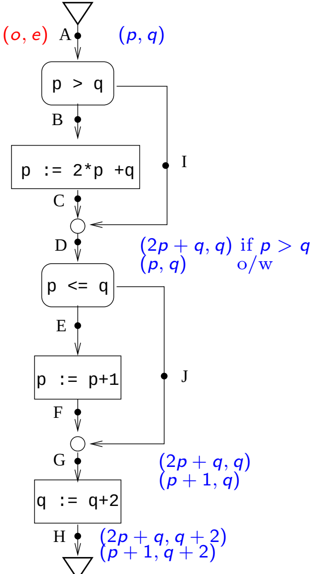
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



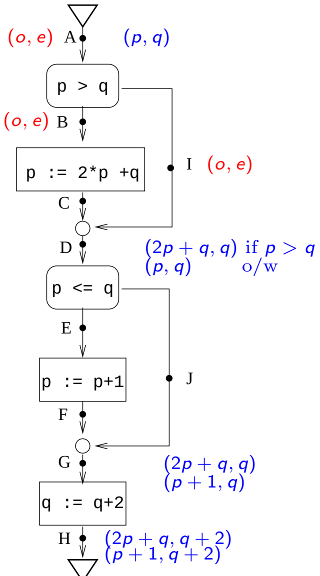
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto$ $\{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



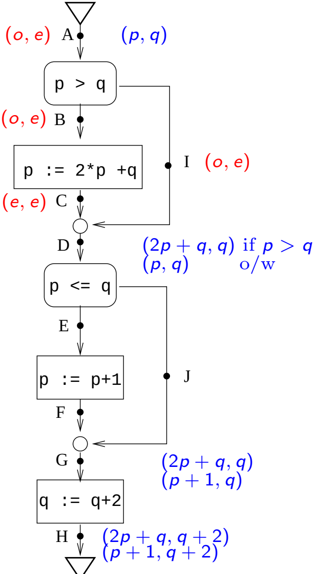
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto$ $\{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



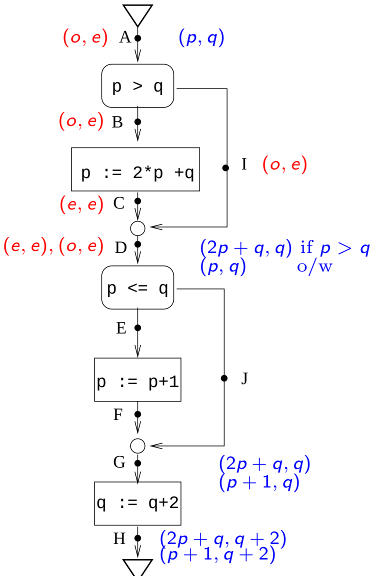
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



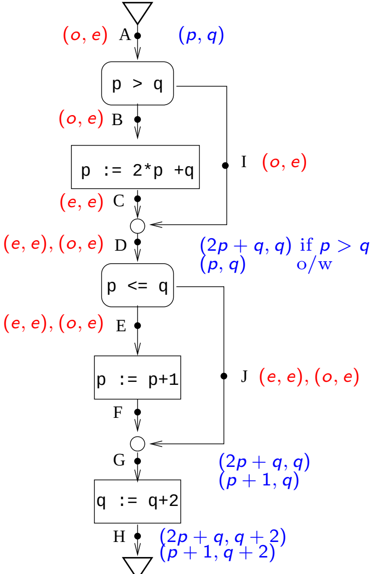
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



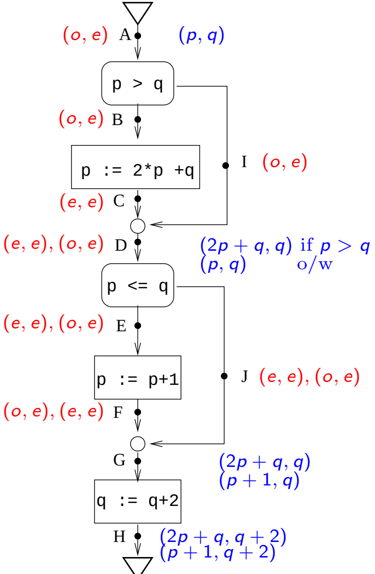
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto$
  $\{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



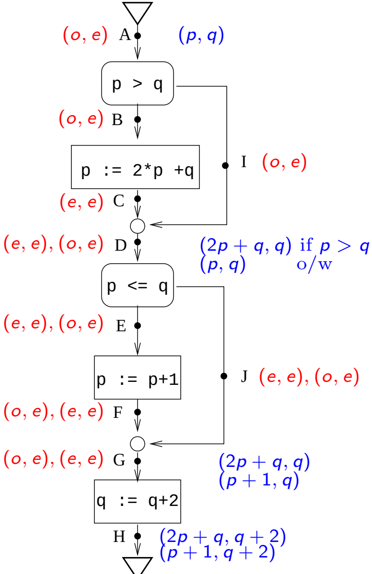
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



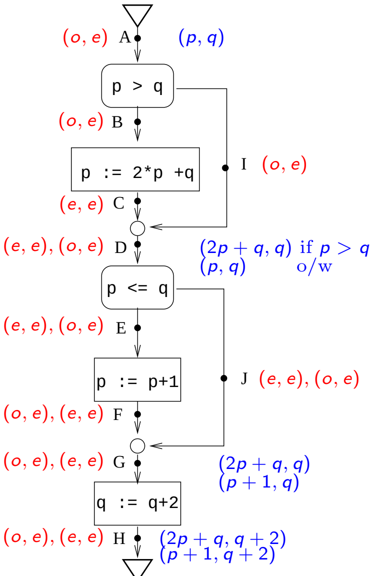
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.



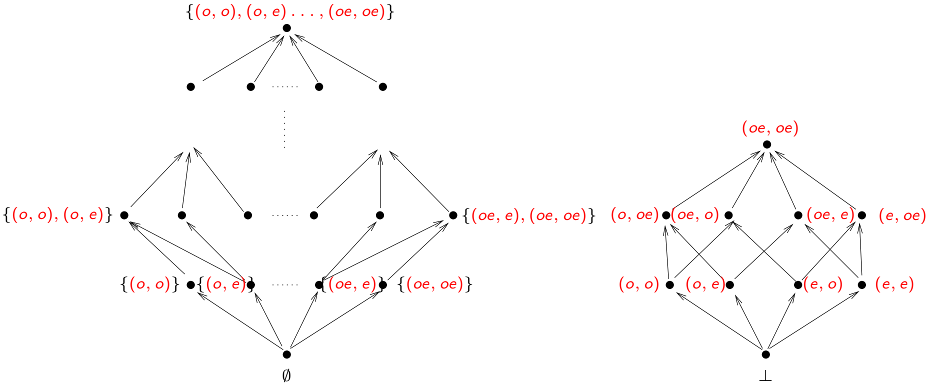
Data-flow analysis as approximation of collecting semantics

- Collecting semantics of a program: For each program point N, the set of states the program could be in at point N.
- Example: Parity-based abstract interpretation.
- Abstract values: o, e, oe
- States represented:
  $o \mapsto \{1, 3, 5, \ldots\}$,
  $e \mapsto \{0, 2, 4, \ldots\}$,
  $oe \mapsto \{0, 1, 2, 3, \ldots\}$.
- So $(o, e) \mapsto \{1, 3, 5, \ldots\} \times \{0, 2, 4, \ldots\}$.
- Abstract states at H represents $\mathbb{N} \times 2\mathbb{N}$, which is a safe approx



Why abstract data should have a "lattice" structure

- A natural subset lattice structure:



- ... and a more "efficient" but less-precise lattice.
- Ordering is "is more precise than".
- Take "join" or "least upper bound" of abstract states at a point.

Why transfer functions should be "monotonic"

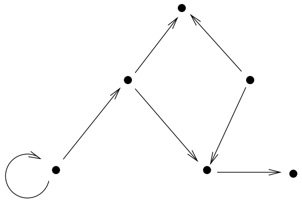
- More precise source state should lead to more precise target state.

Partial Orders

- A partially ordered set is a non-empty set D along with a partial order $\leq$.
  - $\leq$ 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 $d = d'$).

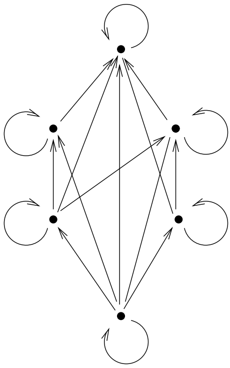
Binary relations as Graphs

We can view a binary relation on a set as a directed graph.

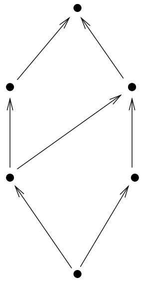


Partial Order as a graph

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



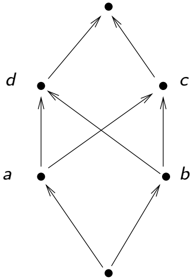
Graph representation



Hasse-diagram representation

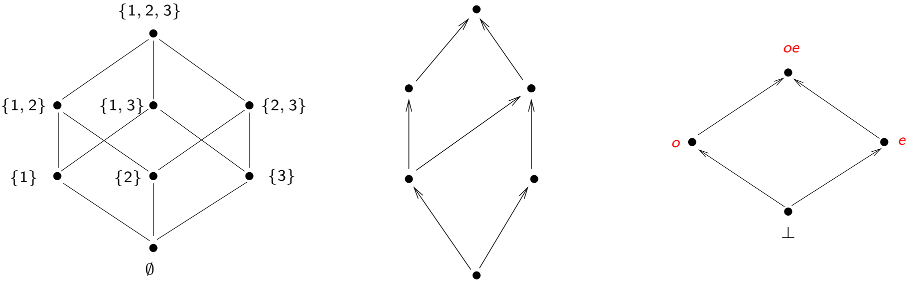
Upper bounds etc.

- 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

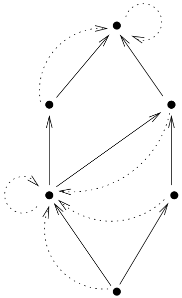
- A lattice is a partially order set in which every pair of elements has an lub and a glb.
- A complete lattice is a lattice in which every subset of elements has a lub and glb.



Question: Example of lattice which is not complete?

Monotonic functions

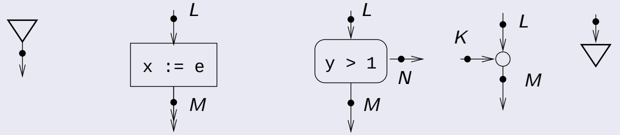
- A function $f : D \to D$ is monotonic or order-preserving if whenever $x \leq y$ we have $f(x) \leq f(y)$.



Data-flow / abstract-interpretation framework

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

Nodes or statements in a program



- Expressions:

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

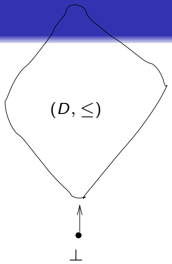
- Boolean expressions:

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

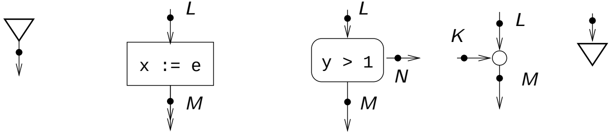
- Assume unique initial node I.

Data-flow framework contd.

- Complete lattice $L = (D, \leq)$.
- Add new bottom element to get $L_\perp = (D_\perp, \leq_\perp)$.



- Transfer function $f_{LM} : D_\perp \rightarrow D_\perp$ for each node and incoming edge L and outgoing edge M.



- We assume transfer functions are monotonic, and satisfy $f(\perp) = \perp$.
- 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:

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

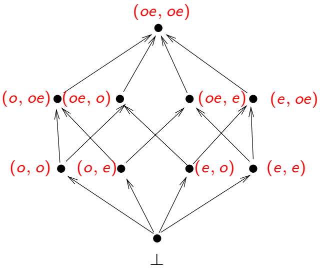
- For "infeasible" paths p, f_p will be $\lambda d.\bot$.
- Join over all paths (JOP) definition: For each program point N

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

where d_0 is a given initial value at entry node.

Example framework: parity interpretation

- Underlying lattice



- Transfer functions: for `x := e` node:

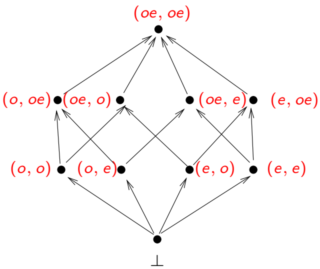
$$f_{MN}(s) = \begin{cases} s[x \mapsto o] & \text{if } [e]_s = o \\ s[x \mapsto e] & \text{if } [e]_s = e \\ s[x \mapsto oe] & \text{if } [e]_s = oe \end{cases}$$

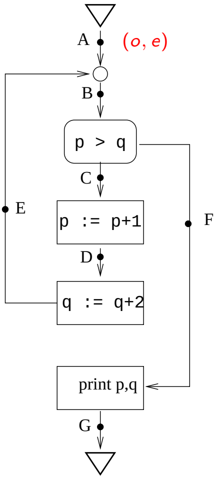
Kildall's algorithm to compute over-approximation of JOP

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

Kildall's algo on parity interpretation example

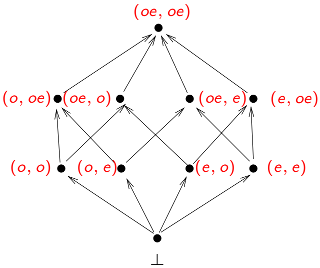
Underlying lattice

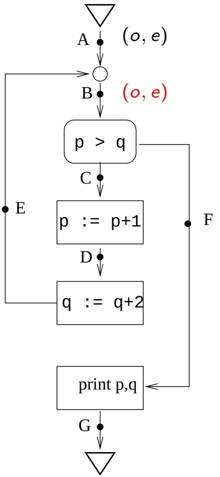




Kildall's algo on parity interpretation example

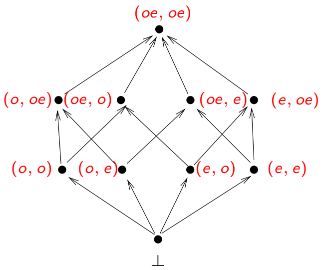
Underlying lattice

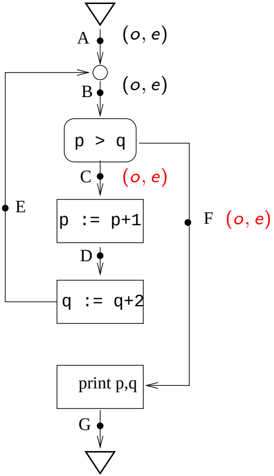




Kildall's algo on parity interpretation example

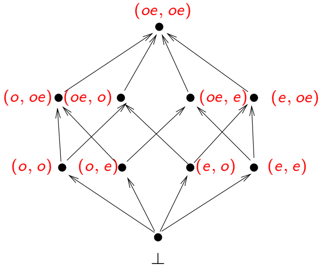
Underlying lattice

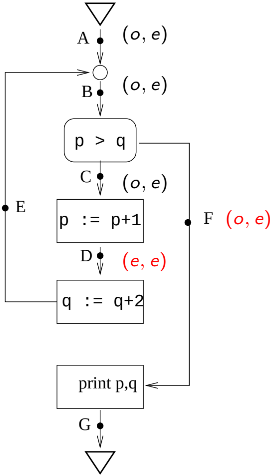




Kildall's algo on parity interpretation example

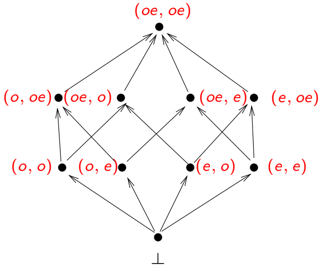
Underlying lattice

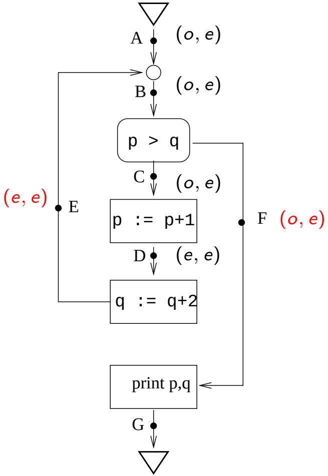




Kildall's algo on parity interpretation example

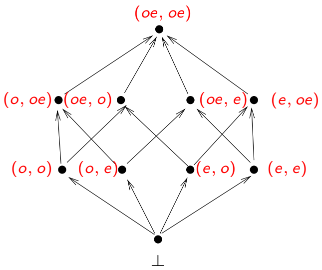
Underlying lattice

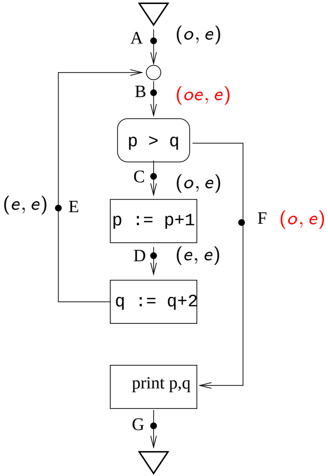




Kildall's algo on parity interpretation example

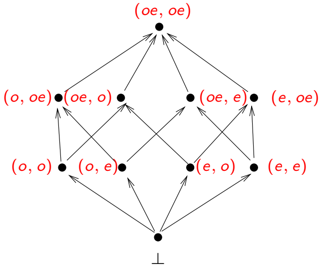
Underlying lattice

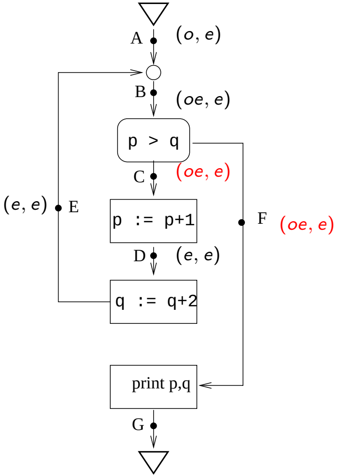




Kildall's algo on parity interpretation example

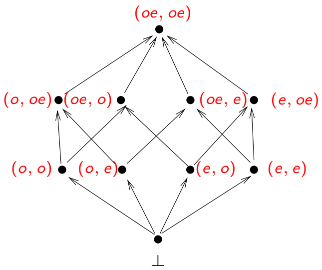
Underlying lattice

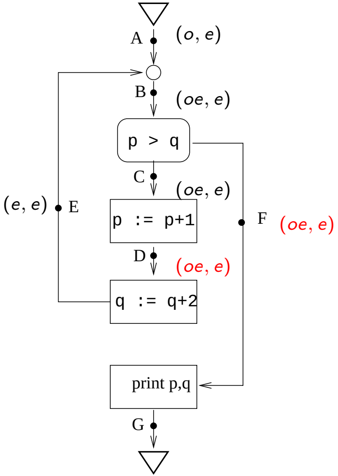




Kildall's algo on parity interpretation example

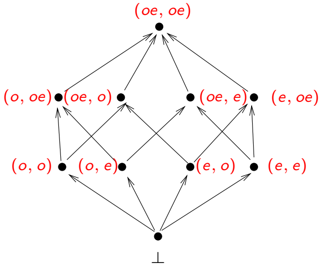
Underlying lattice

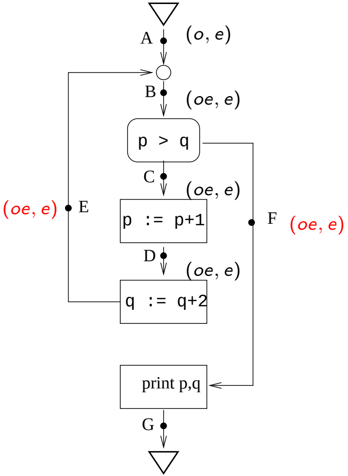




Kildall's algo on parity interpretation example

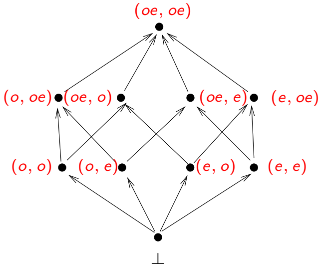
Underlying lattice

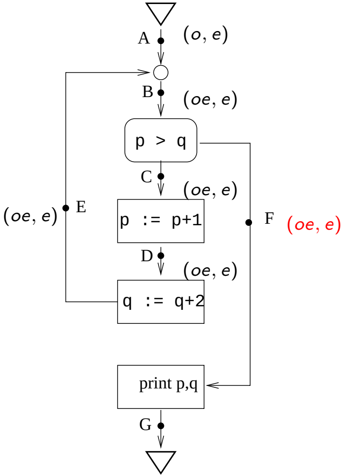




Kildall's algo on parity interpretation example

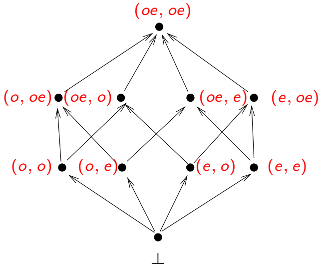
Underlying lattice

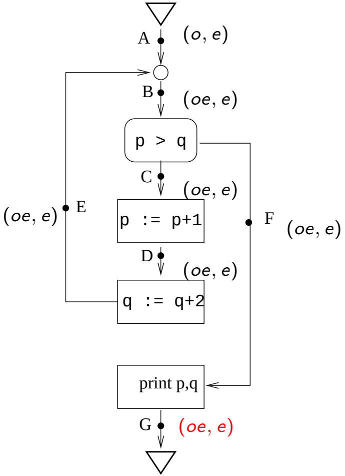




Kildall's algo on parity interpretation example

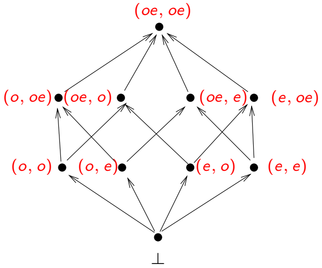
Underlying lattice

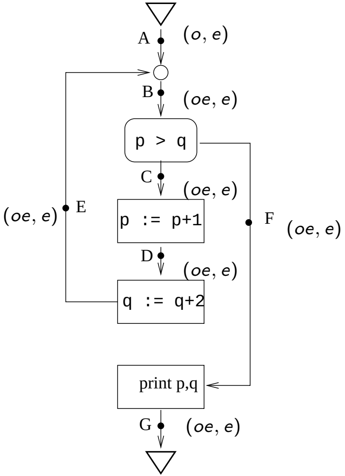




Kildall's algo on parity interpretation example

Underlying lattice

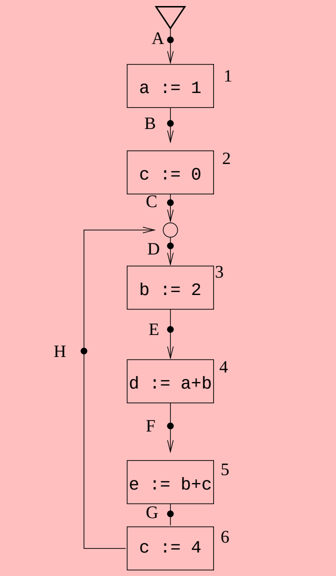




Another example analysis: Constant Propogation

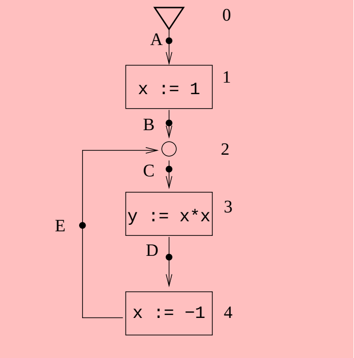
A variable x has constant value c at a program point N if along every execution the value of x at N is c.

Example: At program point G, constants are $R_G = \{(a,1),(b,2),(d,3)\}$.



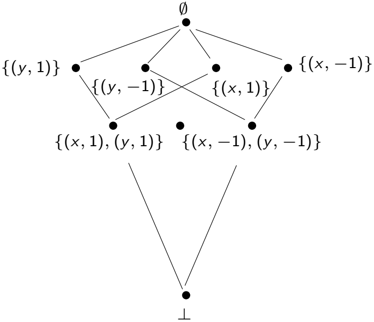
Another example program

ProgPt	Actual constant data
A	$\emptyset$
B	$(x, 1)$
C	$\emptyset$
D	$(y, 1)$
E	$(x, -1), (y, 1)$



Framework instance for CP

- Underlying lattice

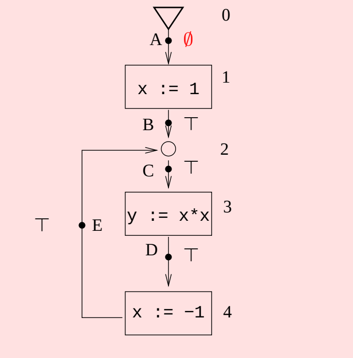


- Transfer function for assignment node n of the form $x := exp$.

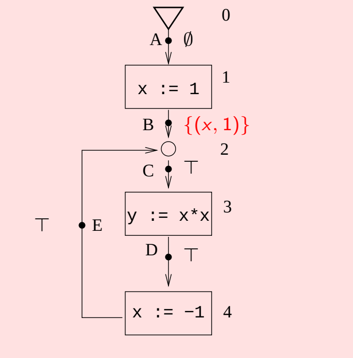
$$f_n(P) = \{(y, c) \mid y \neq x\} \cup \begin{cases} \{(x, d)\} & \text{if } [exp]_P = d \\ \emptyset & \text{otherwise.} \end{cases}$$

- Initial value at entry node: $\emptyset$.
- Transfer functions monotonic?

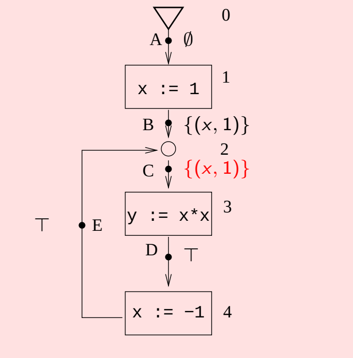
Kildall's algo on CP example: 1



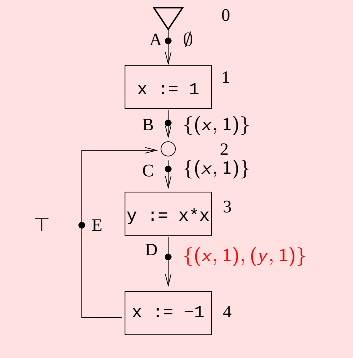
Kildall's algo on CP example: 2



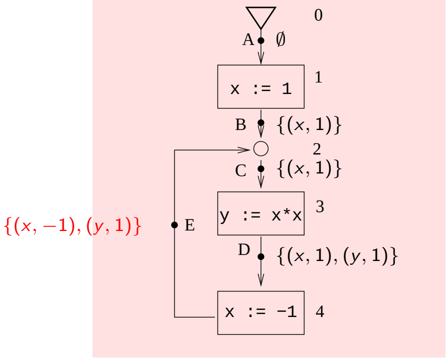
Kildall's algo on CP example: 3



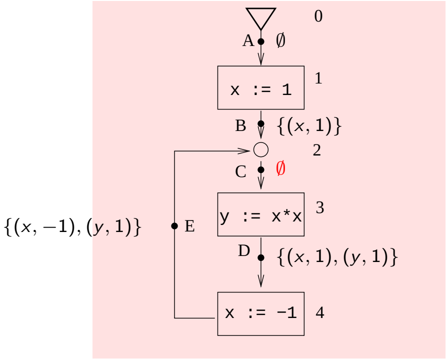
Kildall's algo on CP example: 4



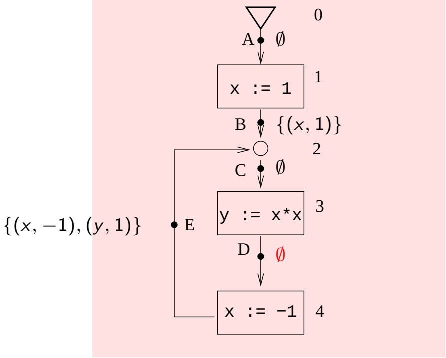
Kildall's algo on CP example: 5



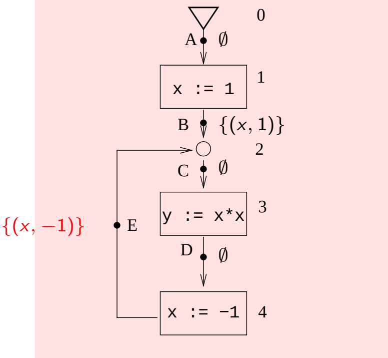
Kildall's algo on CP example: 6



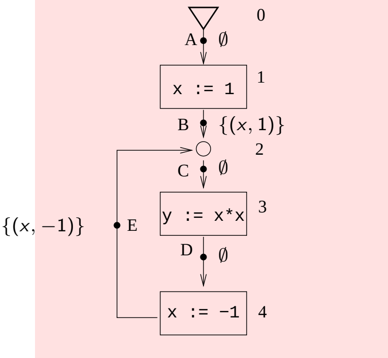
Kildall's algo on CP example: 7



Kildall's algo on CP example: 8

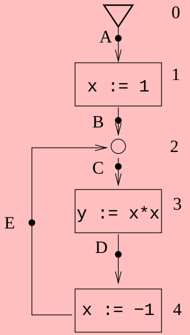


Kildall's algo on CP example: 9



Kildall's algo vs Actual Constant data

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

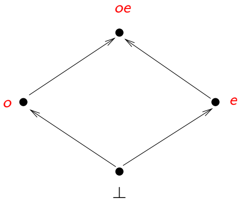


What Kildall's algo computes

- In general, computes an over-approximation of JOP.
- Always terminates if lattice has no infinite ascending chains.

More on lattices

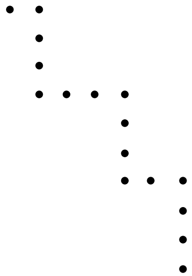
- A Chain in a partial order $(D, \leq)$ is a totally ordered subset of D.
- Ascending chain: $d_0 < d_1 < d_2 < \ldots$.
- Let $L = (D, \leq)$ be a complete lattice.
- The product lattice $\overline{L} = (D \times D, \leq')$ where $(d_1, d_2) \leq' (d_1', d_2')$ iff $d_1 \leq d_1'$ and $d_2 \leq d_2'$ is also a complete lattice.
- Exercise: compute product of parity lattice below with itself.



- Maximum ascending chain in $\overline{L} = L \times L$ is bounded by twice max ascending chain in L (*if* there is a max ascending chain in L).

Termination of Kildall's algo

- Let $\overline{d}_i$ be the vector of values after the i-th step of algo.
- Then after each step i, either number of marks decreases by 1 and $\overline{d}_{i+1} = \overline{d}_i$, or number of marks increase by 0 or 1 and $\overline{d}_{i+1} > \overline{d}_i$.
- Thus each $\overline{d}_i$ increases ($\geq$), and if it doesn't strictly increase we lose a mark.
- Thus maximum number of steps in algo is bounded by length of longest increasing chain in $\overline{L}$ * number of program points.



Viewing correctness

Extend f_n's to $\overline{f}$ over $\overline{D} = D \times \cdots \times D$ given by

$$\overline{f}(d_1, \ldots, d_k) = (\cdots, f_m(D_j), \cdots).$$

Then:

- $\overline{L} = (\overline{D}, \leq')$ is also a complete lattice.
- $\overline{f}$ is montonic on $\overline{L}$ if each f_n is.
- Set up equations Eq relating the data values at each program point.
- Least solution to Eq is same as LFP of functional $\overline{f}$ on lattice $\overline{L}$.
- If each f_n is distributive, then JOP $=$ LFP$(\overline{f})$.
- Otherwise, if f_n is only monotonic, JOP $\leq$ LFP$(\overline{f})$.
- Kildall's algo computes least solution to Eq, for monotone frameworks.
- Note this is a stronger claim than "Kildall's algo computes JOP for distributive frameworks".

Induced Equations

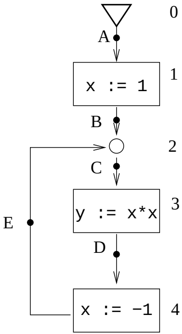
Framework induces natural data-flow equations:

x_E	$=$	e	for an entry node E
x_N	$=$	$f_n(x_M)$	for an assignment node n 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.

Equations for CP example

Equations induced by CP analysis:

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

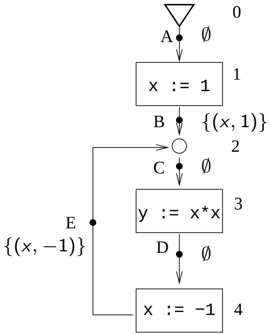


Equations for CP example

Equations induced by CP analysis:

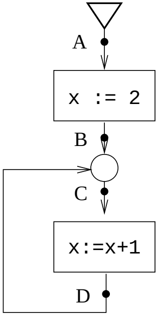
$$
\begin{aligned}
x_A &= \emptyset \\
x_B &= f_1(x_A) \\
x_C &= x_B \sqcup x_E \\
x_D &= f_3(x_C) \\
x_E &= f_4(x_D).
\end{aligned}
$$

Values computed by Kildall are a solution to these equations.



Exercise: Give 2 solutions to equations induced for this program

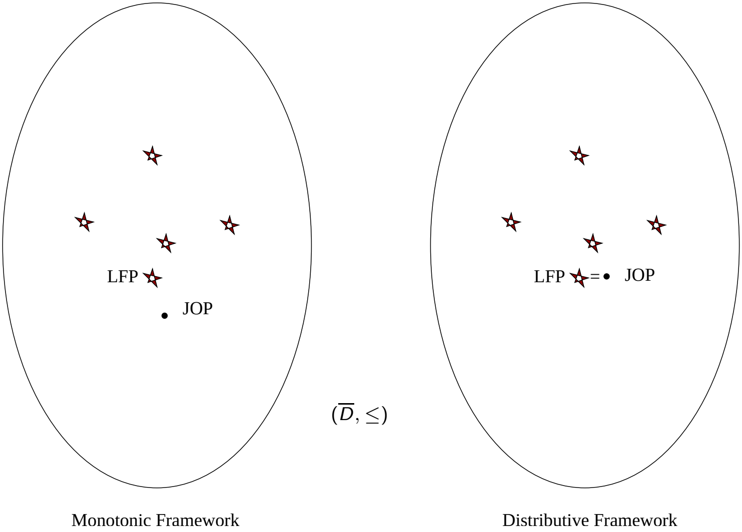
- Use collecting semantics with concrete stores in $\{x\} \to \mathbb{Z}$.
- Write down induced equations.
- Give two different solutions to the equations.



Natural ordering on solutions

- Consider "vectorised" lattice $\overline{D} = (D^k, \leq')$ (similar to product lattice $L \times L$).
- Each solution is a point in this vectorised lattice
- We will see that these solutions form a complete lattice, with least and greatest element.
- This is the least solution we mean.
- In fact a solution is a "fixpoint" of a natural function $\overline{f}$ induced by transfer functions for each node.

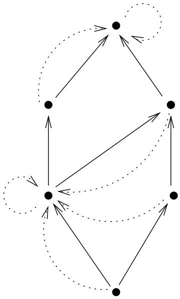
Correctness



Kildall's algo always computes LFP.

Knaster-Tarski fixpoint theorem for 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.
- A function $f : D \rightarrow D$ is monotonic or order-preserving if whenever $x \leq y$ we have $f(x) \leq f(y)$.
- A fixpoint of a function $f : D \rightarrow D$ is an element $x \in D$ such that $f(x) = x$.
- A pre-fixpoint of f is an element x such that $x \leq f(x)$.



Knaster-Tarski Fixpoint Theorem

Theorem (Knaster-Tarski)

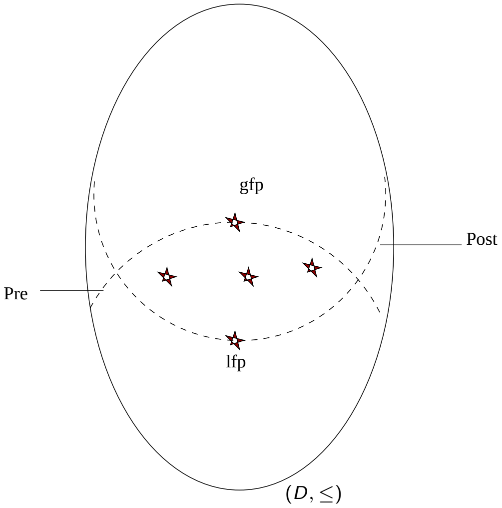
Let $(D, \leq)$ be a complete lattice, and $f : D \rightarrow D$ a monotonic function on $(D, \leq)$. Then:

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

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

(c) *The least fixpoint of f coincides with the glb of the set of postfixpoints of f, and the greatest fixpoint of f coincides with the lub of the prefixpoints of f.*

Fixpoints of f



Proof of Knaster-Tarski theorem

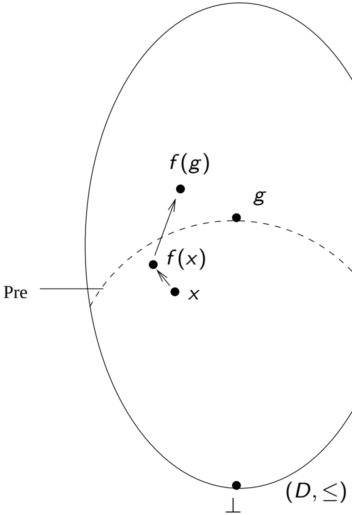
(a) $g = \bigsqcup \mathit{Pre}$ is a fixpoint of f.

(b) g is the greatest fixpoint of f.

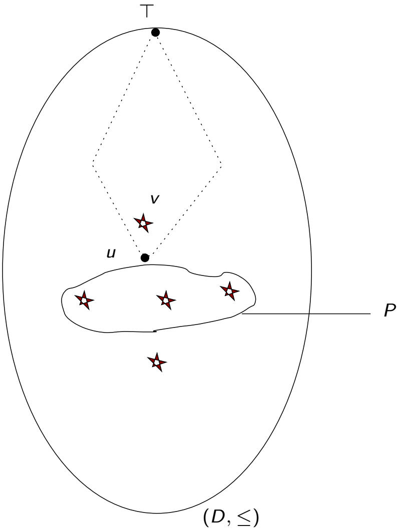
(c) Similarly $l = \bigsqcap \mathit{Post}$ is the least fixpoint of f.

(d) Let P be the set of fixpoints of f. Then $(P, \leq)$ is a *complete* lattice.

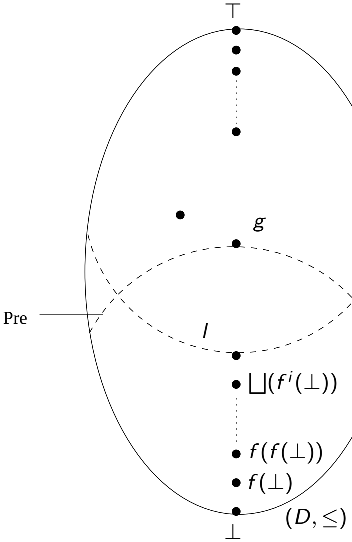
Proof of K-T theorem: (a)



Proof of K-T theorem: (d)



Computing lfp's and gfp's



Computing lfp's and gfp's

- "Ascending Chain Condition": No infinite ascending chains, or
- Continuity:
  - $X \subseteq D$ is *directed* if every finite subset of X has an upper bound in X.
  - f on $(D, \leq)$ is *continuous* if for every directed subset X of D we have $f(\bigsqcup X) = \bigsqcup(f(X))$.

Then

$$lfp(f) = \bigsqcup(f^n(\bot)).$$

A more general condition

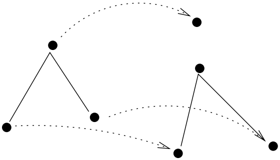
- A complete partial order (cpo) is a partial order in which every ascending chain has an lub.
- A pointed cpo is one which has a least element $\bot$.
- Let $(D, \leq)$ be a cpo. A function $f : D \to D$ is continuous if for any ascending chain X in D, $f(\bigsqcup X) = \bigsqcup(f(X))$.

Fact

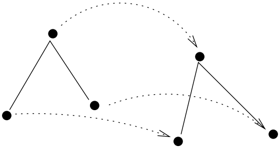
If f is a continuous function on a pointed cpo $(D, \leq)$ then f has a least fixpoint and

$$lfp(f) = \bigsqcup_{i \geq 0}(f^i(\bot)).$$

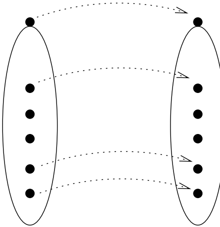
Monotonicity, distributivity, and continuity



Monotonic



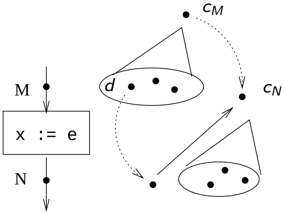
Distributive



Continuous

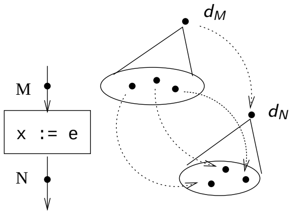
Back to Kildall: JOP $\leq$ LFP for monotone framework

- We show JOP $\leq \overline{c}$, for any FP $\overline{c}$.
- JOP $= \bigsqcup_{i \geq 0} \mathrm{JOP}_i$, where $\mathrm{JOP}_i = \bigsqcup_{\mathrm{paths}\ p, |p| \leq i} f_p(e)$.
- Claim: $\mathrm{JOP}_i \leq \overline{c}$ for any fixpoint $\overline{c}$.
    - By induction on i: Base case immediate.
    - Assume $\mathrm{JOP}_i \leq \overline{c}$, and consider JOP_{i+1}



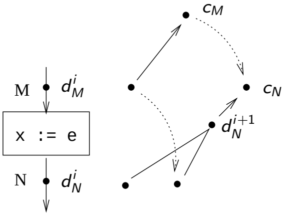
Correctness: JOP = LFP for finite distributive framework

- JOP = LFP for distributed framework, finite lattice.
- Enough to show that JOP is a fixpoint of $\overline{f}$.



What Kildall's algo computes (ctd)

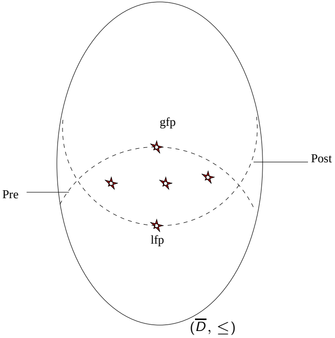
- Values at each step are bounded above by any fixed point $\overline{c}$.



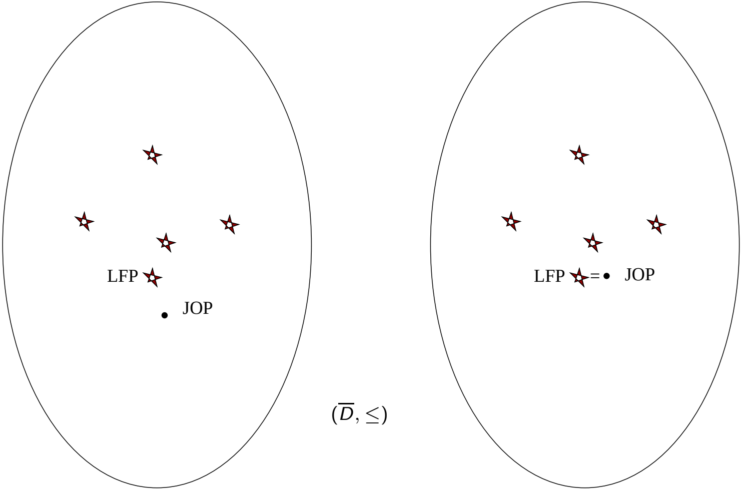
- Thus it follows that $\overline{d} \leq \overline{l}$ where $\overline{l}$ is LFP of $\overline{f}$.

What Kildall's algo computes (ctd)

- Sufficient now to show that $\overline{d} \geq \overline{l}$.
  - Suffices to show that $\overline{d}$ is such that $\overline{d} \geq \overline{f}(\overline{d})$ (i.e. $\overline{d}$ is a postfixpoint of $\overline{f}$)
    - We observe that if a value d_M^i was unmarked at some step in the algo, its value would have been propogated.
    - Thus, in particular, $d_N \geq f_{MN}(d_M)$, since d_M would have been propogated.
  - By Knaster-Tarski theorem, $\overline{l} = glb(Post)$, and hence $\overline{d} \geq \overline{l}$.



Correctness



Monotonic Framework

Distributive Framework

Kildall's algo always computes LFP.

Back to Constant Propogation

- f_n^{CP} is monotonic
- f_n^{CP} is *not* distributive.
  - Consider node n with statement $y := x * x$, and abstract values $d_1 = \{(x, 1)\}$ and $d_2 = \{(x, -1)\}$.
  - $f_n(d_1 \sqcup d_2) = \top$
  - $f_n(d_1) \sqcup f_n(d_2) = \{(y, 1)\}$.

Why computing JOP for CP is undecidable

- Post Correspondence Problem (PCP): Given strings $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$, is there a string $w = u_{i_1} u_{i_2} \cdots u_{i_l}$ such that $w = v_{i_1} v_{i_2} \cdots v_{i_l}$, with $i_1 = 1$.
- Consider program for which computing JOP for Constant Propogation implies solution to PCP.

```
while (*) {
  if(*) {
    x := x * u_1;
    y := y * v_1;
  }
  ......
  if(*) {
    x := x * u_n;
    y := y * v_n;
  }
}
if (x == y) z := 1 else z := -1;
```