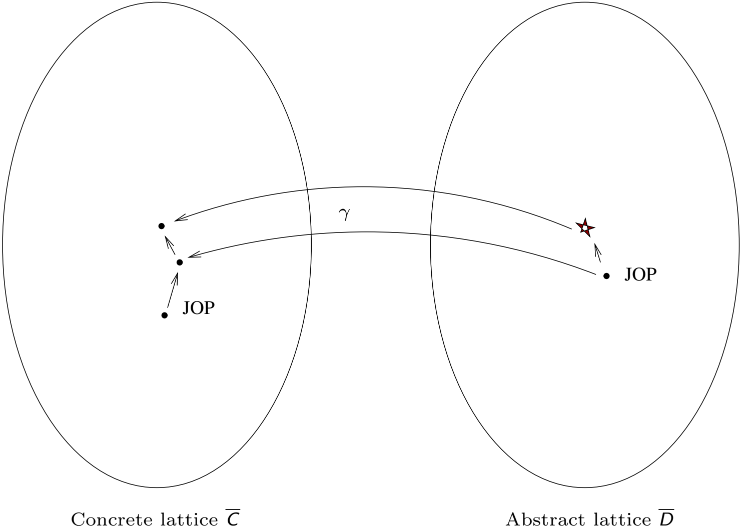
Kildall's algorithm for over-approximate JOP

Deepak D'Souza and K.V. Raghavan

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

Why over-approximation of JOP in abstract lattice is useful



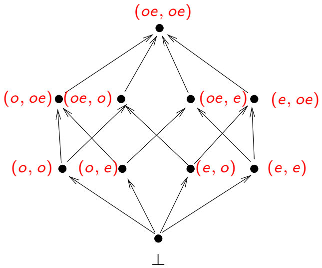
Kildall's algorithm to compute over-approximation of JOP

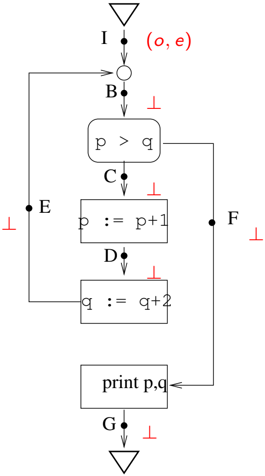
Input: An instance (P, d_0) of a monotone data-flow framework $((D, \leq), F)$.
Output: For each program point N in P, a data-value d_N such that $\mathrm{JOP}_N^{d_0} \leq d_N$.

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

Kildall's algo on parity interpretation example

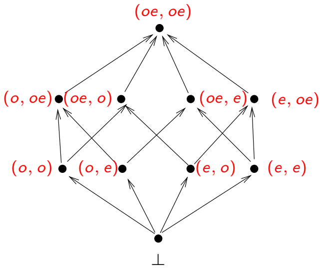
Underlying lattice

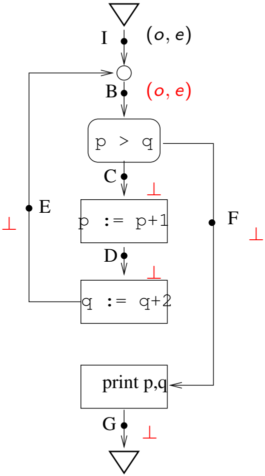




Kildall's algo on parity interpretation example

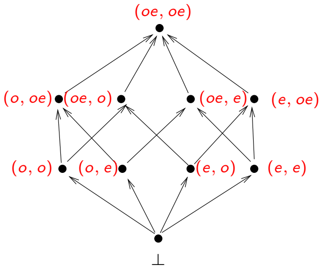
Underlying lattice

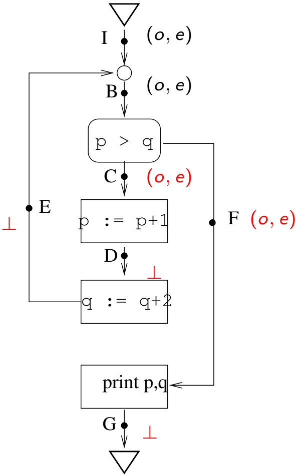




Kildall's algo on parity interpretation example

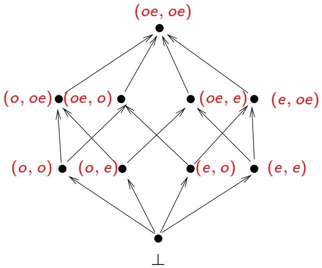
Underlying lattice

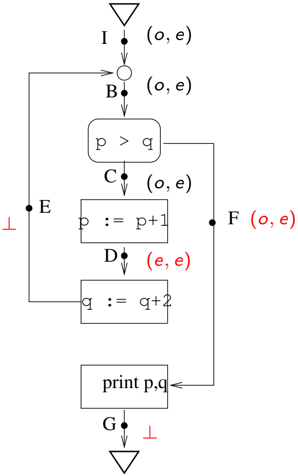




Kildall's algo on parity interpretation example

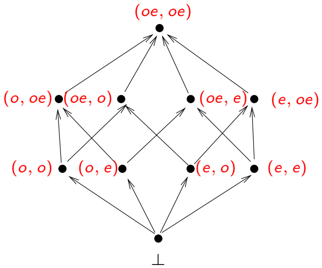
Underlying lattice

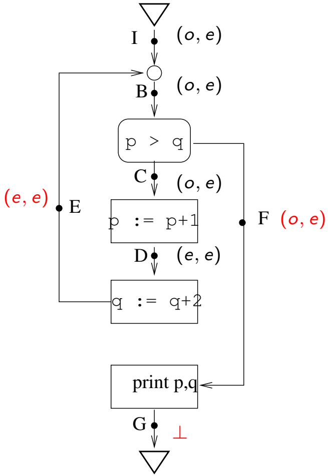




Kildall's algo on parity interpretation example

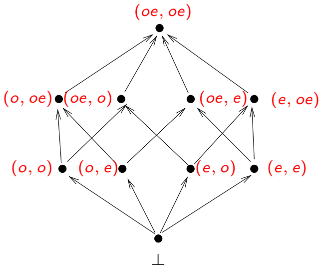
Underlying lattice

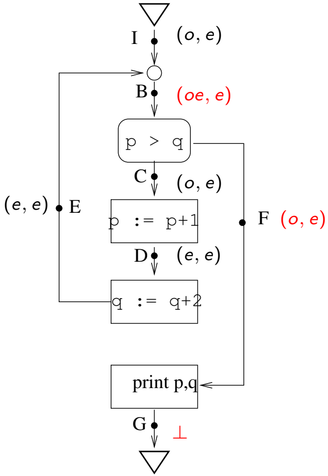




Kildall's algo on parity interpretation example

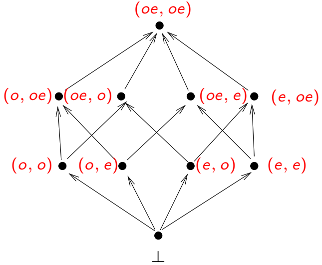
Underlying lattice

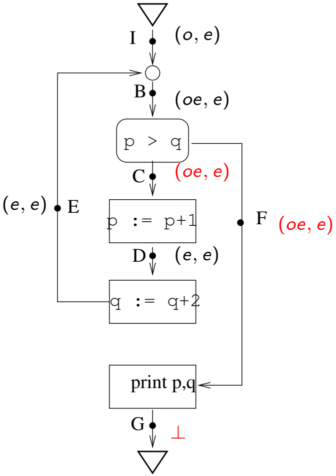




Kildall's algo on parity interpretation example

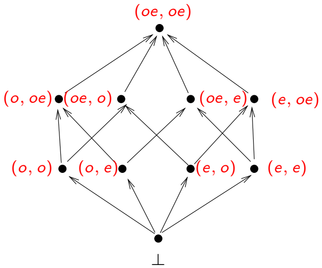
Underlying lattice

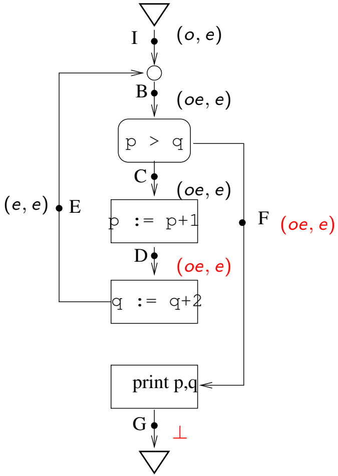




Kildall's algo on parity interpretation example

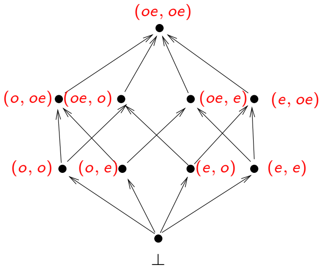
Underlying lattice

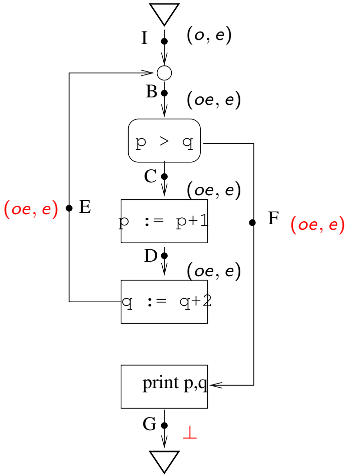




Kildall's algo on parity interpretation example

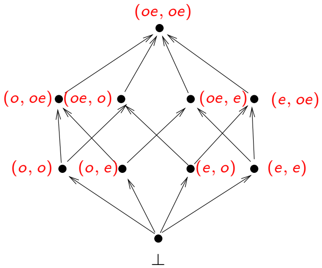
Underlying lattice

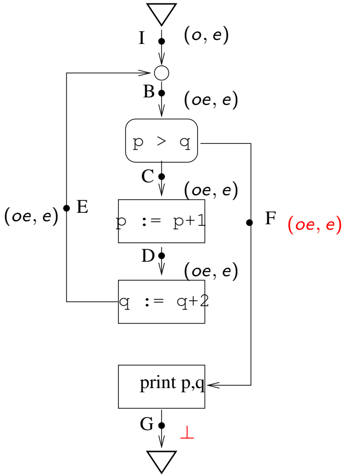




Kildall's algo on parity interpretation example

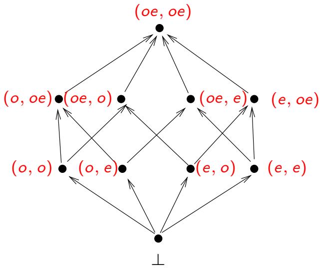
Underlying lattice

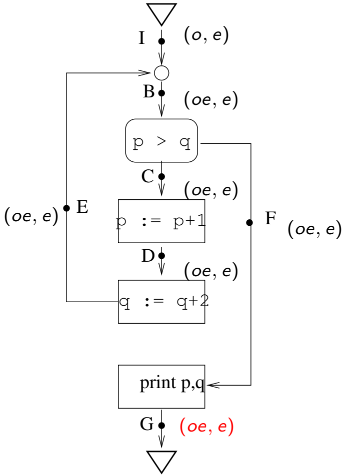




Kildall's algo on parity interpretation example

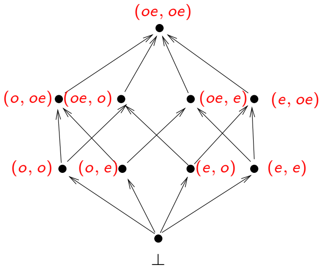
Underlying lattice

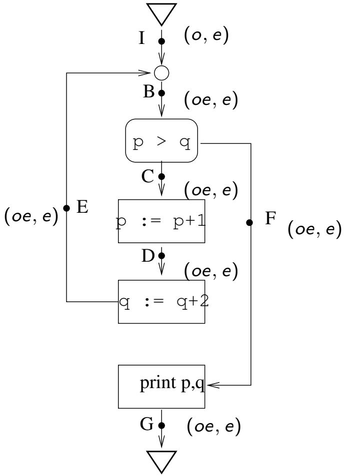




Kildall's algo on parity interpretation example

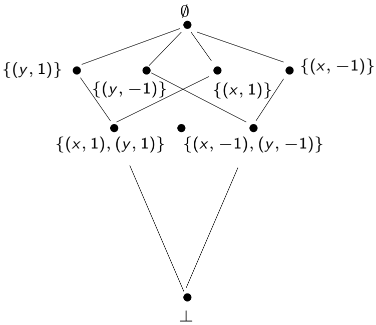
Underlying lattice

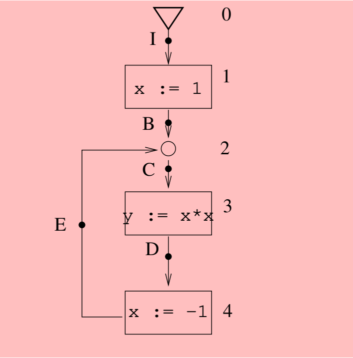




Values computed coincide with JOP values.

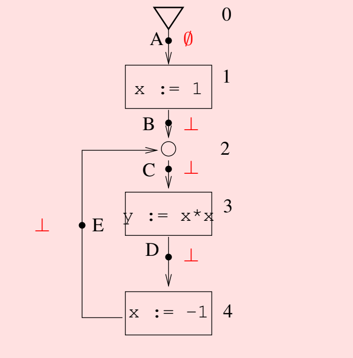
Constant propagation example



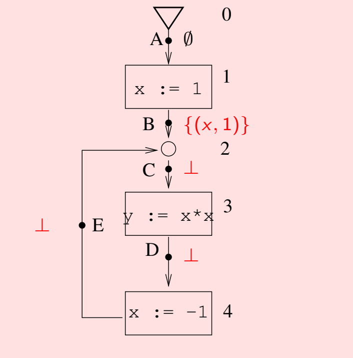


ProgPt	JOP values
A	$\emptyset$
B	$\{(x,1)\}$
C	$\emptyset$
D	$\{(y,1)\}$
E	$\{(x,-1),(y,1)\}$

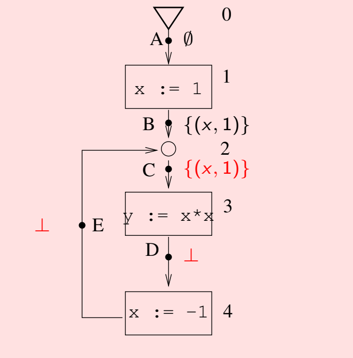
Kildall's algo on CP example: 1



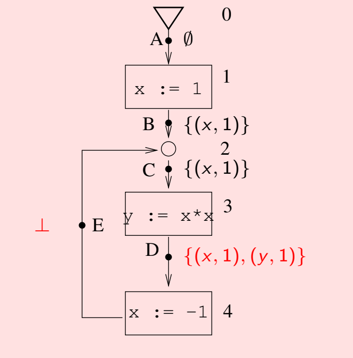
Kildall's algo on CP example: 2



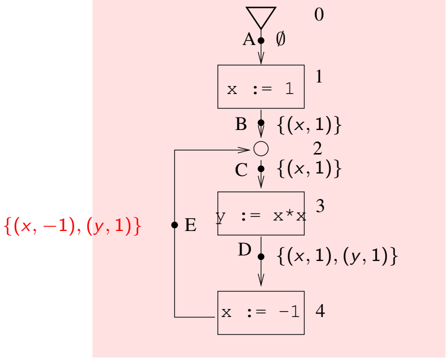
Kildall's algo on CP example: 3



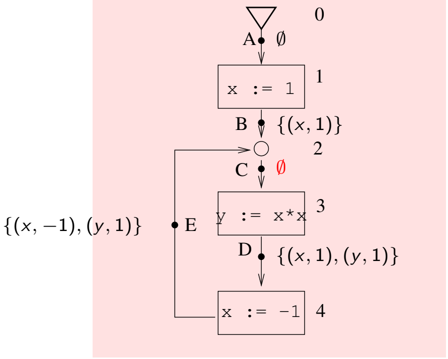
Kildall's algo on CP example: 4



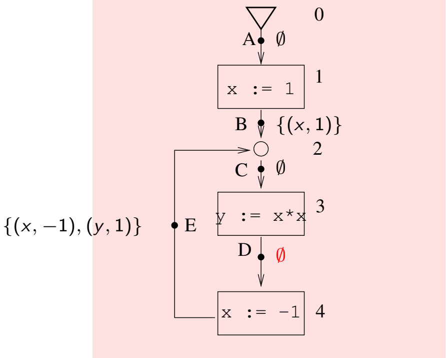
Kildall's algo on CP example: 5



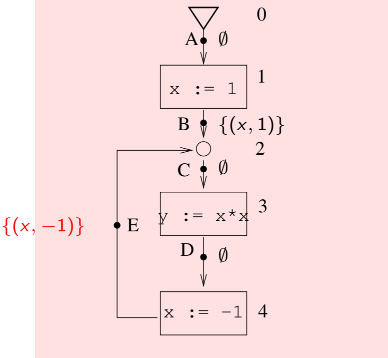
Kildall's algo on CP example: 6



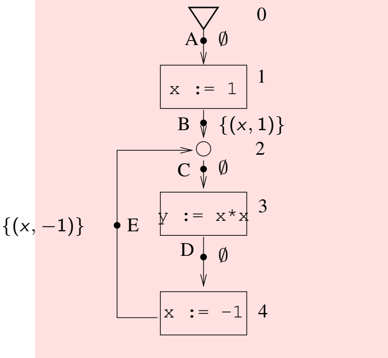
Kildall's algo on CP example: 7



Kildall's algo on CP example: 8

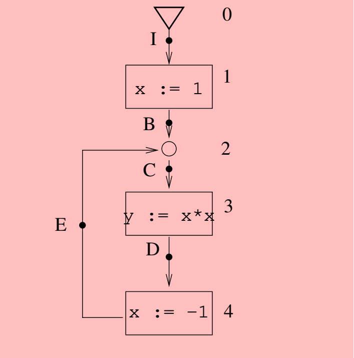


Kildall's algo on CP example: 9



Kildall's algo vs Actual Constant data

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



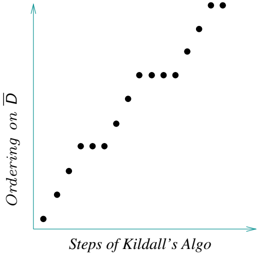
Note that Kildall's values are $\geq$ the actual JOP values at all points.

What Kildall's algo computes

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

Termination of Kildall's algo

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



Induced Equations

The program induces a set of data-flow equations:

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

Induced Equations

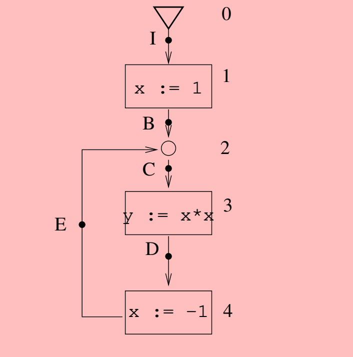
The program induces a set of data-flow equations:

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

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

Example equations

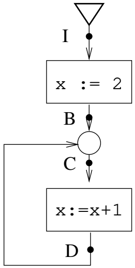
$$
\begin{aligned}
x_I &= \emptyset\ (= d_0) \\
x_B &= f_1(x_I) \\
x_C &= x_B \sqcup x_E \\
x_D &= f_3(x_C) \\
x_E &= f_4(x_D).
\end{aligned}
$$



Equations can have multiple solutions

Exercise: Give two solutions to equations induced for this program

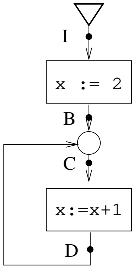
- Use lattice of subsets of concrete stores, with integer values for `x`.
- Write down induced equations.
- Give two different solutions to the equations.



Equations can have multiple solutions

Exercise: Give two solutions to equations induced for this program

- Use lattice of subsets of concrete stores, with integer values for `x`.
- Write down induced equations.
- Give two different solutions to the equations.

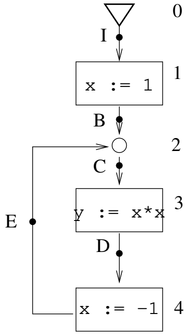


Note: collecting semantics of any program is the least solution to its data-flow equations using the concrete lattice (to be shown).

Function $\overline{f}$ induced by equations

Equations:

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



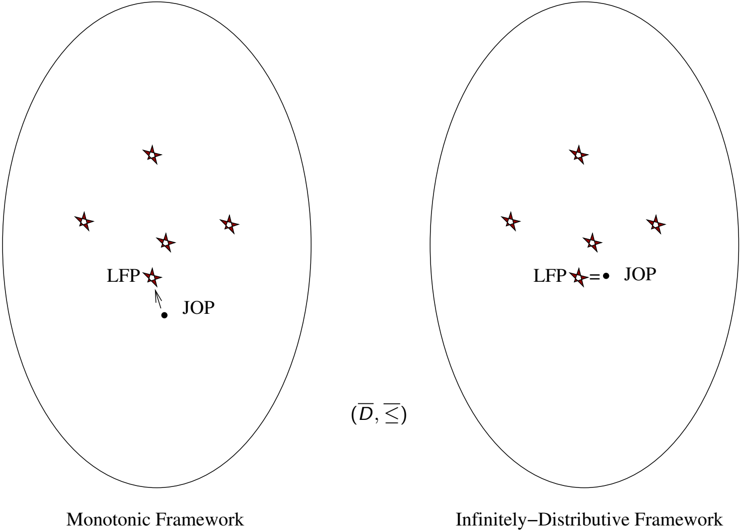
Corresponding $\overline{f}$ function:

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

Natural ordering on solutions to Eq

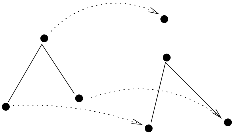
- Consider "vectorised" lattice $\overline{D} = (D^k, \overline{\leq})$, where D is the underlying lattice.
- Each solution to the equations is a point in this vectorised lattice.
- The solutions are precisely the fix-points of the function $\overline{f}: \overline{D} \rightarrow \overline{D}$.
- If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

Correctness

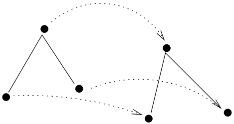


Kildall's algo always computes LFP of $\overline{f}$.

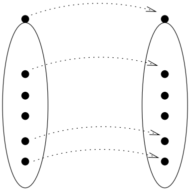
Monotonicity, distributivity, and continuity



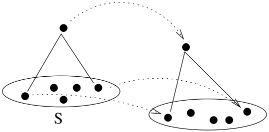
Monotonic



Distributive



Continuous



Inf−Distributive
(S is any subset of the lattice,
(including empty subset, or an infinite subset)

1. JOP $\leq$ LFP for monotone framework

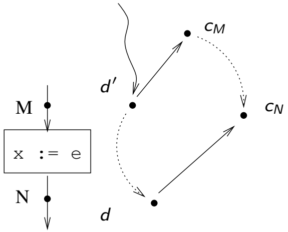
- Let $\overline{c}$ be any FP of $\overline{f}$. Consider any program point N. Let $c_N \equiv \overline{c}[N]$.
- **Claim**: For any path p, if N is the point at the end of p, c_N dominates $d \equiv f_p(d_0)$ reaching N.
  The argument is by induction on length of path p.
  - Base case $|p| = 0$: Then $N = I$, and $d = c_N = d_0$.
  - Let path p be of length $i + 1$. Let M be the program that p passes through just before reaching N. Let d' be $f_p^M(d_0)$, where f_p^M is the path transfer function of the prefix of path p that ends at point M. The inductive hypothesis is that $d' \sqsubseteq c_M$.
    The rest of the proof is in two cases.

1. JOP $\leq$ LFP for monotone framework

Case (node between M and N is not a join node):

By definition of $\overline{f}$, $(\overline{f}(\overline{c}))[N] = f_{MN}(c_M)$. Now, since $\overline{c}$ is an FP of $\overline{f}$, $c_N = (\overline{f}(\overline{c}))[N]$. Therefore, $c_N = f_{MN}(c_M)$.

Now, since $d = f_{MN}(d')$, by monotinicity of f_{MN}, and from the hypothesis $d' \sqsubseteq c_M$, it follows that $d \sqsubseteq c_N$.



1. JOP $\leq$ LFP for monotone framework

Case (node between M and N is a join node):
Let P be the other predecessor of the join node.

1. $d = d'$ (because join nodes have identity transfer function)
2. $c_M \sqsubseteq c_N$. The argument for this is as follows. By definition of $\overline{f}$, $(\overline{f}(\overline{c}))[N] = c_M \sqcup c_P$. Now, since $\overline{c}$ is an FP of $\overline{f}$, $c_N = (\overline{f}(\overline{c}))[N]$. Therefore, $c_N = c_M \sqcup c_P$.

The two observations above in conjunction with the inductive hypothesis imply that $d \sqsubseteq c_N$.

1. JOP $\leq$ LFP for monotone framework

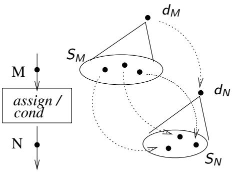
- That is, for every path p that reaches a point N, $f_p(d_0) \sqsubseteq c_N$.
- Therefore, JOP d_N at N is $\sqsubseteq c_N$

2. JOP = LFP for infinitely-distributive framework

Proof: Enough to show that the JOP $\overline{d}$ is a fixpoint of $\overline{f}$. We denote $\overline{d}[M]$ as d_M, $\overline{d}[N]$ as d_N, etc.

2. JOP = LFP for infinitely-distributive framework

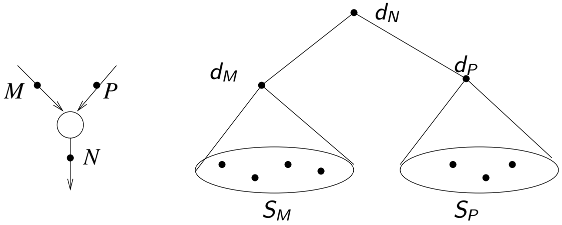
Proof: Enough to show that the JOP $\overline{d}$ is a fixpoint of $\overline{f}$. We denote $\overline{d}[M]$ as d_M, $\overline{d}[N]$ as d_N, etc.
Let N be any program point.
Case (the node before N is not a join node):



- Let S_M (resp. S_N) be the set of all facts that reach M (resp. N) along all paths.
- It is clear that $S_N = \{f_{MN}(s) | s \in S_M\}$.
- It is clear that the JOP d_M at M is equal to $\sqcup S_M$, and the JOP d_N at N is equal to $\sqcup S_N$.
- Therefore, by the previous two observations, and due to infinite distributivity, it follows that $d_N = f_{MN}(d_M)$.
- Therefore, $\overline{d}$ satisfies N's equation, which is $x_N = f_{MN}(x_M)$.

2. JOP = LFP for infinitely-distributive framework

Case (the node before N is a join node):



- Say S_M (resp. S_P resp. S_N) is the set of lattice values reaching M along all paths (resp. reaching P resp. reaching N).
- Clearly, d_M (resp. d_P resp. d_N) is equal to $\sqcup S_M$ (resp. $\sqcup S_P$ resp. $\sqcup S_N$).
- It is clear that $S_N = S_M \cup S_P$. Therefore, $d_N = d_M \sqcup d_P$.
- Therefore, $\overline{d}$ satisfies N's equation, which is $x_N = x_M \sqcup x_P$.

2. JOP = LFP for infinitely-distributive framework

- Since the argument in the previous two slides applies at all points N, we have shown that the vector $\overline{d}$ satisfies all the equations, and is hence a fix-point of $\overline{f}$.
- Note: Lattice is finite, and functions are pairwise distributive, and $f_i(\bot) = \bot$ implies framework is infinitely distributive.

Back to Constant Propagation

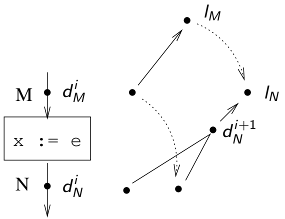
- f_n^{CP} is monotonic
- f_n^{CP} is *not* distributive.
  - Consider node n with statement $y := x * x$, and abstract states $P_1 = \{(x, 1)\}$ and $P_2 = \{(x, -1)\}$.
  - Since $P_1 \sqcup P_2$ is $\top$, $f_n(P_1 \sqcup P_2) = \top$
  - On the other hand, $f_n(P_1) \sqcup f_n(P_2) = \{(y, 1)\}$.
- The $nstate'$ functions are all distributive.

3. Kildall's algo computes LFP

- Let $\overline{d}$ be values computed by Kildall's algo upon termination, and $\overline{l}$ be LFP of $\overline{f}$. Let l_N denote $\overline{l}[N]$, l_M denote $\overline{l}[M]$, etc.
- Intermediate vector $\overline{d}^i$ after any step i is bounded above by $\overline{l}$. We prove this using induction on number of steps.
- Let N by any program point whose value gets updated in Step $i + 1$.

3. Kildall's algo computes LFP

Case (the node before N is a non-join node):



Explanation:

- $d^i_M \sqsubseteq l_M$ and $d^i_N \sqsubseteq l_N$ by inductive hypothesis.
- $l_N = f_{MN}(l_M)$, because $\bar{l}$ is a solution to the equations and because we have the equation $x_N = f_{MN}(x_M)$.
- Therefore, due to monotonicity of f_{MN}, $f_{MN}(d^i_M) \sqsubseteq l_N$.
- Since $d^{i+1}_N = d^i_N \sqcup f_{MN}(d^i_M)$, we derive $d^{i+1}_N \sqsubseteq l_N$.

3. Kildall's algo computes LFP

Case (the node before N is a join node):

- Let M and P be the points that precede the join node. Let d^i_M, d^i_P, d^i_N be the data values at the respective program points after Step i.
- Say propagation happens from M to N in Step i (argument is similar if propagation happened from P to N).
- Since $\bar{l}$ is a solution to the equations, and since we have the equation $x_N = x_M \sqcup x_P$, it follows that $l_N = l_M \sqcup l_P$. In other words, $l_M \sqsubseteq l_N$. In conjunction with $d^i_M \sqsubseteq l_M$ (inductive hypothesis), we get $d^i_M \sqsubseteq l_N$.
- By inductive hypothesis, $d^i_N \sqsubseteq l_N$. Therefore, $(d^{i+1}_N = (d^i_M \sqcup d^i_N)) \sqsubseteq l_N$.

Thus it follows that $\overline{d} \leq \bar{l}$.

3. Kildall's algo computes LFP

Let $\overline{d}$ be the vector computed by the algorithm upon termination.
We now show that $\overline{d} \geq \overline{f}(\overline{d})$ (i.e. $\overline{d}$ is a postfixpoint of $\overline{f}$)
Let N be any program point.
Case (the node before N is a non-join node):

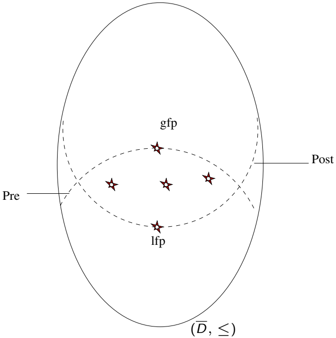
- Let M be the point that precedes this node. By definition of $\overline{f}$, $(\overline{f}(\overline{d}))[N]$ is equal to $f_{MN}(d_M)$.
- Since all points are unmarked, value d_M must have been propagated to N. That is, d_N must dominate $f_{MN}(d_M)$. That is, d_N dominates $(\overline{f}(\overline{d}))[N]$.

Case (the node before N is a join node):

- Let M and P be the points that precede the join node. By definition of $\overline{f}$, $(\overline{f}(\overline{d}))[N]$ is equal to $d_M \sqcup d_P$.
- Since all points are unmarked, value d_M and d_P must have been propagated to N. That is, d_N must dominate both d_M and d_P. That is, d_N dominates $d_M \sqcup d_P$. Hence, d_N dominates $(\overline{f}(\overline{d}))[N]$.

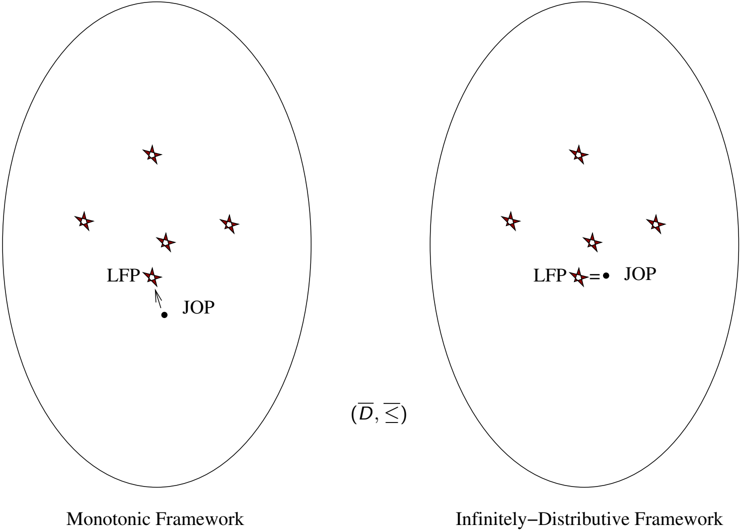
3. Kildall's algo computes LFP

- Therefore, by Knaster-Tarski theorem, $\bar{l} = glb(Post)$, and hence $\overline{d} \geq \bar{l}$.



- We have earlier proved that $\overline{d} \leq \bar{l}$. Therefore, it follows that $\overline{d} = \bar{l}$.

Correctness



Kildall's algo always computes LFP.

Overview of correctness

- Every program induces a set of equations on variables whose domain is lattice D. The equations, in turn, induce a function $\overline{f} : \overline{D} \to \overline{D}$.
- If each f_i is monotone and D is a complete lattice then $\overline{f}$ has a least fix-point $\text{LFP}(\overline{f})$.
  - If each f_i is infinitely distributive, then $\text{JOP} = \text{LFP}(\overline{f})$.
  - Otherwise, if each f_i is only monotonic, $\text{JOP} \leq \text{LFP}(\overline{f})$.

Overview of correctness

- Every program induces a set of equations on variables whose domain is lattice D. The equations, in turn, induce a function $\overline{f} : \overline{D} \rightarrow \overline{D}$.
- If each f_i is monotone and D is a complete lattice then $\overline{f}$ has a least fix-point LFP($\overline{f}$).
  - If each f_i is infinitely distributive, then JOP $=$ LFP($\overline{f}$).
  - Otherwise, if each f_i is only monotonic, JOP $\leq$ LFP($\overline{f}$).
- Kildall's algorithm, for monotone frameworks:
  - Solution *at any point* during its execution is $\leq$ LFP($\overline{f}$)
  - If and when it terminates, solution is equal to LFP($\overline{f}$)
  - Note this is a stronger claim than "Kildall's algo computes JOP for distributive frameworks" [Killdall, 'POPL 73].
  - Kildall's algorithm is not only for program analysis. It can be used to find least solution to *any* set of simultaneous equations, as long as (a) domain of variables' values is a complete lattice, (b) each variable occurs in the lhs of a unique equation, and (c) all operators occurring in rhs's are monotone.

Summary of sufficient conditions

	Termination	LFP $\geq$ JOP	LFP $=$ JOP	Kild computes LFP upon termination
f_{MN}'s monotonic	$\surd$	$\surd$		$\surd$
No inf. asc. chains	$\surd$			
Inf. distributive			$\surd$	

- Each column is a property, and each row is a sufficient condition
- For a property to hold, *each* sufficient condition mentioned in its column needs to hold