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.

September 15, 2017
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 \(\text{JOP}^{d_0}_{d_N} \leq d_N\).

- Initialize data value at each program point to \(\bot\), 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.
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

\[
\begin{align*}
& (o, e) \\
& (o, o) \\
& (e, o) \\
& (e, e) \\
& (oe, o) \\
& (oe, e) \\
& (oe, oe)
\end{align*}
\]

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

\((oe, oe)\)

\((o, oe)\) \(\rightarrow\) \((oe, o)\) \(\rightarrow\) \((oe, e)\) \(\rightarrow\) \((e, oe)\)

\((o, o)\) \(\rightarrow\) \((o, e)\) \(\rightarrow\) \((e, o)\) \(\rightarrow\) \((e, e)\)

\(\bot\)

\((o, e)\)

\(p > q\)

\(p := p+1\)

\(q := q+2\)

print \(p,q\)

\(\bot\)
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

\[(e, oe) \perp (oe, e) \perp (o, e) \perp (o, o) \perp (e, e)\]

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

\[(o_e, o_e)\]  \[
(o_e, o) \rightarrow (o, o_e) \rightarrow (o, o) \rightarrow (e, o) \rightarrow (e, e) \rightarrow \bot \]

\[(e, o_e) \rightarrow (e, o) \rightarrow (e, e) \rightarrow \bot \]

Values computed coincide with JOP values.
Correctness of Kildall

Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Kildall’s algo on parity interpretation example

Underlying lattice

Values computed coincide with JOP values.
Constant propagation example

ProgPt | JOP values
-------|---------
A      | ∅       
B      | {(x, 1)}
C      | ∅       
D      | {(y, 1)}
E      | {(x, -1), (y, 1)}
Kildall’s algo on CP example: 1

\[
A \rightarrow B \rightarrow C \rightarrow E
\]

1. \(x := 1\)
2. \(y := x \times x\)
3. \(x := -1\)
Kildall’s algo on CP example: 2

\[
x := 1 \\
y := x \times x \\
x := -1
\]
Kildall's algo on CP example: 3

\[
\begin{align*}
x & := 1 \\
y & := x \times x \\
x & := -1 \\
E & \downarrow \\
\downarrow & \\
0 & \\
\emptyset & \\
\{ (x, 1) \} & \\
\{ (x, 1) \} & \\
\{ (x, 1) \} & \\
\{ (x, 1) \} & \\
1 & \\
2 & \\
3 & \\
4 & \\
\end{align*}
\]
Kildall’s algo on CP example: 4

x := 1
y := x*x
x := −1

∅

{(x, 1)}

{(x, 1)}

y := x*x

{(x, 1), (y, 1)}

x := −1
Kildall’s algo on CP example: 5

\[
\begin{align*}
\text{A} & \quad \emptyset & 0 \\
\text{B} & \quad \{(x, 1)\} & 1 \\
\text{C} & \quad \{(x, 1)\} & 2 \\
\text{E} & \quad \{(x, -1), (y, 1)\} & 3 \\
\text{D} & \quad \{(x, 1), (y, 1)\} & 4 \\
\text{D} & \quad \{(x, 1), (y, 1)\} & 5 \\
\text{E} & \quad \{(x, -1), (y, 1)\} & 6
\end{align*}
\]
Kildall’s algo on CP example: 6

\[
\begin{align*}
A & \quad \emptyset \\
1 & \quad x := 1 \\
2 & \quad B \quad \{(x, 1)\} \\
3 & \quad C \quad \emptyset \\
3 & \quad \{(x, -1), (y, 1)\} \\
4 & \quad D \quad \{(x, 1), (y, 1)\} \\
4 & \quad \{(x, -1), (y, 1)\} \\
5 & \quad x := -1
\end{align*}
\]
Kildall’s algo on CP example: 7

{(x, -1), (y, 1)}

A \rightarrow \emptyset

x := 1

B \rightarrow \{(x, 1)\}

C \rightarrow \emptyset

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

y := x*x

D \rightarrow \emptyset

x := -1
Kildall’s algo on CP example: 8

A

B

D

x := 1

y := x*x

x := -1

C

2

3

4

∅

∅

∅

{(x, 1)}

{(x, -1)}

{(x, 1)}

{(x, -1)}

∅

∅

∅

{(x, 1)}

{(x, -1)}

={(x, 1)}

={(x, -1)}

E

x := -1
Kildall’s algo on CP example: 9

\[ x := 1 \]
\[ y := x \times x \]
\[ x := -1 \]
Kildall’s algo vs Actual Constant data

<table>
<thead>
<tr>
<th>ProgPt</th>
<th>Actual JOP values</th>
<th>Kildall’s data</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>B</td>
<td>{(x, 1)}</td>
<td>{(x, 1)}</td>
</tr>
<tr>
<td>C</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>D</td>
<td>{(y, 1)}</td>
<td>∅</td>
</tr>
<tr>
<td>E</td>
<td>{(x, −1), (y, 1)}</td>
<td>{(x, −1)}</td>
</tr>
</tbody>
</table>

Note that Kildall’s values are ≥ 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.
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 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 \( N \).

\[ \ldots \] etc.
The program induces a set of data-flow equations:

\[
x_I = d_0 \quad \text{for entry point } I
\]

\[
x_N = f_{MN}(x_M) \quad \text{for an assignment or conditional node } n \text{ with}
\]

\[
\text{with incoming point } M \text{ and outgoing point } N
\]

\[
x_N = x_L \sqcup x_M \quad \text{for a junction node with incoming points } L,M
\]

\[
\text{and outgoing } N.
\]

\[
\ldots
\]

\[
\text{etc.}
\]

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

\[ x_A = \emptyset \ (= d_0) \]
\[ x_B = f_1(x_A) \]
\[ x_C = x_B \sqcup x_E \]
\[ x_D = f_3(x_C) \]
\[ x_E = f_4(x_D). \]
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.

```
A
x := 2

B

C
x := x + 1

D
```
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.

\[
\begin{align*}
\text{A} & \quad \text{B} \\
& \downarrow \\
\text{x := 2} & \quad \text{C} \\
& \downarrow \\
\text{D} & \downarrow \\
& \text{x:=x+1}
\end{align*}
\]

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

\[
\begin{align*}
  x_A &= \emptyset (= d_0) \\
  x_B &= f_1(x_A) \\
  x_C &= x_B \sqcup x_E \\
  x_D &= f_3(x_C) \\
  x_E &= f_4(x_D).
\end{align*}
\]

Corresponding \( \bar{f} \) function:

\[
\bar{f}(d_A, d_B, d_C, d_D, d_E) = (d_0, f_1(d_A), d_B \sqcup d_E, f_3(d_C), f_4(d_D)).
\]
Consider “vectorised” lattice $\overline{D} = (D^k, \subseteq)$, where $D$ is the underlying lattice.

Each solution to the equations is a point in this vectorised lattice.

The solutions are precisely the fix-points of the function $\overline{f}: \overline{D} \to \overline{D}$.

If $D$ is a complete lattice and $f_i$’s are monotone, then $\overline{D}$ is complete and $\overline{f}$ is monotone.

- Note: Concrete analysis satisfies these properties.

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 $\bot_D, \overline{f}(\bot_D), \overline{f}^2(\bot_D), \ldots$. 

Correctness

Kildall’s algo always computes LFP of $\bar{f}$.
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 \( d' \subseteq c_M \).
    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 \( \bar{f} \), \( (\bar{f}(\bar{c}))[N] = f_{MN}(c_M) \). Now, since \( \bar{c} \) is an FP of \( \bar{f} \), \( c_N = (\bar{f}(\bar{c}))[N] \). 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 \( d \sqsubseteq c_N \).
Correctness of Kildall

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 $N$, $f_p(d_0) \subseteq c_N$.
- Therefore, JOP $d_N$ at $N$ is $\subseteq c_N$. 
2. JOP = LFP for infinitely-distributive framework

**Proof:** Enough to show that JOP is a fixpoint of \( \bar{f} \).
2. JOP = LFP for infinitely-distributive framework

**Proof:** Enough to show that JOP is a fixpoint of $\overline{f}$. Let $N$ be any program point. Case (the node before $N$ is not a join node):

Points shown are lattice values that reach $M$ and $N$, respectively, due to all paths paths that come via $M$ and end at $N$. Therefore, $d_M$ and $d_N$ are the JOP values at $M$ and $N$. Now, $d_N = f_{MN}(d_M)$ because of infinite distributivity. Therefore, if $\overline{d}$ is any vector s.t. $\overline{d}[M] = d_M$ and $\overline{d}[N] = d_N$, then, by definition of $\overline{f}$, $(\overline{f}(\overline{d}))[N]$ is equal to $d_N$. 
2. JOP = LFP for infinitely-distributive framework

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

- Say $S_M$ is set of lattice values reaching $M$, and $S_P$ is set of lattice values reaching $P$.
- Lattice values reaching $N$ is $S_M \cup S_P$. Therefore, $d_N$ is $\sqcup (S_M \cup S_P)$. It then follows that $d_N = d_M \sqcup d_P$.
- Therefore, if $\overline{d}$ is any vector s.t. $\overline{d}[M] = d_M$, $\overline{d}[P] = d_P$, and $\overline{d}[N] = d_N$, then, by definition of $\overline{f}$, $(\overline{f}(\overline{d}))[N]$ is equal to $d_N$. 
2. JOP = LFP for infinitely-distributive framework

- Since the argument in the previous two slides applies at all points $\mathbb{N}$, we have shown that the vector $\bar{d}$ consisting of all the JOP values is a fix-point of $\bar{f}$.
- Note: Lattice is finite, and functions are pairwise distributive, and $f_i(\bot) = \bot$ implies framework is infinitely distributive.
$f_n^{CP}$ is monotonic

$\bullet$ $f_n^{CP}$ is not distributive.

Consider node $n$ with statement $y := x \times x$, 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, $f_n(P_1) \sqcup f_n(P_2) = \{(y, 1)\}$. 
Let $\overline{d}$ be values computed by Kildall’s algo upon termination, and $\overline{l}$ be LFP of $\overline{f}$.

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 $N$ be any program point whose value gets updated in Step $i + 1$. 

3. Kildall’s algo computes LFP
3. Kildall’s algo computes LFP

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

\[
\begin{align*}
& \text{Explanation:} \\
& \bullet d_M^i \sqsubseteq l_M \text{ and } d_N^i \sqsubseteq l_N \text{ by inductive hypothesis.} \\
& \bullet l_N = f_{MN}(l_M) \text{ because } \overline{l} \text{ is a FP of } \overline{f} \text{ (see argument in first “Case” in proof that JOP } \leq \text{ LFP).} \\
& \bullet \text{Therefore, due to monotonicity of } f_{MN}, \ f_{MN}(d_M^i) \sqsubseteq l_N. \\
& \bullet \text{Hence, } d_{N}^{i+1} \sqsubseteq l_N.
\end{align*}
\]
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 $\overline{l}$ is a FP of $\overline{f}$, by definition of $\overline{f}$, $l_N = l_M \sqcup l_P$. 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 $\overline{d} \leq \overline{l}$. 
We now show that $\bar{d} \geq \bar{f}(\bar{d})$ (i.e. $\bar{d}$ is a postfixpoint of $\bar{f}$)

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 $\bar{f}$, $(\bar{f}(\bar{d}))[N]$ 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 $(\bar{f}(\bar{d}))[N]$.

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

- Let $M$ and $P$ be the points that precede the join node. By definition of $\bar{f}$, $(\bar{f}(\bar{d}))[N]$ 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 $(\bar{f}(\bar{d}))[N]$. 
3. Kildall’s algo computes LFP

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

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

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 \( \overline{f} : \overline{D} \rightarrow \overline{D} \).

- 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, \( \text{JOP} \leq \text{LFP}(\overline{f}) \).
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 $\text{LFP}(\overline{f})$.

- If each $f_i$ is infinitely distributive, then $\text{JOP} = \text{LFP}(\overline{f})$.
- Otherwise, if each $f_i$ is only monotonic, $\text{JOP} \leq \text{LFP}(\overline{f})$.

Kildall’s algorithm, for monotone frameworks:

- Solution at any point during its execution is $\leq \text{LFP}(\overline{f})$.
- If and when it terminates, solution is equal to $\text{LFP}(\overline{f})$.
- Note this is a stronger claim than “Kildall’s algo computes JOP for distributive frameworks” [Kildall, 'POPL 73].
- Kildall is applicable even if equations are not from a program, as long as lattice is complete and each variable occurs in the lhs of a unique equation.
## Summary of sufficient conditions

<table>
<thead>
<tr>
<th></th>
<th>Termination</th>
<th>LFP ≥ JOP</th>
<th>LFP = JOP</th>
<th>Kild computes LFP upon termination</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_{MN}'s monotonic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>No inf. asc. chains</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Inf. distributive</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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.