Lecture 7: Dataflow Analysis Termination and Correctness

17-355/17-655/17-819: Program Analysis

Rohan Padhye and Jonathan Aldrich

February 24, 2021

* Course materials developed with Claire Le Goues

Motivation: Value Set Analysis

- Goal: track the set of integers each variable could have
  - $\sigma : Var \rightarrow Set[Int]$
  - Generalizes constant propagation to track multiple possibilities
- What challenges might there be with this analysis?

Motivation: Value Set Analysis

- Goal: track the set of integers each variable could have
  - $\sigma : Var \rightarrow Set[Int]$
  - Generalizes constant propagation to track multiple possibilities
- What challenges might there be with this analysis?
- Consider the following program

```
1. x := 0
2. x := x + 1
3. if x < limit goto 2
```

Termination of Kildall’s Algorithm

- Recall our analysis of performance
- Key observation: running a flow function on a statement the second time either results in no change, or output information that is *higher* in the lattice.
  - Let’s formalize this using *ascending chains*

Ascending Chain A sequence σ_k is an *ascending chain* iff $n \leqslant m$ implies $\sigma_n \sqsubseteq \sigma_m$

$$\begin{array}{c} \top \\ \uparrow \\ \mathrm{N} \\ \uparrow \\ \bot \end{array}$$

Heights of chains and lattices

- We can use this to define lattice and chain heights

Ascending Chain — A sequence σ_k is an *ascending chain* iff $n \leqslant m$ implies $\sigma_n \sqsubseteq \sigma_m$

Height of an Ascending Chain — An ascending chain σ_k has finite height h if it contains $h + 1$ distinct elements.

Height of a Lattice — A lattice $(L, \sqsubseteq)$ has finite height h if there is an ascending chain in the lattice of height h, and no ascending chain in the lattice has height greater than h

$\top$

$\uparrow$

N

$\uparrow$

$\bot$

height = 2

Termination also requires monotonicity

- Intuition: program analysis will terminate if the lattice height is finite, and if analysis information at each program point only ascends in the lattice.

Careful – we don't compare the input and output of a flow function. Rather we compare successive output information at a given program point.

We need a new property for this—monotonicity!

Monotonicity A function f is *monotonic* iff $\sigma_1 \sqsubseteq \sigma_2$ implies $f(\sigma_1) \sqsubseteq f(\sigma_2)$

Worklist algorithm, revisited

- Slight variant from earlier notes
  - We compute input when we visit a node, rather than when updating its predecessors
  - We test for output change rather than input change
- Goal: make correctness, termination more obvious
  - Exercise to reader: show this is the same as the previous version

```
worklist = ∅
for Node n in cfg
    input[n] = output[n] = ⊥
    add n to worklist
output[programStart] = initialDataflowInformation

while worklist is not empty
    take a Node n off the worklist
    input[n] = ⊔_{k∈preds(n)} output[k]
    newOutput = flow(n, input[n])
    if newOutput ≠ output[n]
        output[n] = newOutput
        for Node j in succs(n)
            add j to worklist
```

Dataflow Analysis Termination Theorem

Theorem 1 (Dataflow Analysis Termination). *If a dataflow lattice* $(L, \sqsubseteq)$ *has finite height, and the corresponding flow functions are monotonic, the worklist algorithm will terminate.*

Proof: consider the following termination metric: $M = |worklist| + EpN * LC(\sigma)$

- $|worklist|$ is the length of the worklist
- *EpN* is the maximum outgoing *E*dges *p*er *N*ode
- *LC(σ)* is the longest ascending chain from σ to ⊤

We must show that *M* is finite and decreases with each loop iteration.

M is finite because…?

Dataflow Analysis Termination Theorem

Theorem 1 (Dataflow Analysis Termination). *If a dataflow lattice $(L, \sqsubseteq)$ has finite height, and the corresponding flow functions are monotonic, the worklist algorithm will terminate.*

Proof: consider the following termination metric: $M = |worklist| + EpN * LC(\sigma)$

- $|worklist|$ is the length of the worklist
- *EpN* is the maximum outgoing *E*dges *p*er *N*ode
- $LC(\sigma)$ is the longest ascending chain from σ to ⊤

We must show that *M* is finite and decreases with each loop iteration.

M is finite because $|worklist|$ is bounded by the number of nodes in the program, because *EpN* is finite, and because the lattice has finite height.

Termination Theorem Proof, Continued

$M = |worklist| + EpN * LC(\sigma)$ decreases with each loop iteration. Why?

$|worklist|$ generally decreases each iteration because we remove one node from it

But—we add to $|worklist|$ when outputs change. This only happens when `newOutput` is different—and therefore higher in the lattice than `output[n]`. Thus, $LC(\sigma)$ decreased by at least one. Since we add at most EpN edges to the worklist, the $EpN * LC(\sigma)$ factor cancels out the added edges, and we still have a net decrease.

```
worklist = ∅
for Node n in cfg
    input[n] = output[n] = ⊥
    add n to worklist
output[programStart] = initialData

while worklist is not empty
    take a Node n off the worklist
    input[n] = ⊔k∈preds(n) output[k]
    newOutput = flow(n, input[n])
    if newOutput ≠ output[n]
        output[n] = newOutput
        for Node j in succs(n)
            add j to worklist
```

Does Zero Analysis terminate?

Theorem 1 (Dataflow Analysis Termination). *If a dataflow lattice* $(L, \sqsubseteq)$ *has finite height, and the corresponding flow functions are monotonic, the worklist algorithm will terminate.*

- We know it has a finite height – a height of 2 for each variable
- Need to show that the flow functions are monotonic

Zero Analysis flow functions are monotonic

Monotonicity A function f is *monotonic* iff $\sigma_1 \sqsubseteq \sigma_2$ implies $f(\sigma_1) \sqsubseteq f(\sigma_2)$

Case $f_Z[\![x := 0]\!](\sigma) = \sigma[x \mapsto Z]$:

Zero Analysis flow functions are monotonic

Monotonicity A function f is *monotonic* iff $\sigma_1 \sqsubseteq \sigma_2$ implies $f(\sigma_1) \sqsubseteq f(\sigma_2)$

Case $f_Z[\![x := y]\!](\sigma) = \sigma[x \mapsto \sigma(y)]$:

What does it mean for an analysis to be correct?

- Intuition: we would like the program analysis results to correctly describe the result of every actual execution of the program
- We'll formalize this using *traces*

Program Trace A trace T of a program P is a potentially infinite sequence $\{c_0, c_1, ...\}$ of program configurations, where $c_0 = E_0, 1$ is called the initial configuration, and for every $i \geqslant 0$ we have $P \vdash c_i \rightsquigarrow c_{i+1}$

What does it mean for an analysis to be correct?

- Intuition: we would like the program analysis results to correctly describe the result of every actual execution of the program
- Now we can define soundness:

Dataflow Analysis Soundness The result $\{\sigma_n \mid n \in P\}$ of a program analysis running on program P is sound iff, for all traces T of P, for all i such that $0 \leqslant i < \mathit{length}(T)$, $\alpha(c_i) \sqsubseteq \sigma_{n_i}$

Exercise 1: Show that a flow function is unsound

- Consider the following (incorrect) flow function for Zero Analysis

$$f_Z[\![x := y + z]\!](\sigma) = \sigma[x \mapsto Z]$$

- Write a program that, when run, will generate a trace proving that this analysis is unsound, i.e. $\alpha(c_i) \not\sqsubseteq \sigma_{n_i}$ for some i in the trace.

Local Soundness shows that flow functions are correct

- Intuition: we want a flow function to map σ_{in} to σ_{out} in a way that matches the instruction's concrete semantics
- This is formalized by *local soundness:*

Local Soundness A flow function f is *locally sound* iff $P \vdash c_i \rightsquigarrow c_{i+1}$ and $\alpha(c_i) \sqsubseteq \sigma_{n_i}$ and $f[\![P[n_i]]\!](\sigma_{n_i}) = \sigma_{n_{i+1}}$ implies $\alpha(c_{i+1}) \sqsubseteq \sigma_{n_{i+1}}$

Zero Analysis is Locally Sound

Local Soundness A flow function f is *locally sound* iff $P \vdash c_i \rightsquigarrow c_{i+1}$ and $\alpha(c_i) \sqsubseteq \sigma_{n_i}$ and $f[\![P[n_i]]\!](\sigma_{n_i}) = \sigma_{n_{i+1}}$ implies $\alpha(c_{i+1}) \sqsubseteq \sigma_{n_{i+1}}$

Case $f_Z[\![x := 0]\!](\sigma_{n_i}) = \sigma_{n_i}[x \mapsto Z]$:

Zero Analysis is Locally Sound

Local Soundness A flow function f is *locally sound* iff $P \vdash c_i \rightsquigarrow c_{i+1}$ and $\alpha(c_i) \sqsubseteq \sigma_{n_i}$ and $f[\![P[n_i]]\!](\sigma_{n_i}) = \sigma_{n_{i+1}}$ implies $\alpha(c_{i+1}) \sqsubseteq \sigma_{n_{i+1}}$

Case $f_Z[\![x := m]\!](\sigma_{n_i}) = \sigma_{n_i}[x \mapsto N]$ where $m \neq 0$:

Zero Analysis is Locally Sound

Local Soundness A flow function f is *locally sound* iff $P \vdash c_i \rightsquigarrow c_{i+1}$ and $\alpha(c_i) \sqsubseteq \sigma_{n_i}$ and $f[\![P[n_i]]\!](\sigma_{n_i}) = \sigma_{n_{i+1}}$ implies $\alpha(c_{i+1}) \sqsubseteq \sigma_{n_{i+1}}$

Case $f_Z[\![x := y \; op \; z]\!](\sigma_{n_i}) = \sigma_{n_i}[x \mapsto \top]$:

Zero Analysis is Locally Sound

Local Soundness A flow function f is *locally sound* iff $P \vdash c_i \rightsquigarrow c_{i+1}$ and $\alpha(c_i) \sqsubseteq \sigma_{n_i}$ and $f[\![P[n_i]]\!](\sigma_{n_i}) = \sigma_{n_{i+1}}$ implies $\alpha(c_{i+1}) \sqsubseteq \sigma_{n_{i+1}}$

Exercise 2: prove the case for $f_Z[\![x := y]\!](\sigma) = \sigma[x \mapsto \sigma(y)]$

Proving Dataflow Analysis Soundness

- We'd like to prove that a dataflow analysis is sound. Local soundness is part of it.
- We also need Kildall's algorithm to compute an analysis *fixed point:*

Fixed Point A dataflow analysis result $\{\sigma_i \mid i \in P\}$ is a fixed point iff $\sigma_0 \sqsubseteq \sigma_1$ where σ_0 is the initial analysis information and σ_1 is the information before the first instruction, and for each instruction i we have $\bigsqcup_{j \in \texttt{preds}(i)} f[\![P[j]]\!](\sigma_j) \sqsubseteq \sigma_i$.

Kildall's Algorithm computers a fixed point

Fixed Point A dataflow analysis result $\{\sigma_i \mid i \in P\}$ is a fixed point iff $\sigma_0 \sqsubseteq \sigma_1$ where σ_0 is the initial analysis information and σ_1 is the information before the first instruction, and for each instruction i we have $\bigsqcup_{j \in \mathtt{preds}(i)} f[\![P[j]]\!](\sigma_j) \sqsubseteq \sigma_i$.

- Kildall's Algorithm computes a fixed point when it terminates.
- Proof: the following is a loop invariant of Kildall's Algorithm
  $\forall i \,.\, (\exists j \in preds(i)\ such\ that\ f[\![P[j]]\!](\sigma_j) \not\sqsubseteq \sigma_i) \Rightarrow i \in \mathtt{worklist}$
- Trivially true at beginning, since everything is on the worklist
- When instruction i is removed from the worklist, the invariant is established
- When the worklist is empty, the above is equivalent to a fixed point

Finally, we prove global soundness

Theorem 2 (A fixed point of a locally sound analysis is globally sound). *If a dataflow analysis's flow function f is monotonic and locally sound, and for all traces T we have $\alpha(c_0) \sqsubseteq \sigma_0$ where σ_0 is the initial analysis information, then any fixed point $\{\sigma_n \mid n \in P\}$ of the analysis is sound.*

- Consider an arbitrary trace T. The proof is by induction on the configurations c_i in the trace.

Finally, we prove global soundness

Theorem 2 (A fixed point of a locally sound analysis is globally sound). *If a dataflow analysis's flow function f is monotonic and locally sound, and for all traces T we have $\alpha(c_0) \sqsubseteq \sigma_0$ where σ_0 is the initial analysis information, then any fixed point $\{\sigma_n \mid n \in P\}$ of the analysis is sound.*

Case c_0:

Finally, we prove global soundness

Theorem 2 (A fixed point of a locally sound analysis is globally sound). *If a dataflow analysis's flow function f is monotonic and locally sound, and for all traces T we have $\alpha(c_0) \sqsubseteq \sigma_0$ where σ_0 is the initial analysis information, then any fixed point $\{\sigma_n \mid n \in P\}$ of the analysis is sound.*

Case c_{i+1}: