Lecture Notes on Monotone Frameworks

15-411: Compiler Design
André Platzer

Lecture 27

1 Introduction

More information on dataflow analysis and monotone frameworks can be found in [NNH99]. More information on abstract interpretation can be found in [CC92, CC77, CC79] and [WM95, Chapter 10].

2 Monotone Frameworks

Even though all of them are different, the **forward/backward may/must dataflow analysis** are nevertheless very similar. They **all follow a more general pattern**:

$$\begin{aligned} A_\circ(\ell) &= \begin{cases} \iota & \text{if } \ell \in E \\ \bigsqcup\{A_\bullet(\ell') \;:\; (\ell',\ell) \in F\} & \text{otherwise} \end{cases} \\ A_\bullet(\ell) &= f_\ell(A_\circ(\ell)) \end{aligned}$$

where, depending on the specific analysis:

- the operator $\bigsqcup$ is either $\bigcup$ for information from any source or $\bigcap$ for information joint to all sources
- the flow relation F is either the forward control flow or the backward control flow
- the initialization set E is either the initial block or the set of final nodes
- ι specifies the starting point of the analysis at the initial or final nodes

- f_ℓ is the transfer function for the node, which, in the previous examples is always of the special form

$$f_\ell(X) = (X \setminus kill(\ell)) \cup gen(\ell)$$

More formally, the property that we are analyzing is part of a ***property space*** L. This space L could be the set of all sets of variables $\wp(Vars)$, if we are looking for the set of all **live variables**. Or, for available expressions, it could be the set of all sets of expressions $\wp(Expr)$ ordered by $\supseteq$. Or, in fairly advanced analyses, we might even be tempted to try the set of all mappings $Vars \to \mathbb{Z}^2$ from variables to intervals, if we are trying to find interval bounds for each variable. The latter scenario is more difficult, though.

For the property space and the way how property values flow through the control flow, we need a number of assumptions for the above approach to work.

Definition 1 *A* monotone framework *consists of*

1. *a* ***Noetherian complete semi-lattice*** *L: a set L with a* ***partial order*** *$\sqsubseteq$ that is* ***complete****, i.e., such that* each *subset $Y \subseteq L$* ***has a least upper bound*** *$\bigsqcup Y$. A lattice is* ***Noetherian iff*** *it satisfies the condition that* ***each ascending chain***

$$a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \ldots$$

   is finite*, i.e., there is an n such that $a_n = a_{n+1} = a_{n+2} = \ldots$.*

2. *a set $\mathcal{F}$ of monotone functions $f : L \to L$ that contains the identify function* ***id*** *$: L \to L; a \mapsto a$ and is closed under composition, i.e., if $f, g \in \mathcal{F}$ then the composition $f \circ g \in \mathcal{F}$. A function $f : L \to L$ is* ***monotone*** *iff*

$$a \sqsubseteq b \text{ implies } f(a) \sqsubseteq f(b)$$

A **"distributive" framework** *is a monotone framework where each $f \in \mathcal{F}$ is distributive (or, more precisely, a homomorphism)*

$$f(a \sqcup b) = f(a) \sqcup f(b)$$

Note that we use the binary operator notation $a \sqcup b$ as an abbreviation for the more verbose $\bigsqcup\{a, b\}$. In addition, we denote the least upper bound $\bigsqcup \emptyset$ of the empty set $\emptyset \subseteq L$ by $\bot$, which is the least element of semi-lattice L.

Definition 2 *The analysis equations corresponding to a monotone framework are*

$$A_\circ(\ell) = \bigsqcup\{A_\bullet(\ell') \ : \ (\ell',\ell) \in F\} \sqcup \begin{cases} \iota & \textit{if } \ell \in E \\ \bot & \textit{if } \ell \notin E \end{cases} \tag{1}$$

$$A_\bullet(\ell) = f_\ell(A_\circ(\ell)) \tag{2}$$

Table 1: Dataflow analysis examples as monotone frameworks

	L	$\sqsubseteq$	$\bigsqcup$	$\bot$	ι	E	F
For-may: Reach def	$\wp(Lbl)$	$\subseteq$	$\bigcup$	$\emptyset$	Lbl	$init$	$flow$
For-must: Available expr	$\wp(Exp)$	$\supseteq$	$\bigcap$	Exp	$\emptyset$	$init$	$flow$
Back-may: Live variables	$\wp(Var)$	$\subseteq$	$\bigcup$	$\emptyset$	$\emptyset$	$final$	$flow^{-1}$
Back-must: Very busy expr	$\wp(Exp)$	$\supseteq$	$\bigcap$	Exp	$\emptyset$	$final$	$flow^{-1}$

$$f_\ell(l) = (l \setminus kill(\ell)) \cup gen(\ell)$$
$$\mathcal{F} = \{f : L \to L \ : \ f(l) = (l \setminus kill) \cup gen \text{ for some } kill, gen\}$$

3 Solving Monotone Framework Equations as Least Fixed Points

A simple way of solving the analysis equations of a monotone framework works by **iteratively updating the left hand side of (1) to match the right hand side of (1) until nothing changes** anymore. This is the worklist algorithm[1]:

```
W := F      // working list
 for $\ell \in F \cup E$
      if ($\ell \in E$) then $A_\circ[\ell]$ := $\iota$ else $A_\circ[\ell]$ := $\bot$
 for $(\ell',\ell) \in W$
       W := W $\setminus\{(\ell',\ell)\}$
        if $f_{\ell'}(A_\circ[\ell']) \not\sqsubseteq A_\circ[\ell]$ then
             $A_\circ[\ell]$ := $A_\circ[\ell] \sqcup f_{\ell'}(A_\circ[\ell'])$
             W := $\{(\ell,\ell'') : (\ell,\ell'') \in F\} \cup$ W
  end for
```

[1]Also called "Maximum" Fixed Point, which is rather confusing for a least fixed point.

Does this algorithm solve the equations (1)? This algorithm computes the least fixed point of (1), because it starts at the bottom $\bot$ of the semi-lattice and successively follows the fixed point condition. It also always terminates. Let's convince ourselves why.

Theorem 3 (Kleene fixed point theorem) *If L is a complete partial order and $f : L \to L$ is a **Scott-continuous function** (i.e., $\bigsqcup f(Y) = f(\bigsqcup Y)$ for every subset $Y \subseteq L$ with a supremum $\bigsqcup Y \in L$), then the least fixed point μf of f is*

$$\mu f = \bigsqcup\{f^n(\bot) \ : \ n \in \mathbb{N}\}$$

The Kleene fixed point theorem is very useful, because it **shows that successive iteration** like we are doing in the worklist algorithm really **yields the least fixed point**. Yet are we in a position to use the theorem? It doesn't quite look like it. **We have montone transfer functions but need Scott-continuous functions**. Every **Scott-continuous function $f : L \to L$ is monotone**:

$$a \sqsubseteq b \ \Rightarrow \ \bigsqcup\{f(a), f(b)\} = f(\bigsqcup\{a, b\}) = f(b) \ \Rightarrow \ f(a) \sqsubseteq f(b)$$

But in monotone frameworks, we deal with monotone functions $f : L \to L$ and need to know whether they are Scott-continuous. Now fortunately, the semi-lattice L underlying monotone frameworks is actually **Noetherian**. Thus, if we start with any set $Y \subseteq L$ and want to compare $\bigsqcup f(Y)$ and $f(\bigsqcup Y)$, we first note that **Y cannot contain an infinite ascending chain** but can only contain a finite ascending chain:

$$a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq ... \sqsubseteq a_n$$

Now **by monotonicity** of f we know

$$f(a_1) \sqsubseteq f(a_2) \sqsubseteq f(a_3) \sqsubseteq ... \sqsubseteq f(a_n)$$

Thus,

$$\bigsqcup\{f(a_1), \ldots, f(a_n)\} = f(a_n) = f(\bigsqcup\{a_1, \ldots, a_n\})$$

The **same argument holds for all ascending chains in Y**. **The ends of all those finite ascending chains have least upper bounds, because L is a lattice**. **Because L is Noetherian, even those extensions can only give finite ascending chains**.

The only trouble is that the function f we are using in the algorithm to form a fixed point is not just one of the transfer functions f_ℓ, which we know to be monotone. The **overall function f is not a transfer function but really**

$$f(Z)(\ell) := \bigsqcup\{f_{\ell'}(Z(\ell')) \,:\, (\ell',\ell) \in F\} \sqcup \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases}$$

To be precise, let's represent the function of ℓ as a relation instead:

$$\begin{aligned} f(Z) := \; & \left\{\left(\ell, \bigsqcup\{f_{\ell'}(Z(\ell')) \,:\, (\ell',\ell) \in F\}\right) \;:\; \ell \in E \cup F\right\} \\ & \sqcup \{(\ell,\iota) \,:\, \ell \in E\} \sqcup \{(\ell,\bot) \,:\, \ell \in F \setminus E\} \end{aligned}$$

Now this function is more complicated because it has the nodes ℓ and ℓ' as extra arguments, so the domain is a slightly different one and things become more complicated. Nevertheless, the principles above still apply and we can see that f is monotone, because the result only increases at every point if the input increases.

Why does the worklist algorithm terminate? That's easy to see if the semi-lattice L is finite, because there can only be finitely many monotone changes to the sets then. What if the property space L is infinite? Well the algorithm still terminates, because $A_\circ[\ell]$ increases at its assignment, and it can only increase to a finite ascending chain, not an infinite one, because L is Noetherian.

References

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[CC79] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.

[CC92] Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Log. Program.*, 13(2&3):103–179, 1992.

[NNH99] F. Nielson, H. R. Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999.

[WM95] Reinhard Wilhelm and Dieter Maurer. *Compiler Design*. Addison-Wesley, 1995.