Control-flow analysis

Lecture 22 Thursday, April 17, 2014

1 Dataflow analysis for functional programs

Last lecture we looked at abstract interpretation of IMP, an imperative programming language. Abstract interpretation provides a way to statically approximate the runtime behavior of a program more precisely than the type systems that we have seen so far.

However, how would we perform abstract interpretation for a functional language? That is, in a functional language, functions are values, and to perform abstract interpretation on a functional language, we would need an abstract domain for functions.

In this lecture we will consider *dataflow analysis* for functional programs. Dataflow analysis is a technique for statically determining what "facts" are true at various points in the program. The dataflow facts often concern what the set of possible values that variables may have at certain program points. (Dataflow analysis and abstract interpretation are related to each other, and in some ways are similar to each other; we will not dig into the precise connection between them here.)

In order to perform dataflow analysis (i.e., to track possible values through a program), we need to know the control flow of a program. In IMP, this is straightforward: the lexical structure of the program tells us the control flow structure. But in a functional language, if we see a application $f\ y$, then control will jump to the function body of whatever function the variable f evaluates to. This makes determining the control flow of a functional language more complex than determining the control flow structure of a language like IMP, where functions are not first-class values. (Similar issues arise in object-oriented languages, where objects are first class and contain executable code.)

To determine which functions the variable f may evaluate to, we can perform dataflow analysis to approximate the values that f may evaluate to. So in a functional language, we need to perform dataflow analysis and control-flow analysis simultaneously.

We will consider the 0CFA analysis. (*CFA* stands for Control-Flow Analysis; it is an instance of k-CFA analysis, where the parameter k determines the precision of the analysis.) Like our type inference analysis, we will generate a set of constraints from a program, and then find a solution to the set of constraints.

1.1 Labeled lambda calculus

Let's consider a lambda calculus with integers. The syntax of the language is similar to what we've seen before, but every expression in our source program will have a unique label. Labels are used to uniquely identify program expressions. We use l to range over labels.

$$e ::= n^l \mid x^l \mid (\lambda x.\, e)^l \mid (e_1\ e_2)^l \mid (e_1 + e_2)^l$$

Every expression has a label, and we write $\mathit{labelof}(e)$ for the label of expression e. For convenience, we also write e^l to mean that l is a label such that $\mathit{labelof}(e) = l$.

Also, given a program e, we write $\mathit{exprof}(e, l)$ for the (unique) subexpression e' of e such that $\mathit{labelof}(e') = l$. That is $\mathit{exprof}(e, l)$ allows us to get the expression associated with label l.

We will use a large-step environment semantics for this analysis (see Lecture 17). Recall that environments ρ are maps from variables to values, and that judgment $\langle e, \rho\rangle \Downarrow v$ means that expression e in environment ρ evaluates to value v. Note that in an environment semantics, a function $\lambda x.\, e$ evaluates to a *closure* $((\lambda x.\, e)^l, \rho_{lex})$, where ρ_{lex} is the environment that was current when $\lambda x.\, e$ was evaluated. That is, ρ_{lex} binds the free variables (except x) in the function body e. We define the label of a closure to be the label of the function: $\mathit{labelof}(((\lambda x.\, e)^l, \rho_{lex})) = l$.

Values in our language are thus the following.

$$v ::= n^l \mid ((\lambda x.\, e)^l, \rho)$$

$$\frac{}{\langle x^l, \rho\rangle \Downarrow \rho(x)} \qquad \frac{}{\langle n^l, \rho\rangle \Downarrow n^l} \qquad \frac{\langle e_1, \rho\rangle \Downarrow n_1^{l_1} \quad \langle e_2, \rho\rangle \Downarrow n_2^{l_1}}{\langle (e_1 + e_2)^l, \rho\rangle \Downarrow n^l}\, n = n_1 + n_2$$

$$\frac{}{\langle (\lambda x.\, e)^l, \rho\rangle \Downarrow ((\lambda x.\, e)^l, \rho)} \qquad \frac{\langle e_1, \rho\rangle \Downarrow ((\lambda x.\, e)^{l_1}, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho_{lex}[x \mapsto v_2]\rangle \Downarrow v}{\langle (e_1\ e_2)^l, \rho\rangle \Downarrow v}$$

Notice that every value is labeled with the label of the expression that created it. (For integer values, the label is either the label of the integer literal that appeared in the source program, or the label of an addition.)

1.2 Analysis

The aim of our analysis is to approximate the values that expressions can evaluate to. Since functions are values, this will also tell us about control flow, since given an application $e_1\ e_2$, knowing which function values expression e_1 may evaluate to tells us about the possible control flow of the function application.

Our analysis will find a function $\mathsf{C} : \textbf{Label} \to \mathcal{P}(\textbf{Label})$ that approximates for each label l the set of values that the expression labeled l may evaluate to. Specifically, if $l' \in \mathsf{C}(l)$, then the expression labeled l may evaluate to a value labeled l'.

Our analysis will also find a function $\mathsf{r} : \textbf{Var} \to \mathcal{P}(\textbf{Label})$ that for each variable x will approximate the set of values that x may be bound to. That is, $l \in \mathsf{r}(x)$, then the variable x may be bound to a value labeled l'. We assume without loss of generality that variable names in a program are unique. (If we don't have this assumption, the analysis will still work, it will just be less precise.)

Given a program e, we produce a set of constraints on functions C and C by examining the program. Intuitively, if we can find functions C and r that satisfy the constraints, then C and r will give us correct information about what values expressions and variables may evaluate to.

We generate the set of constraints using function $\mathcal{C}[\![\cdot]\!]_e : \textbf{Expr} \to \mathcal{P}(\textbf{Constraint})$. Here, e is the program we are analyzing, and we use it in order to map labels to expressions. $\mathcal{C}[\![\cdot]\!]_e$ is defined as follows.

$$\begin{aligned}
\mathcal{C}[\![n^l]\!]_e &= \{l \in \mathsf{C}(l)\} \\
\mathcal{C}[\![(e_1 + e_2)^l]\!]_e &= \mathcal{C}[\![e_1]\!]_e \cup \mathcal{C}[\![e_2]\!]_e \cup \{l \in \mathsf{C}(l)\} \\
\mathcal{C}[\![x^l]\!]_e &= \{\mathsf{r}(x) \subseteq \mathsf{C}(l)\} \\
\mathcal{C}[\![(\lambda x.\, e_1)^l]\!]_e &= \{l \in \mathsf{C}(l)\} \cup \mathcal{C}[\![e_1]\!]_e \\
\mathcal{C}[\![(e_1^{l_1}\ e_2^{l_2})^l]\!]_e &= \mathcal{C}[\![e_1^{l_1}]\!]_e \cup \mathcal{C}[\![e_2^{l_2}]\!]_e \\
&\quad \cup \{l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_2) \subseteq \mathsf{r}(x) \mid \mathit{exprof}(e, l') = (\lambda x.\, e_0^{l_0})^{l'}\} \\
&\quad \cup \{l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_0) \subseteq \mathsf{C}(l) \mid \mathit{exprof}(e, l') = (\lambda x.\, e_0^{l_0})^{l'}\}
\end{aligned}$$

Let's consider what each of these constraints mean. For values n^l, $(\lambda x.\, e_1)^l$ and addition $(e_1 + e_2)^l$ we have constraint $l \in \mathsf{C}(l)$. This means that the expression labeled l may evaluate to a value labeled l, and, if we look at the semantics, it is indeed the case.

$\mathcal{C}[\![x^l]\!]_e$ produces the constraint $\mathsf{r}(x) \subseteq \mathsf{C}(l)$, meaning that if x may be bound to a value labeled l' (i.e., $l' \in \mathsf{r}(x)$), then the expression x^l may evaluate to that value (i.e., $l' \in \mathsf{C}(l)$.

The constraints for application $(e_1^{l_1}\ e_2^{l_2})^l$ are most interesting. The *conditional constraint* $l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_2) \subseteq \mathsf{r}(x)$ requires that if expression e_1 may evaluate to function value $(\lambda x.\, e_0^{l_0})^{l'}$ (i.e., $l' \in \mathsf{C}(l_1)$), then x, the argument of that function, may be bound to anything that e_2 can evaluate to (i.e., $\mathsf{C}(l_2) \subseteq \mathsf{r}(x)$).

Similarly, constraint $l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_0) \subseteq \mathsf{C}(l)$ requires that if expression e_1 may evaluate to function value $(\lambda x.\, e_0^{l_0})^{l'}$ (i.e., $l' \in \mathsf{C}(l_1)$), then the application expression may evaluate to anything that function body e_0 may evaluate to (i.e., $\mathsf{C}(l_0) \subseteq \mathsf{C}(l)$).

Solving these constraints is straightforward. We will start off with a very bad approximation to the functions: $\mathsf{C}_0(l) = \emptyset$ for all labels l, and $\mathsf{r}_0(x) = \emptyset$ for all variables x. We will then iteratively improve these approximations by adding labels to the sets to satisfy the constraints. We will keep iterating until we reach a fixed point. Since there are only a finite number of labels, and in each iteration we only add labels to the sets, we are guaranteed to reach a fixed point.

$$\begin{aligned}
\mathsf{C}_0 &= \lambda l.\, \emptyset \\
\mathsf{r}_0 &= \lambda x.\, \emptyset
\end{aligned}$$

$$\begin{aligned}
\mathsf{C}_{i+1} &= \lambda l.\, \mathsf{C}_i(l) \\
&\quad \cup \{l \mid (l \in \mathsf{C}(l)) \in \mathcal{C}[\![e]\!]_e\} \\
&\quad \cup \bigcup \{\mathsf{r}_i(x) \mid (\mathsf{r}(x) \subseteq \mathsf{C}(l)) \in \mathcal{C}[\![e]\!]_e\} \\
&\quad \cup \bigcup \{\mathsf{C}_i(l_0) \mid (l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_0) \subseteq \mathsf{C}(l)) \in \mathcal{C}[\![e]\!]_e \text{ and } l' \in \mathsf{C}_i(l_1)\} \\
\mathsf{r}_{i+1} &= \lambda x.\, \mathsf{r}_i(x) \\
&\quad \cup \bigcup \{\mathsf{C}_i(l_2) \mid (l' \in \mathsf{C}(l_1) \Rightarrow \mathsf{C}(l_2) \subseteq \mathsf{r}(x)) \in \mathcal{C}[\![e]\!]_e \text{ and } l' \in \mathsf{C}_i(l_1)\}
\end{aligned}$$

The least fixed point, which we will denote C^* and r^* is the bound of all of the approximations.

$$\begin{aligned}
\mathsf{C}^* &= \lambda l.\, \bigcup_{i \in \mathbb{N}} \mathsf{C}_i(l) \\
\mathsf{r}^* &= \lambda x.\, \bigcup_{i \in \mathbb{N}} \mathsf{r}_i(x)
\end{aligned}$$

1.3 Example

Let's work through a simple example. Consider the following program.

$$e \;\equiv\; (((\lambda a.\, a^1)^2\ (\lambda b.\, b^3)^4)^5\ 99^6)^7$$

The set of constraints for this simple program is as follows. (Exercise: make sure you understand how these constraints were derived.)

$$\begin{aligned}
\mathcal{C}[\![e]\!]_e = \{\ & 2 \in \mathsf{C}(2), 4 \in \mathsf{C}(4), 6 \in \mathsf{C}(6), \\
& \mathsf{r}(a) \subseteq \mathsf{C}(1), \mathsf{r}(b) \subseteq \mathsf{C}(3), \\
& 2 \in \mathsf{C}(5) \Rightarrow \mathsf{C}(6) \subseteq \mathsf{r}(a), \\
& 4 \in \mathsf{C}(5) \Rightarrow \mathsf{C}(6) \subseteq \mathsf{r}(b), \\
& 2 \in \mathsf{C}(5) \Rightarrow \mathsf{C}(1) \subseteq \mathsf{C}(7), \\
& 4 \in \mathsf{C}(5) \Rightarrow \mathsf{C}(3) \subseteq \mathsf{C}(7), \\
& 2 \in \mathsf{C}(2) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{r}(a), \\
& 4 \in \mathsf{C}(2) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{r}(b), \\
& 2 \in \mathsf{C}(2) \Rightarrow \mathsf{C}(1) \subseteq \mathsf{C}(5), \\
& 4 \in \mathsf{C}(2) \Rightarrow \mathsf{C}(3) \subseteq \mathsf{C}(5)\ \}
\end{aligned}$$

Let's consider the result of solving it iteratively. In the following table, columns indicate the values of $\mathsf{C}_i(l)$ and $\mathsf{r}_i(x)$ over the various iterations.

i	$\mathsf{C}_i(1)$	$\mathsf{C}_i(2)$	$\mathsf{C}_i(3)$	$\mathsf{C}_i(4)$	$\mathsf{C}_i(5)$	$\mathsf{C}_i(6)$	$\mathsf{C}_i(7)$	$\mathsf{r}_i(a)$	$\mathsf{r}_i(b)$
0									
1		2		4		6			
2		2		4		6		4	
3	4	2		4		6		4	
4	4	2		4	4	6		4	
5	4	2		4	4	6		4	6
6	4	2	6	4	4	6		4	6
7	4	2	6	4	4	6	6	4	6
8	4	2	6	4	4	6	6	4	6

At the 8th iteration, we have $\mathsf{C}_7 = \mathsf{C}_8$ and $\mathsf{r}_7 = \mathsf{r}_8$, and we have reached a fixed point, and so $\mathsf{C}^* = \mathsf{C}_8$ and $\mathsf{r}^* = \mathsf{r}_8$.

Let's double check that this analysis returned reasonable results. For example, $\mathsf{C}^*(5) = \{4\}$, meaning that the expression $((\lambda a.\, a^1)^2\ (\lambda b.\, b^3)^4)^5$ may evaluate it a value labeled 4, i.e., to the value $(\lambda b.\, b^3)^4$. That is indeed consistent with the actual execution of the program. Another example: $\mathsf{C}^*(7) = \{6\}$, meaning that the whole program may evaluate to a value labeled 6, i.e., to the labeled integer 99^6.

Note that $2 \notin \mathsf{C}^*(5)$. That is, the analysis correctly says that expression $((\lambda a.\, a^1)^2\ (\lambda b.\, b^3)^4)^5$ can not evaluate to $(\lambda a.\, a^1)^2$.

1.4 Soundness

The analysis is sound, meaning that, given a program e_0, if r^* and C^* satisfy the set of constraints $\mathcal{C}[\![e_0]\!]_{e_0}$, then C^* conservatively describes what expressions may evaluate to, and r^* conservatively describes what variables may be bound to.

Theorem (Soundness)**.** *Let e_0 be a program. Let r^* and C^* satisfy the set of constraints $\mathcal{C}[\![e_0]\!]_{e_0}$. If $\langle e_0, \emptyset\rangle \Downarrow v_0$ and $\langle e, \rho\rangle \Downarrow v$ appears in the derivation of $\langle e_0, \emptyset\rangle \Downarrow v_0$, then $\mathit{labelof}(v) \in \mathsf{C}^*(\mathit{labelof}(e))$.*

In order to prove the soundness theorem, we need a stronger lemma. To state the lemma, we will extend the set-constraint generation function to environments and closures.

$$\mathcal{C}[\![\rho]\!]_e = \bigcup_{x \in \mathrm{dom}(\rho)} \{\mathsf{C}(\mathit{labelof}(\rho(x))) \subseteq \mathsf{r}(x)\} \cup \mathcal{C}[\![\rho(x)]\!]_e$$

$$\mathcal{C}[\![((\lambda x.\, e_1)^l, \rho)]\!]_e = \mathcal{C}[\![(\lambda x.\, e_1)^l]\!]_e \cup \mathcal{C}[\![\rho]\!]_e$$

With this extended definition in hand, we can state the lemma, which can then be proved by induction on derivations $\langle e, \rho\rangle \Downarrow v$.

Lemma. *Let e_0 be a program, e an expression and ρ an environment. Let r^* and C^* satisfy the constraints $\mathcal{C}[\![(e, \rho)]\!]_{e_0}$. If $\langle e, \rho\rangle \Downarrow v$ then $\mathit{labelof}(v) \in \mathsf{C}^*(\mathit{labelof}(e))$ and r^* and C^* satisfy the constraints $\mathcal{C}[\![v]\!]_{e_0}$.*