Lecture 13: Control-Flow Analysis for Functional Programming Languages

17-355/17-665/17-819: Program Analysis

Rohan Padhye

March 3, 2022

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Analyzing Functional Programming Languages

$$\begin{array}{lll} e & \in & \mathit{Expressions} \\ t & \in & \mathit{Term} \\ l & \in & \mathcal{L} \end{array}$$

...or labelled terms

...or unlabelled expressions

labels

$$\begin{array}{rcl} e & ::= & t^l \\ t & ::= & \lambda x.e \\ & | & x \\ & | & (e_1)\ (e_2) \\ & | & \text{let } x = e_1 \text{ in } e_2 \\ & | & \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \\ & | & n \mid e_1 + e_2 \mid \ldots \end{array}$$

How to analyze these programs?

- $(\lambda x. x + 1)(3)$
- $(\lambda f. f\ 3)(\lambda x. x + 1)$
- $\textbf{let}\ add = \lambda x. \lambda y. x + y\ \textbf{in}$
  $\textbf{let}\ addfive = (add\ 5)\ \textbf{in}$
  $\textbf{let}\ addsix = (add\ 6)\ \textbf{in}$
  $addfive\ 2$

Analysis of Labelled programs

$$(((\lambda f.(f^a\ 3^b)^c)^e(\lambda x.(x^g + 1^h)^i)^j)^k$$

What values can occur at labelled program points?

Control-Flow Analysis / 0-CFA

- Static analysis of functional languages
- Similar to data-flow analysis but without explicit CFG
- Analysis definition is syntax-driven, similar to specifying semantics
- Static analysis is hence a form of giving a program abstract semantics
- σ needs to map not just variables but also expression labels
    - The labels are "program points" similar to CFG nodes
    - The edges are implicit in the nested syntax (no loops to worry about)
- σ(x) may be a variable OR a function, and both must be tracked
    - Higher-order function application is resolved while doing the analysis
    - Hence the name "control-flow analysis", but usually just CFA
- 0-CFA is the simplest, context-insensitive variant

0-CFA for Constant Propagation

$$\sigma \in Var \cup \mathcal{L} \to L$$

$$L = \mathbb{Z} + \top + \mathcal{P}(\lambda x.e)$$

Question: what is the $\sqsubseteq$ *relation on this dataflow state?*

0-CFA Rules

$$\frac{}{[\![n]\!]^l \hookrightarrow \alpha(n) \sqsubseteq \sigma(l)}\ \textit{const}$$

$$\frac{}{[\![x]\!]^l \hookrightarrow \sigma(x) \sqsubseteq \sigma(l)}\ \textit{var}$$

$$\frac{[\![e]\!]^{l_0} \hookrightarrow C}{[\![\lambda x.e^{l_0}]\!]^l \hookrightarrow \{\lambda x.e\} \sqsubseteq \sigma(l) \cup C}\ \textit{lambda}$$

$$\frac{[\![e_1]\!]^{l_1} \hookrightarrow C_1 \quad [\![e_2]\!]^{l_2} \hookrightarrow C_2}{[\![e_1^{l_1}\ e_2^{l_2}]\!]^l \hookrightarrow C_1 \cup C_2 \cup \mathbf{fn}\ l_1 : l_2 \Rightarrow l}\ \textit{apply}$$

0-CFA Rules

$$\frac{\lambda x.e_0^{l_0} \in \sigma(l_1)}{\textbf{fn}\ l_1 : l_2 \Rightarrow l \hookrightarrow \sigma(l_2) \sqsubseteq \sigma(x) \wedge \sigma(l_0) \sqsubseteq \sigma(l)}\ \textit{function-flow}$$

$$\frac{[\![e_1]\!]^{l_1} \hookrightarrow C_1 \quad [\![e_2]\!]^{l_2} \hookrightarrow C_2}{[\![e_1^{l_1}\ e_2^{l_2}]\!]^{l} \hookrightarrow C_1 \cup C_2 \cup \textbf{fn}\ l_1 : l_2 \Rightarrow l}\ \textit{apply}$$

0-CFA Rules

Question: what might the rules for the if-then-else or arithmetic operator expressions look like?

$$\frac{[\![e_1]\!]^{l_1} \hookrightarrow C_1 \quad [\![e_2]\!]^{l_2} \hookrightarrow C_2}{[\![e_1^{l_1}\ e_2^{l_2}]\!]^{l} \hookrightarrow C_1 \cup C_2 \cup \mathbf{fn}\ l_1 : l_2 \Rightarrow l}\ \textit{apply}$$

0-CFA Rules

$$\frac{}{[\![n]\!]^l \hookrightarrow \alpha(n) \sqsubseteq \sigma(l)}\ \textit{const}$$

$$\frac{[\![e_1]\!]^{l_1} \hookrightarrow C_1 \quad [\![e_2]\!]^{l_2} \hookrightarrow C_2}{[\![e_1^{l_1} + e_2^{l_2}]\!]^l \hookrightarrow C_1 \cup C_2 \cup (\sigma(l_1) +_\top \sigma(l_2)) \sqsubseteq \sigma(l)}\ \textit{plus}$$

0-CFA Example

$$((\lambda x.(x^a + 1^b)^c)^d (3)^e)^g$$

Simple 0-CFA Example

$$((\lambda x.(x^a + 1^b)^c)^d(3)^e)^g$$

$(\sigma(x) \sqsubseteq \sigma(a))$	*var*
$(\{\lambda x.x + 1\} \sqsubseteq \sigma(d))$	*lambda*
$(\sigma(e) \sqsubseteq \sigma(x)) \wedge (\sigma(c) \sqsubseteq \sigma(g))$	*apply* *function-flow*
$(\alpha(3) \sqsubseteq \sigma(e))$	*const*
$(\alpha(1) \sqsubseteq \sigma(b))$	*const*
$(\sigma(a) +_\top \sigma(b) \sqsubseteq \sigma(c))$	*plus*

Simple 0-CFA Example

$$((\lambda x.(x^a + 1^b)^c)^d(3)^e)^g$$

$$(\sigma(x) \sqsubseteq \sigma(a))$$

$$(\{\lambda x.x + 1\} \sqsubseteq \sigma(d))$$

$$(\sigma(e) \sqsubseteq \sigma(x)) \wedge (\sigma(c) \sqsubseteq \sigma(g))$$

$$(\alpha(3) \sqsubseteq \sigma(e))$$

$$(\alpha(1) \sqsubseteq \sigma(b))$$

$$(\sigma(a) +_\top \sigma(b) \sqsubseteq \sigma(c))$$

Label	Abstract Value

Exercise: 0-CFA with Constant Propagation

$$(((\lambda f.(f^a\ 3^b)^c)^e(\lambda x.(x^g + 1^h)^i)^j)^k$$

Label	Abstract Value

Exercise: 0-CFA with Constant Propagation

$$((((\lambda f.(f^a\ 3^b)^c)^e(\lambda x.(x^g + 1^h)^i)^j)^k$$

$Var \cup Lab$	L	by rule
e	$\lambda f.f\ 3$	lambda
j	$\lambda x.x+1$	lambda
f	$\lambda x.x+1$	apply
a	$\lambda x.x+1$	var
b	3	const
x	3	apply
g	3	var
h	1	const
i	4	add
c	4	apply
k	4	apply

Context Sensitivity

$$\textbf{let}\ add = \lambda x.\ \lambda y.\ x + y$$
$$\textbf{let}\ add5 = (add\ 5)^{a5}$$
$$\textbf{let}\ add6 = (add\ 6)^{a6}$$
$$\textbf{let}\ main = (add5\ 2)^{m}$$

Context Sensitivity

$$\textbf{let } add = \lambda x.\ \lambda y.\ x + y$$
$$\textbf{let } add5 = (add\ 5)^{a5}$$
$$\textbf{let } add6 = (add\ 6)^{a6}$$
$$\textbf{let } main = (add5\ 2)^{m}$$

$Var \cup Lab$	L	notes
add	$\lambda x.\ \lambda y.\ x + y$	
x	5	when analyzing first call
$add5$		
x		
$add6$		
$main$		

k-CFA and m-CFA

- Context-sensitive version of 0-CFA
- Analyze each program point with some call string $\delta \in \Delta$
- Limit analysis depth to constant *k* (or *m*)
- We'll get to k-CFA vs. m-CFA later, but for now they are similar

$$
\begin{array}{rcl}
\sigma & \in & (Var \cup Lab) \times \Delta \to L \\
\Delta & = & Lab^{n \leqslant m} \\
L & = & \mathbb{Z} + \top + \mathcal{P}((\lambda x.e, \delta))
\end{array}
$$

m-CFA

$$\frac{}{\delta \vdash [\![n]\!]^l \hookrightarrow \alpha(n) \sqsubseteq \sigma(l, \delta)}\ \textit{const} \qquad \frac{}{\delta \vdash [\![x]\!]^l \hookrightarrow \sigma(x, \delta) \sqsubseteq \sigma(l, \delta)}\ \textit{var}$$

$$\frac{}{\delta \vdash [\![\lambda x.e^{l_0}]\!]^l \hookrightarrow \{(\lambda x.e, \delta)\} \sqsubseteq \sigma(l, \delta)}\ \textit{lambda}$$

$$\frac{\delta \vdash [\![e_1]\!]^{l_1} \hookrightarrow C_1 \qquad \delta \vdash [\![e_2]\!]^{l_2} \hookrightarrow C_2}{\delta \vdash [\![e_1^{l_1}\ e_2^{l_2}]\!]^l \hookrightarrow C_1 \cup C_2 \cup \mathbf{fn}_\delta\ l_1 : l_2 \Rightarrow l}\ \textit{apply}$$

m-CFA

$$\frac{\begin{array}{c}(\lambda x.e_0^{l_0}, \delta) \in \sigma(l_1, \delta) \qquad \delta' = \mathit{suffix}(\delta{+}{+}l, m) \\ C_1 = \sigma(l_2, \delta) \sqsubseteq \sigma(x, \delta') \wedge \sigma(l_0, \delta') \sqsubseteq \sigma(l, \delta) \\ C_2 = \{\sigma(y, \delta) \sqsubseteq \sigma(y, \delta') \mid y \in FV(\lambda x.e_0)\} \\ \delta' \vdash [\![e_0]\!]^{l_0} \hookrightarrow C_3\end{array}}{\mathbf{fn}_\delta\ l_1 : l_2 \Rightarrow l \hookrightarrow C_1 \cup C_2 \cup C_3}\ \textit{function-flow-}\delta$$

$$\frac{}{\delta \vdash [\![\lambda x.e^{l_0}]\!]^l \hookrightarrow \{(\lambda x.e, \delta)\} \sqsubseteq \sigma(l, \delta)}\ \textit{lambda}$$

$$\frac{\delta \vdash [\![e_1]\!]^{l_1} \hookrightarrow C_1 \qquad \delta \vdash [\![e_2]\!]^{l_2} \hookrightarrow C_2}{\delta \vdash [\![e_1^{l_1}\ e_2^{l_2}]\!]^l \hookrightarrow C_1 \cup C_2 \cup \mathbf{fn}_\delta\ l_1 : l_2 \Rightarrow l}\ \textit{apply}$$

m-CFA

$$\textbf{let } add = \lambda x.\ \lambda y.\ x + y$$
$$\textbf{let } add5 = (add\ 5)^{a5}$$
$$\textbf{let } add6 = (add\ 6)^{a6}$$
$$\textbf{let } main = (add5\ 2)^{m}$$

Var / *Lab*, δ	L	notes
add, $\bullet$		
x, a5		
add5, $\bullet$		
x, a6		
add6, $\bullet$		
main, $\bullet$		

m-CFA vs. k-CFA

- Original formulation of k-CFA by Olin Shivers in 1988
  - Call strings AND variable capture are both context-sensitive
  - Very expensive; proved to be EXPTIME by Van Horn & Marison in 2008
- But k-context-sensitive seems to work in polynomial time in OOP!
- Paradox explored by Might, Smaragdakis, and Van Horn in 2010
  - m-CFA defined as the polynomial counterpart to the OOP formulation of k-context-sensitivity
  - Runs in polynomial time (see text for details)