Control flow analysis for functional languages

Jonathan Lam

10/10/21

Simplifying assumptions in previous analyses: easy to tell what function was being called, more difficult when there is dynamic dispatch or in functional languages

1 A simple, labeled, functional language

- Grammar for a simple language:

$$e \in Expressions \tag{1}$$
$$t \in Term \tag{2}$$
$$l \in \mathcal{L} \tag{3}$$
$$\tag{4}$$
$$e ::= t^l \tag{5}$$
$$t ::= \lambda x.e \mid x \mid (e_1)(e_2) \mid \text{let } x = e_1 \text{ in } e_2 \tag{6}$$
$$\mid \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \mid n \mid e_1 + e_2 \mid \dots \tag{7}$$

- Each expression is labeled (i.e., an **expression** is a labeled term, or a **term** is an unlabeled expression)
  - This is useful for keeping track of analysis information (similar to program points in imperative analysis)

2 Simple control flow analysis

- What is a program point? We have nested expressions rather than successors and predecessors.
- Functions are first-class and may be passed around as variables, making it difficult to know which function is being called where.

- **Control flow analysis** (CFA) "seeks to statically determine which functions could be associated with which variables. Because functional languages are not based on statements but rather expressions, it is appropriate to reason about both the values of variables and the values expressions evaluate to"
- CFA and DFA are forms of **abstract interpretation**, an overall framework of sound approximation of program semantics; at a high level, abstract interpretation associates **labels** with **properties** by manipulating sets of states using monotonic functions over ordered sets defined by lattices
  - Analysis information σ maps each variable and label to a lattice value

2.1 0-CFA

- Simplest form of CFA, context-insensitive
- Track analysis information for variables and labels
- Abstract domain:

$$\sigma \in (Var \cup \mathcal{L}) \to L \tag{8}$$

$$L = \top + \mathcal{P}(\lambda x.e) \tag{9}$$

  i.e., we perform the analysis on each label or variable, and the abstract values are all of the possible set of functions that a variable can represent

- 0-CFA is a **constraint-based analysis**: it is defined via inference rules that generate constraints over the possible dataflow values for each variable or labeled location; those constraints are then solved
  - The judgment $[e]^l \hookrightarrow C$ can be read "the analysis of expression e with label l generates constraints C over dataflow state σ"

– Rules:

$$\frac{}{[x]^l \hookrightarrow \sigma(x) \sqsubseteq \sigma(l)}\ var \tag{10}$$

$$\frac{[e]^{l_0} \hookrightarrow C}{[\lambda x.e^{l_0}]^l \hookrightarrow \{\lambda x.e\} \sqsubseteq \sigma(l) \cup C}\ lambda \tag{11}$$

$$\frac{[e_1]^{l_1} \hookrightarrow C_1 \qquad [e_2]^{l_2} \hookrightarrow C_2}{[e_1^{l_1}\ e_2^{l_2}]^l \hookrightarrow C_1 \cup C_2 \cup \textbf{fn}\ l_1 : l_2 \Rightarrow l}\ apply \tag{12}$$

$$\frac{\lambda x.e_0^{l_0} \in \sigma(l_1)}{\textbf{fn}\ l_1 : l_2 \Rightarrow l \hookrightarrow \sigma(l_2) \sqsubseteq (x) \wedge \sigma(l_0) \sqsubseteq \sigma(l)}\ function\text{-}flow \tag{13}$$

* The notation **fn** $l_1 : l_2 \Rightarrow l$ is called a **function flow constraint**, which generates additional constraints according to the fourth rule

3 Questions

- What is PPA? (50, footnote)