Context-Sensitive Demand-Driven Control-Flow Analysis

Tim Whiting[0000−0003−4016−1071] and Kimball Germane[0000−0003−4903−5645]

Brigham Young University, Provo UT 84601, USA
tim@whitings.org
kimball@cs.byu.edu

Abstract. By decoupling and decomposing control flows, demand control-flow analysis (CFA) resolves only the flow segments determined necessary to produce a specified control-flow fact. It therefore presents a more flexible interface and pricing model than typical CFA, making many useful applications practical. At present, the only realization of demand CFA is the context-insensitive Demand 0CFA. Typical mechanisms for adding context sensitivity are not compatible with the demand setting because the analyzer is dispatched at arbitrary program points in indeterminate contexts. We overcome this challenge by identifying a context suitable for a demand analysis and designing a representation thereof that allows it to model incomplete knowledge of the context. On top of this design, we construct Demand m-CFA, a context-sensitive demand CFA hierarchy. With the attractive pricing model of demand analysis and the precision offered by context sensitivity, we show that Demand m-CFA can replace its exhaustive counterpart in compiler backends and integrate into interactive tools such as language servers.

Keywords: Demand CFA, m-CFA, Context-Sensitivity, Control Flow Analysis

1 Demand m-CFA Correctness

1.1 Demand ∞-CFA and Demand Evaluation

To show that Demand m-CFA is sound with respect to a standard call-by-value (CBV) semantics, we consider the limit of the hierarchy, Demand ∞-CFA, in which context lengths are unbounded. From here, we bridge Demand ∞-CFA to a CBV semantics with a concrete form of demand analysis called *Demand Evaluation*. Our strategy will be to show that the Demand ∞-CFA semantics is equivalent to Demand Evaluation which itself is sound with respect to a standard CBV semantics.

Demand Evaluation is defined in terms of relations $\Downarrow^{de}_{eval}$, $\Rightarrow^{de}_{expr}$, and $\Downarrow^{de}_{call}$ which are counterpart to $\Downarrow^{m}_{eval}$, $\Rightarrow^{m}_{expr}$, and $\Downarrow^{m}_{call}$, respectively. Like their counterparts, $\Downarrow^{de}_{eval}$, $\Rightarrow^{de}_{expr}$, and $\Downarrow^{de}_{call}$ relate configurations to configurations. However, a Demand Evaluation configuration includes a store σ from addresses n to calls

consisting of a call site and its environment. Demand Evaluation environments, rather than being a sequence of contexts, are sequences of addresses. Like contexts, an address may denote an indeterminate context (i.e. call) which manifests as an address which is not mapped in the store. Formally, the components of stores and environments are defined

$$(s, n), \sigma \in \mathit{Store} = (\mathit{Addr} \to \mathit{Call}) \times \mathit{Addr} \qquad \rho \in \mathit{Env} = \mathit{Addr}^*$$
$$\mathit{cc} \in \mathit{Call} = \mathit{App} \times \mathit{Env} \qquad n \in \mathit{Addr} = \mathbb{N}$$

A store is a pair consisting of a map from addresses to calls and the next address to use; the initial store is $(\bot, 0)$.

Figure 1 presents the definitions of $\Downarrow^{de}_{eval}$, $\Rightarrow^{de}_{expr}$, and $\Downarrow^{de}_{call}$.

$$\Downarrow^{de}_{eval}, \Rightarrow^{de}_{expr}, \Downarrow^{de}_{call} \subseteq \mathsf{Cursor} \times \mathit{Env} \times \sigma \times \mathsf{Cursor} \times \mathit{Env} \times \sigma$$
$$\Rightarrow^{de}_{find} \subseteq \mathsf{Var} \times \mathsf{Cursor} \times \mathit{Env} \times \sigma \times \mathsf{Cursor} \times \mathit{Env} \times \sigma$$

Most rules are unchanged from Demand m-CFA modulo the addition of stores. Instantiation in Demand Evaluation is captured by creating a mapping in the store. For instance, Demand m-CFA's *App* rule "discovers" the caller of the entered call, which effects an instantiation via *App-Body-Instantiation*. In contrast, Demand Evaluation's *App* rule allocates a fresh address n using fresh, maps it to the caller in the store, and extends the environment of the body with it. The fresh metafunction extracts the unused address and returns a store with the next one. Store extension is simply lifted over the next unused address. Formally, they are defined as follows.

$$\mathsf{fresh}((s, n)) := (n, (s, n + 1)) \qquad (s, n)[n_0 \mapsto \mathit{cc}] := (s[n_0 \mapsto \mathit{cc}], n)$$

Unknown-Call applies when the address n is unmapped in the store. It instantiates the environment by mapping n with the discovered caller. *Known-Call* uses $\equiv_\sigma$ to ensure that the known and discovered environments are isomorphic in the store. The $\equiv_\sigma$ relation is defined on addresses and lifted elementwise to environments. We have $n_0 \equiv_\sigma n_1$ iff $\sigma(n_0) = \bot = \sigma(n_1)$ or $\sigma(n_0) = (C[(e_0\, e_1)], \rho_0)$, $\sigma(n_1) = (C[(e_0\, e_1)], \rho_1)$, and $\rho_0 \equiv_\sigma \rho_1$. If the environments are isomorphic, then all instances of the known environment are substituted with the discovered environment in the store, ensuring that queries in terms of the known are kept up to date. This rule corresponds directly to the instantiation relation of Demand m-CFA.

1.2 Demand Evaluation Equivalence

In order to show a correspondence between Demand ∞-CFA and Demand Evaluation, we establish a correspondence between the environments of the former and the environment–store pairs of the latter, captured by the judgment $\hat{\rho}\, \hat{\rho} \Leftrightarrow_\rho \rho, \sigma$

$$\textsc{Lam}\ \frac{}{C[\lambda x.e], \rho, \sigma \Downarrow^{de}_{eval} C[\lambda x.e], \rho, \sigma} \qquad \textsc{Operator}\ \frac{}{C[([e_0]\, e_1)], \rho, \sigma \Rightarrow^{de}_{expr} C[(e_0\, e_1)], \rho, \sigma}$$

$$\textsc{Ref}\ \frac{(C_x[e_x], \rho_x) = \mathsf{bind}(x, C[x], \rho) \quad C_x[e_x], \rho_x, \sigma_0 \Downarrow^{de}_{call} C'[(e_0\, e_1)], \rho', \sigma_1 \qquad C'[(e_0\, [e_1])], \rho', \sigma_1 \Downarrow^{de}_{eval} C_v[\lambda x.e], \rho_v, \sigma_2}{C[x], \rho, \sigma_0 \Downarrow^{de}_{eval} C_v[\lambda x.e], \rho_v, \sigma_2}$$

$$\textsc{App}\ \frac{(n, \sigma_2) := \mathsf{fresh}(\sigma_1) \quad \begin{array}{c} C[([e_0]\, e_1)], \rho, \sigma_0 \Downarrow^{de}_{eval} C'[\lambda x.e], \rho', \sigma_1 \\ C'[\lambda x.[e]], n :: \rho', \sigma_2[n \mapsto (C[(e_0\, e_1)], \rho)] \Downarrow^{de}_{eval} C_v[\lambda x.e_v], \rho_v, \sigma_3 \end{array}}{C[(e_0\, e_1)], \rho, \sigma_0 \Downarrow^{de}_{eval} C_v[\lambda x.e_v], \rho_v, \sigma_3}$$

$$\textsc{Operand}\ \frac{(n, \sigma_2) := \mathsf{fresh}(\sigma_1) \quad \begin{array}{c} C[([e_0]\, e_1)], \rho, \sigma_0 \Downarrow^{de}_{eval} C'[\lambda x.e], \rho', \sigma_1 \\ x, C'[\lambda x.[e]], n :: \rho', \sigma_2[n \mapsto (C[(e_0\, e_1)], \rho)] \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_3 \\ C_x[x], \rho_x, \sigma_3 \Rightarrow^{de}_{expr} C''[(e_2\, e_3)], \rho'', \sigma_4 \end{array}}{C[(e_0\, [e_1])], \rho, \sigma_0 \Rightarrow^{de}_{expr} C''[(e_2\, e_3)], \rho'', \sigma_4}$$

$$\textsc{Body}\ \frac{C[\lambda x.[e]], \rho, \sigma_0 \Downarrow^{de}_{call} C'[(e_0\, e_1)], \rho', \sigma_1 \qquad C'[(e_0\, e_1)], \rho', \sigma_1 \Rightarrow^{de}_{expr} C''[(e_2\, e_3)], \rho'', \sigma_2}{C[\lambda x.[e]], \rho, \sigma_0 \Rightarrow^{de}_{expr} C''[(e_2\, e_3)], \rho'', \sigma_2}$$

$$\textsc{Find-Ref}\ \frac{}{x, C[x], \rho, \sigma \Rightarrow^{de}_{find} C[x], \rho, \sigma} \qquad \textsc{Find-Operator}\ \frac{x, C[([e_0]\, e_1)], \rho, \sigma_0 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_1}{x, C[(e_0\, e_1)], \rho, \sigma_0 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_1}$$

$$\textsc{Find-Operand}\ \frac{x, C[(e_0\, [e_1])], \rho, \sigma_0 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_1}{x, C[(e_0\, e_1)], \rho, \sigma_0 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_1}$$

$$\textsc{Find-Body}\ \frac{x \neq y \qquad (n, \sigma_1) := \mathsf{fresh}(\sigma_0) \qquad x, C[\lambda y.[e]], n :: \rho, \sigma_1 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_2}{x, C[\lambda y.e], \rho, \sigma_0 \Rightarrow^{de}_{find} C_x[x], \rho_x, \sigma_2}$$

$$\textsc{Unknown-Call}\ \frac{\sigma_0(n) = \bot \qquad C[\lambda x.e], \rho, \sigma_0 \Rightarrow^{de}_{expr} C'[(e_0\, e_1)], \rho', \sigma_1 \qquad \sigma_2 := \sigma_1[n \mapsto (C[(e_0\, e_1)], \rho')]}{C[\lambda x.[e]], n :: \rho, \sigma_0 \Downarrow^{de}_{call} C'[(e_0\, e_1)], \rho', \sigma_2}$$

$$\textsc{Known-Call}\ \frac{\sigma_0(n) = (C[(e_0\, e_1)], \rho') \quad C[\lambda x.e], \rho, \sigma_0 \Rightarrow^{de}_{expr} C'[(e_0\, e_1)], \rho'', \sigma_1 \qquad \rho' \equiv_{\sigma_1} \rho'' \qquad \sigma_2 := \sigma_1[\rho''/\rho']}{C[\lambda x.[e]], n :: \rho, \sigma_0 \Downarrow^{de}_{call} C'[(e_0\, e_1)], \rho'', \sigma_2}$$

Fig. 1: Demand Evaluation

defined by the following rules.

$$\frac{\hat{cc}_1 \;_{\hat{cc}} \Leftrightarrow_n n_1, \sigma \qquad \ldots \qquad \hat{cc}_k \;_{\hat{cc}} \Leftrightarrow_n n_k, \sigma}{\langle \hat{cc}_1, \ldots, \hat{cc}_k \rangle \;_{\hat{\rho}} \Leftrightarrow_\rho \langle n_1, \ldots, n_k \rangle, \sigma} \qquad \frac{}{\langle\rangle \;_{\hat{cc}} \Leftrightarrow_\rho \langle\rangle, \sigma}$$

$$\frac{C[(e_0\, e_1)] :: \hat{cc} \;_{\hat{cc}} \Leftrightarrow_n n, \sigma}{C[(e_0\, e_1)] :: \hat{cc} \;_{\hat{cc}} \Leftrightarrow_\rho n :: \rho, \sigma} \qquad \frac{\sigma(n) = \bot}{?_x \;_{\hat{cc}} \Leftrightarrow_n n, \sigma}$$

$$\frac{\sigma(n) = (C[(e_0\, e_1)], \rho) \qquad \hat{cc} \;_{\hat{cc}} \Leftrightarrow_\rho \rho, \sigma}{C[(e_0\, e_1)] :: \hat{cc} \;_{\hat{cc}} \Leftrightarrow_n n, \sigma}$$

This judgment ensures that each context in the Demand ∞-CFA environment matches precisely with the corresponding address with respect to the store: if the context is indeterminate, the address must not be mapped in the store; otherwise, if the heads of the context are the same, the relation recurs.

Now it is straightforward to express the equivalence between the Demand ∞-CFA relations and Demand Evaluation.

Theorem 1 (Evaluation Equivalence). *If* $\hat{\rho}_0 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_0, \sigma_0$ *then* $C[e], \hat{\rho}_0 \Downarrow^{\infty}_{eval} C'[\lambda x.e], \hat{\rho}_1$ *iff* $C[e], \rho_0, \sigma_0 \Downarrow^{de}_{eval} C'[\lambda x.e], \rho_1, \sigma_1$ *where* $\hat{\rho}_1 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_1, \sigma_1$.

Theorem 2 (Trace Equivalence). *If* $\hat{\rho}_0 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_0, \sigma_0$ *then* $C[e], \hat{\rho}_0 \Rightarrow^{\infty}_{expr} C'[(e_0\, e_1)], \hat{\rho}_1$ *iff* $C[e], \rho_0, \sigma_0 \Rightarrow^{de}_{expr} C'[(e_0\, e_1)], \rho_1, \sigma_1$ *where* $\hat{\rho}_1 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_1, \sigma_1$.

Theorem 3 (Caller Equivalence). *If* $\hat{\rho}_0 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_0, \sigma_0$ *then* $C[e], \hat{\rho}_0 \Downarrow^{\infty}_{call} C'[(e_0\, e_1)], \hat{\rho}_1$ *iff* $C[e], \rho_0, \sigma_0 \Downarrow^{de}_{call} C'[(e_0\, e_1)], \rho_1, \sigma_1$ *where* $\hat{\rho}_1 \;_{\hat{\rho}} \Leftrightarrow_\rho \rho_1, \sigma_1$.

These theorems are proved by induction on the derivations, corresponding instantiation of environments on the Demand ∞-CFA side with mapping an address on the Demand Evaluation side.

2 Detailed Precision Results

Figure 2 shows the number of singleton flow sets found by Demand m-CFA for each program individually. As can be seen, the majority of programs reach the corresponding exhaustive m-CFA results at low effort. Notably, increasing m doesn't drastically increase the cost. This demonstrates that, due to its cost model, Demand m-CFA can run at much higher levels of m than is practical in exhaustive analyses, obtaining more precise results. On `primtest`, and to a lesser degree `rsa`, Demand m-CFA issues queries on pieces of dead code, resulting in additional singleton flow sets. Additionally, due to the reachability assumption explained previously in the results section, we see precision loss in cases like blur We plan to investigate ways to overcome these limitations in future work.

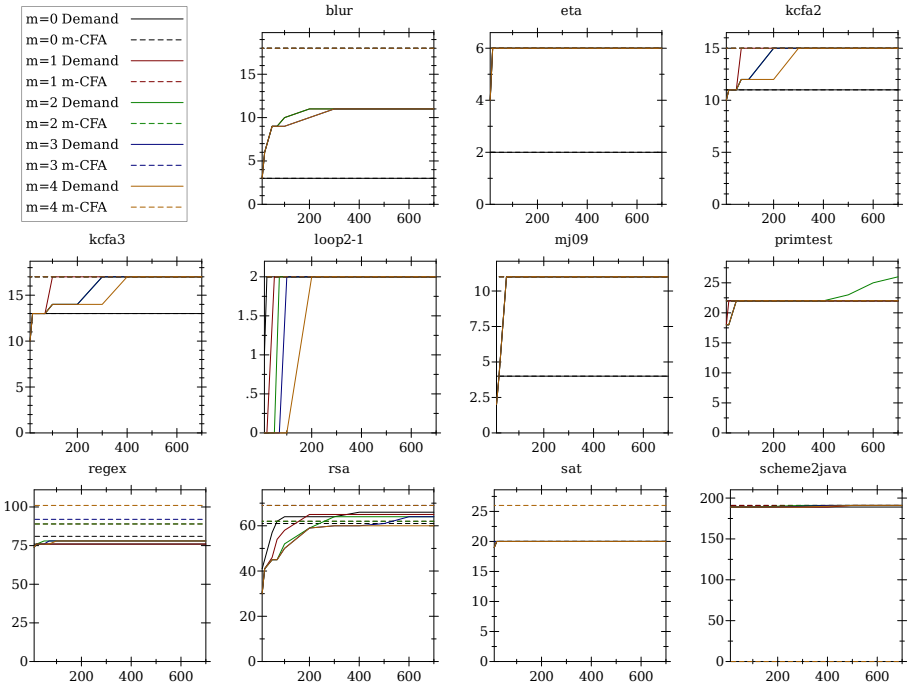

Fig. 2: The number of singleton flow sets (y-axis) found by a Demand m-CFA analysis given gas allocated per query (x-axis). Dashed lines represent the baseline number of singleton flow sets found by an exhaustive exponential m-CFA analysis with a 10 minute timeout. `scheme2java` does not have results for $m >= 2$ exhaustive m-CFA due to timing out.