IDEal: Efficient and Precise Alias-aware Dataflow Analysis

In this paper, authors presented IDEal, an extension to IDE framework to handle aliases in a precise and efficient manner. IDEal has the following features.

- Sound strong updates
- Individual propagation of pointers

IDEal can reuse fine-grained procedure summaries, which contributes to a considerable improvement in efficiency. It relieves the burden of alias information encoding in the dataflow domain, and provides an accessible approach to implement a wide range of dataflow analyses that track object flows.

Overview

The workflow of IDEal consists of two consecutive phases: the object-flow propagation(Phase OF) and the value-flow propagation(Phase VF). Meanwhile, IDEal takes the advantages of typestate analysis by defining a concrete lattice L and the respective environment transformers.

Design

The details of the main algorithm in IDEal include the standard flow functions definition in IDEal, how to handle points of aliasing, how to perform sound strong updates, and how to support context-sensitive alias queries. The following figure illustrates the definitions of the intraprocedural normal-flow functions of IDEal.

$$[\![x \leftarrow new]\!](\langle\!\langle v, sEt \rangle\!\rangle) = \begin{cases} \varnothing & \text{if } v = x \\ \{\langle\!\langle v, sEt \rangle\!\rangle\} \end{cases} \tag{1}$$

$$[\![x \leftarrow y]\!](\langle\!\langle v, sEt \rangle\!\rangle) = \begin{cases} \{\langle\!\langle x, sEt \rangle\!\rangle, \langle\!\langle v, sEt \rangle\!\rangle\} & \text{if } v = y \\ \varnothing & \text{if } v = x \\ \{\langle\!\langle v, sEt \rangle\!\rangle\} \end{cases} \tag{2}$$

$$[\![x \leftarrow y.f]\!](\langle\!\langle v, sEt \rangle\!\rangle) = \begin{cases} \{\langle\!\langle v, sEt \rangle\!\rangle\} \cup \{\langle\!\langle x, G \rangle\!\rangle \mid G \in tail(sEt)\} & \text{if } v = y \wedge s = f \\ \varnothing & \text{if } v = x \\ \{\langle\!\langle v, sEt \rangle\!\rangle\} \end{cases} \tag{3}$$

$$[\![x.f \leftarrow y]\!](\langle\!\langle v, sEt \rangle\!\rangle) = \begin{cases} \{\langle\!\langle v, sEt \rangle\!\rangle\} \cup \langle\!\langle x, cons(f, sEt) \rangle\!\rangle & \text{if } v = y \\ \varnothing & \text{if } v = x \wedge s = f \\ \{\langle\!\langle v, sEt \rangle\!\rangle\} \end{cases} \tag{4}$$

The fixed-point iteration of IDE is controlled by a function defined in the following figure. In the function PROPAGATE, two changes are introduced. A node of the OFG can be a point of aliasing, and the necessary aliases are computed by a pointer analysis. The call of COMPUTEALIASES abstracts the construction. The second change affects the propagation during Phase VF. Some of the points of aliasing introduce strong updates.

```
1:  procedure PROPAGATE(d1 → ⟨s, d2⟩ ,f)
2:      if isPointOfAliasing(⟨s, d2⟩) then
3:          aliases = COMPUTEALIASES(d1 → ⟨s, d2⟩)
4:          for d3 ∈ aliases ∧ d3 ≠ d2 do
5:              PROPAGATE(d1 → ⟨s, d3⟩,f)                ▷ Indirect flows through aliases
6:          end for
7:      end if
8:      if Phase VF ∧ RECEIVESSTRONGUPDATE(d1 → ⟨s, d2⟩) then
9:          return                                       ▷ Strong update in Phase VF kills flow.
10:     end if
11:     f′ = f ⊓ JumpFn(d1 → ⟨s, d2⟩)
12:     if f′ ≠ JumpFn(d1 → ⟨s, d2⟩) then
13:         JumpFn(d1 → ⟨s, d2⟩) = f′
14:         PathWorkList.add(d1 → ⟨s, d2⟩)
15:     end if
16: end procedure
```

Strong updates can only be performed in Phase VF, once all necessary alias queries have already been performed in Phase OF and the results are known.