Concrete and Abstract Interpretation: Better Together

Maria Jenkins
University of Utah
mjenkins@eng.utah.edu

Leif Andersen
Northeastern University
leif@leifandersen.net

Thomas Gilray, Matthew Might
University of Utah
{tgilray, might}@cs.utah.edu

Abstract

Recent work in abstracting abstract machines provides a methodology for deriving sound static analyzers from a concrete semantics by way of abstract interpretation. Consequently, the concrete and abstract semantics are closely related by design. We apply Galois-unions as a framework for combining both concrete and abstract semantics, and explore the benefits of being able to express both in a single semantics. We present a methodology for creating such a unified representation using operational semantics and implement our approach with and A-normal form (ANF) λ-calculus for a CESK style machine in PLT Redex.

1. Introduction

Static analyses aim to reason about the behavior of programs before run-time and have numerous applications including compiler optimization, malware detection, and program verification. Abstract interpretation is a highly general approach to static analysis which produces interpreters which use an approximate *abstract* semantics for evaluation, instead of a fully precise semantics. Accepting imprecision allows analysis designers to ensure termination, and indeed a reasonable bound on the complexity of their analyses. Current techniques in this form of static analyses requires the implementation of two separate interpreters. As we will show concrete semantics and abstract semantics frustratingly resemble one another, leading to large tracts of almost duplicated code in the implementations.

In the Cousots' foundational work on abstract interpretation, they note that the concrete semantics of a language is also a static analysis of that language, albeit an incomputable one [3][4]. We exploit this fact to produce a unified representation of concrete and abstract interpreters we call a *Galois union*. We will show that it is possible to systematically enhance an abstract interpretation calculated from a *Galois connection* (a formal relationship between concrete and abstract semantics) to form this union. For the cost of a static analyzer, one gets an interpreter for free.

We further show how to apply this unified framework for switching between the concrete and abstract semantics mid-analysis. This allows an analysis to use concrete evaluation during the initialiation phase of a program so that top-level definitions are evaluated precisely before switching to an approximate semantics which ensures termination.

We elucidate a number of additional benefits of our unified representation of concrete and abstract semantics:

- It saves engineering time and code by removing the necessity of building a separate interpreter.
- This in-turn promotes maintainability of the code base and improves robustness because testing the interpreter simultaneously tests the analyzer and vice versa.
- It provides a unified framework for combining static and dynamic analysis.

1.1 Contributions

We make the following contributions:

1. Galois unions: A theory for unifying analyses and interpreters.
2. The extraction of a CPS interpreter from k-CFA [11, 12].

In the following section we examine the similarity between a concrete and abstract semantics for the λ-calculus in continuation-passing-style (CPS). Section 3 presents a review of the theory for producing sound static analyses using Galois connections and in section 4 we present a unified theory of Galois unions and how they may be derived automatically from a Galois connection. In section 5 we provide a case study of the CPS-λ-calculus , in section 6 we dis-

SFP '14, 11 19, 2014, Washington D.C, USA.
Copyright © 2014 ACM 978-1-nnnn-nnnn-n/14/11...$15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

cuss the implementation, and in section 7 we discuss related work.

2. Semantics of CPS

To demonstrate the similarity of concrete and abstract semantics it is instructive to define a language, show the corresponding machine that interprets it, and show the abstract machine that analyzes it. We do this for the pure λ-calculus in *continuation-passing-style* (CPS). This language only permits call-sites in tail-position, so continuations must be reified as a call-back function to be invoked on the result.[1]

$$\begin{array}{rcll} e \in \mathsf{Exp} &=& \mathsf{Lam} + \mathsf{Var} & \text{[expressions]} \\ v \in \mathsf{Var} &=& \langle\text{variables}\rangle & \text{[variables]} \\ lam \in \mathsf{Lam} &::=& \lambda v_1 \ldots v_n . call & [\lambda\text{-terms}] \\ call \in \mathsf{Call} &::=& e_0\ e_1 \ldots e_n & \text{[function application]} \end{array}$$

2.1 Concrete and Abstract State Space

Below is the concrete machine. We define a concrete state-space for CPS λ-calculus:

$$\begin{array}{rl} \varsigma \in \Sigma & = \mathsf{Call} \times \mathit{Env} \times \mathit{Store} \\ \rho \in \mathit{Env} & = \mathsf{Var} \rightharpoonup \mathit{Addr} \\ \sigma \in \mathit{Store} & = \mathit{Addr} \rightharpoonup D \\ d \in D & = \mathit{Clo} \\ clo \in \mathit{Clo} & = \mathsf{Lam} \times \mathit{Env} \\ a \in \mathit{Addr} & = \langle\text{an } \textit{infinite} \text{ set of addresses}\rangle, \end{array}$$

and an abstract state-space:

$$\begin{array}{rl} \hat{\varsigma} \in \hat{\Sigma} & = \mathsf{Call}_{\bot}^{\top} \times \widehat{\mathit{Env}} \times \widehat{\mathit{Store}} \\ \hat{\rho} \in \widehat{\mathit{Env}} & = \mathsf{Var} \rightharpoonup \widehat{\mathit{Addr}} \\ \hat{\sigma} \in \widehat{\mathit{Store}} & = \widehat{\mathit{Addr}} \to \widehat{D} \\ \hat{d} \in \hat{D} & = \widehat{\mathit{Clo}} \\ \widehat{clo} \in \widehat{\mathit{Clo}} & = \mathsf{Lam}_{\bot}^{\top} \times \widehat{\mathit{Env}} \\ \hat{a} \in \widehat{\mathit{Addr}} & = \langle\text{a } \textit{finite} \text{ set of addresses}\rangle. \end{array}$$

The concrete and abstract state-spaces look very similar. They differ slightly in their stores. The concrete semantics has an infinite set of addresses and it maps an address to a closure. The abstract semantics obtains a finite state-space by bounding the number of addresses in the store. For this reason, multiple closures may share an address, so flow-sets of possible closures are indicated by each address.

2.2 Concrete and Abstract Semantics

The transfer function $f : \Sigma \to \Sigma$ describes the concrete semantics:

$$([\![(f\ æ_1\ \ldots\ æ_n)]\!], \rho, \sigma) \Rightarrow (ce, \rho'', \sigma')$$

$$\begin{array}{l} \text{where } ([\![(\lambda\ (v_1 \ldots v_n)\ ce)]\!], \rho') = \mathcal{A}(f, \rho, \sigma) \\ \qquad d_i = \mathcal{A}(æ_i, \rho, \sigma) \\ \qquad a_i = alloc(x_i,\ \varsigma) \\ \qquad \rho'' = \rho'[v_i \mapsto a_i] \\ \qquad \sigma' = \sigma[a_i \mapsto d_i] \end{array}$$

where the function $\mathcal{A} : \mathsf{Exp} \times \mathit{Env} \times \mathit{Store} \to D$ is the argument evaluator:

$$\begin{array}{r} \mathcal{A}(v, \rho, \sigma) = \sigma(\rho(v)) \\ \mathcal{A}(lam, \rho, \sigma) = (lam, \rho), \end{array}$$

and the allocator $alloc : \mathsf{Var} \times \Sigma \to \mathit{Addr}$ allocates a fresh address.

A series of calculations (Appendix A.1) then finds a computable static analysis:

$$([\![(f\ æ_1\ \ldots\ æ_n)]\!], \hat{\rho}, \hat{\sigma}) \rightsquigarrow (ce, \hat{\rho}'', \hat{\sigma}')$$

$$\begin{array}{l} \text{where } ([\![(\lambda\ (v_1 \ldots v_n)\ ce)]\!], \hat{\rho}') = \hat{\mathcal{A}}(f, \hat{\rho}, \hat{\sigma}) \\ \qquad \hat{d}_i = \mathcal{A}(æ_i, \hat{\rho}, \hat{\sigma}) \\ \qquad \hat{a}_i = \widehat{alloc}(x_i,\ ) \\ \qquad \hat{\rho}'' = \rho'[\hat{v}_i \mapsto \hat{a}_i] \\ \qquad \hat{\sigma}' = \hat{\sigma} \sqcup [a_i \mapsto d_i] \end{array}$$

where the function $\hat{\mathcal{A}} : \mathsf{Exp} \times \widehat{\mathit{Env}} \times \widehat{\mathit{Store}} \to \hat{D}$ is the abstract evaluator:

$$\begin{array}{r} \hat{\mathcal{A}}(v, \hat{\rho}, \hat{\sigma}) = \hat{\sigma}(\hat{\rho}(v)) \\ \hat{\mathcal{A}}(lam, \hat{\rho}, \hat{\sigma}) = \{(lam, \hat{\rho})\}\,. \end{array}$$

Importantly, the abstract allocator produces a finite number of addresses: $\widehat{alloc} : \mathsf{Var} \times \hat{\Sigma} \to \hat{Addr}$.

2.3 Abstracting abstract-machines

The approach of abstracting a small-step abstract-machine semantics produces a clean correspondence between the interpreters because all unboundedness may be focused on a single machine component and then removed upon abstraction. [9] In the above example, this is done by limiting the address-space. For more complex abstract-machines with other sources of recursion, threading them through the address-space yields an approximation of these components automatically. Take for example an explicit stack of continuations; in an concrete-semantics an unbounded stack is required to ensure perfect precision. In an approximate semantics however, we may obtain a bounded stack by store-allocating continuation-frames as would be done implicitly in our CPS language. [8] In this way, the recursion of this stack is explicitly cut and made finite. With this style of abstraction, it is the only step necessary for introducing non-determinism into the semantics and for bounding the machine's state-space, leading to a computable over-approximation.

Comparing the two semantics it is evident the abstract semantics are isomorphic to the concrete semantics. It would be convenient to be able to unify the semantics and build one interpreter. Galois connections are the starting point of our unification through galois unions.

3. Galois Connections

For the purpose of self-containment, we review Galois connections and adjunctions as used in abstract interpretation. Readers already versed in Galois theory and adjunctions may wish to skim or skip this section. Informally, Galois connections and adjunctions are a generalization of isomorphism to partially ordered sets. That is, in a Galois connection, the two sets need not be locked into a one-to-one, structure-preserving correspondence; rather, a Galois connection ensures the existence of *order*-preserving maps between the sets.
Static analyses use Galois connections because a Galois connection determines the tightest projection of a function over one set, *e.g.*, the concrete transfer function, into another set. This projection is frequently interpretable as the optimal static analysis.[10]

3.1 Conventions

A function $f : X \to X$ is **order-preserving** or **monotonic** on poset $(X, \sqsubseteq)$ iff $x \sqsubseteq x'$ implies $f(x) \sqsubseteq f(x')$. The natural ordering of a function over posets is compared range-wise; that is:

$$f \sqsubseteq g \text{ iff } f(x) \sqsubseteq g(x) \text{ for all } x \in dom(X).$$

3.2 Review of Galois Connections

There are two kinds of Galois connections: monotone Galois connections and antitone Galois connections. Our work focuses on monotone Galois connections, since antitone Galois connections are rarely used in static analysis.[1] From this point forward, *Galois connection* refers to *monotone Galois connection*.

Definition 3.1. The 4-tuple $(X, \alpha, \gamma, \hat{X})$ is a **Galois connection** where:

- $(X, \sqsubseteq_X)$ is a partially ordered set;
- $(\hat{X}, \sqsubseteq_{\hat{X}})$ is a partially ordered set;
- $\alpha : X \to \hat{X}$ is a monotonic function; and
- $\gamma : \hat{X} \to X$ is a monotonic function;

such that:

$$\gamma \circ \alpha \sqsupseteq \lambda x.x \text{ and } \alpha \circ \gamma \sqsubseteq \lambda \hat{x}.\hat{x}. \qquad (3.1)$$

The proposition, "$(X, \alpha, \gamma, \hat{X})$ is a Galois connection," is denoted $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$.

In static analysis, the set X is the concrete space and the set $\hat{X}$ is the abstract space, while the functions α and γ are the abstraction and concretization maps.

The precise formulation of the Galois constraint on maps (3.1) is useful in proofs, but an equivalent (if less terse) formulation of it is more intelligible:

$$\gamma(\alpha(x)) \sqsupseteq x \text{ for all } x \in X \text{ and } \alpha(\gamma(\hat{x})) \sqsubseteq \hat{x} \text{ for all } \hat{x} \in \hat{X}.$$

[1] In fact, we cannot find a paper in static analysis outside of the Cousots' original 1979 paper [4] that makes use of antitone Galois connections.

Informally, $\gamma(\alpha(x)) \sqsupseteq x$ means that abstraction followed by concretization will not discard information, while $\alpha(\gamma(\hat{x})) \sqsubseteq \hat{x}$ means that concretization followed by abstraction may choose a more precise representative. Ordinarily, the second constraint is strengthened to equality, so that:

$$\alpha(\gamma(\hat{x})) = \hat{x},$$

in which case, we are dealing with a **Galois insertion**. In a Galois insertion, there is only one abstract representative for each concrete element, however any given abstract element may have one or more concrete representatives. In most abstract interpretations, the Galois connection is a Galois insertion.

3.3 Adjunctions

It is often useful to cast a Galois connection as its equivalent adjunction.

Definition 3.2. An **adjunction** is a 4-tuple $(X, \alpha, \gamma, \hat{X})$ where:

- $(X, \sqsubseteq_X)$ is a partially ordered set;
- $(\hat{X}, \sqsubseteq_{\hat{X}})$ is a partially ordered set;
- $\alpha : X \to \hat{X}$ is a monotonic function; and
- $\gamma : \hat{X} \to X$ is a monotonic function;

such that:

$$\alpha(x) \sqsubseteq \hat{x} \text{ iff } x \sqsubseteq \gamma(\hat{x}).$$

Theorem 3.1. *$(X, \alpha, \gamma, \hat{X})$ is a Galois connection iff it is an adjunction[10].*

3.4 Calculating the optimal analysis from a Galois connection

Galois connections allow the optimal static analysis to be *calculated* from a concrete semantics. If $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$ and the function $f : X \to X$ is monotonic, then the **projection** of the function f into the poset $\hat{X}$ is the function $\hat{f} = \alpha \circ f \circ \gamma$. In effect, the function $\hat{f}$ concretizes its input, runs the concrete function, and then re-abstracts the output. If the function f is a concrete semantics, then the projection $\hat{f}$ is its **optimal static analysis** (or the **abstract semantics**).

It is not necessary to prove a calculated analysis correct, because all calculated analyses obeys the expected simulation theorem:

Theorem 3.2. *If $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$ and the function $f : X \to X$ is monotonic, then the function $\hat{f} = \alpha \circ f \circ \gamma$ simulates the function f; that is, if:*

$$\alpha(x) \sqsubseteq \hat{x},$$

then:

$$\alpha(f(x)) \sqsubseteq \hat{f}(\hat{x}).$$

Proof. Assume $\alpha(x) \sqsubseteq \hat{x}$.

$$\begin{aligned}
\hat{f}(\hat{x}) &= (\alpha \circ f \circ \gamma)(\hat{x}) \\
&= (\alpha \circ f)(\gamma(\hat{x})) \\
&\sqsupseteq (\alpha \circ f)(x) && \text{by monotonicity of } (\alpha \circ f) \\
& && \text{and } \gamma(\hat{x}) \sqsupseteq x \\
&= \alpha(f(x)).
\end{aligned}$$

□

In fact, given an optimal analysis $\hat{f}$, any function $\hat{f}'$ such that $\hat{f}' \sqsupseteq \hat{f}$ is also a sound simulation of the concrete function f.

4. Galois unions

The Galois union of a Galois connection provides a common space in which to express both the concrete and abstract semantics. Given a Galois connection $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$, a Galois union consists of a third poset $\tilde{X}$—the union space—and two more Galois connections: a concrete-union connection, $X \underset{\mu}{\overset{\nu}{\leftrightarrows}} \tilde{X}$, and an abstract-union connection, $\hat{X} \underset{\kappa}{\overset{\eta}{\leftrightarrows}} \tilde{X}$:

$$\begin{tikzcd}
& \tilde{X} \arrow[dl, shift left, "\nu"] \arrow[dr, shift left, "\eta"] & \\
X \arrow[ur, shift left, "\mu"] \arrow[rr, shift left, "\alpha"] & & \hat{X} \arrow[ul, shift left, "\kappa"] \arrow[ll, shift left, "\gamma"]
\end{tikzcd}$$

The newly introduced Galois connections are constrained so that the projection of the concrete semantics into the union space remains equivalent to the concrete semantics, while the projection of the optimal analysis remains equivalent to the optimal analysis:

Definition 4.1. The structure $(\tilde{X}, \mu, \nu, \kappa, \eta)$ is a **Galois union** with respect to the Galois connection $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$ iff $X \underset{\mu}{\overset{\nu}{\leftrightarrows}} \hat{X}$ and $\tilde{X} \underset{\eta}{\overset{\kappa}{\leftrightarrows}} \hat{X}$, and:

$$\mu \circ \nu = \lambda \tilde{x}.\tilde{x} \tag{4.1}$$

$$\nu \circ \mu = \lambda x.x \tag{4.2}$$

$$\eta \circ \kappa = \lambda \hat{x}.\hat{x} \tag{4.3}$$

$$\eta = \alpha \circ \nu. \tag{4.4}$$

Informally, constraints (4.1) and (4.2) indicate that the union space $\tilde{X}$ is actually isomorphic to the concrete space; we can move between them with absolutely no loss of precision or information. Constraint (4.3) means that we can inject from the abstract space into the union space with no loss of precision, while constraint (4.4) indicates that the abstraction map from the union space to the abstract space is isomorphic to the abstraction map from the concrete space to the abstract space.

It is irrelevant how one decides to construct the Galois union of a Galois connection, because all such unions are structurally identical to one another:

Theorem 4.1. *All Galois unions of a Galois connection $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$ are equivalent up to an order-preserving isomorphism.*

Proof. Let $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$ be a Galois insertion. Let $(\tilde{X}, \mu, \nu, \kappa, \eta)$ and $(\tilde{X}', \mu', \nu', \kappa', \eta')$ be two Galois unions. We shall construct order-preserving maps, f and f', between these two unions, and then show that these maps are inverses to each other. Define the functions $f : \tilde{X} \to \tilde{X}'$ and $f' : \tilde{X}' \to \tilde{X}$ so that:

$$\begin{aligned}
f &= \mu' \circ \nu \\
f' &= \mu \circ \nu'.
\end{aligned}$$

Then, observe:

$$\begin{aligned}
f \circ f' &= (\mu' \circ \nu) \circ (\mu \circ \nu') \\
&= \mu' \circ (\nu \circ \mu) \circ \nu' \\
&= \mu' \circ \nu' \\
&= \lambda \tilde{x}'.\tilde{x}'.
\end{aligned}$$

An identical argument shows that $f' \circ f = \lambda \tilde{x}.\tilde{x}$. □

4.1 The natural Galois union

Given a Galois connection, we can construct a "natural" Galois union from its abstraction and concretization maps. Given a Galois connection $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$, find the set of concrete elements precisely represented in the abstract, P:

$$P = \{x : \gamma(\alpha(x)) = x\}.$$

The natural union space is then $\tilde{X} = \hat{X} + (X - P)$ with ordering $(\sqsubseteq_{\tilde{X}})$:

$$\begin{aligned}
&x \sqsubseteq_{\tilde{X}} x' \text{ iff } x \sqsubseteq_X x' \\
&x \sqsubseteq_{\tilde{X}} \hat{x}' \text{ iff } x \sqsubseteq_X \gamma(\hat{x}') \\
&\hat{x} \sqsubseteq_{\tilde{X}} x' \text{ iff } \gamma(\hat{x}) \sqsubseteq_X x' \\
&\hat{x} \sqsubseteq_{\tilde{X}} \hat{x}' \text{ iff } \gamma(\hat{x}) \sqsubseteq_X \gamma(\hat{x}').
\end{aligned}$$

The definition of the natural Galois union $(\tilde{X}, \mu, \nu, \kappa, \eta)$ is:

$$\begin{aligned}
\mu(x) &= \begin{cases} x & x \notin P \\ \alpha(x) & x \in P \end{cases} \\
\nu(\tilde{x}) &= \begin{cases} x & x \in X \\ \gamma(x) & x \in \hat{X} \end{cases} \\
\kappa(\hat{x}) &= \hat{x} \\
\eta(\tilde{x}) &= \begin{cases} \tilde{x} & \tilde{x} \in \hat{X} \\ \alpha(\tilde{x}) & \tilde{x} \in X. \end{cases}
\end{aligned}$$

4.2 Projecting into the Galois union

We can use the two additional Galois connections provided by the Galois union to project both the concrete and the abstract semantics into the shared union space. According to the Galois connection $X \underset{\mu}{\overset{\nu}{\leftrightarrows}} \tilde{X}$, the projection of the monotonic concrete semantics function $f : X \to X$ into the union space can be calculated, $\tilde{f} : \tilde{X} \to \tilde{X}$:

$$\tilde{f} = \mu \circ f \circ \nu.$$

The projection of the optimal analysis $\hat{f} : \hat{X} \to \hat{X}$ into the union space may be similarly calculated:

$$\begin{aligned}\tilde{\hat{f}} &= \kappa \circ \hat{f} \circ \eta \\ &= \kappa \circ \alpha \circ f \circ \gamma \circ \eta.\end{aligned}$$

4.3 The lattice of semantics

At this point, we are close to our goal of a unified implementation. We have two separate functions—a concrete semantics for interpretation and an abstract semantics for analysis—that inhabit a common state-space. Our next task is to relate these two functions to one another in order to guide a unified implementation. To do so, we will show that these two functions actually form the top and bottom of an entire lattice of hybrid semantics. That is, the bottom of this lattice is the concrete semantics, and the top of this lattice is the optimal analysis.

To construct the lattice, we first show that the concrete semantics ($\tilde{f}$) are weaker than the optimal analysis ($\tilde{\hat{f}}$) according to the natural ordering on functions:

Theorem 4.2. *Given a Galois connection $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X}$, a Galois union thereof $(\tilde{X}, \mu, \nu, \kappa, \eta)$, and a monotonic function $f : X \to X$, the projection of f into the union space, $\tilde{f} : \tilde{X} \to \tilde{X}$ is weaker than the projection of the projection of f into $\hat{X}$ into $\tilde{X}$, $\tilde{\hat{f}}$; that is:*

$$\tilde{f} \sqsubseteq \tilde{\hat{f}},$$

or equivalently:

$$\tilde{f}(\tilde{x}) \sqsubseteq \tilde{\hat{f}}(\tilde{x}) \text{ for all } \tilde{x} \in \tilde{X}.$$

Proof. Pick any $\tilde{x} \in \tilde{X}$. We must show that $\tilde{f}(\tilde{x}) \sqsubseteq \tilde{\hat{f}}(\tilde{x})$. We proceed by cases.

- Case $\tilde{x} \in X$: Observe that:

$$\begin{aligned}\tilde{f}(\tilde{x}) &= (\mu \circ f \circ \nu)(\tilde{x}) \\ &= f(\tilde{x}) \text{ when } f(\tilde{x}) \notin P \text{ or } \alpha(f(\tilde{x})) \text{ when } f(\tilde{x}) \in P,\end{aligned}$$

  and that:

$$\begin{aligned}\tilde{\hat{f}}(\tilde{x}) &= (\kappa \circ \hat{f} \circ \eta)(\tilde{x}) \\ &= (\kappa \circ \alpha \circ f \circ \gamma \circ \eta)(\tilde{x}) \\ &= (\kappa \circ \alpha \circ f \circ \gamma \circ (\alpha \circ \nu))(\tilde{x}) \\ &= (\kappa \circ \alpha \circ f \circ \gamma \circ \alpha)(\tilde{x}) \\ &\sqsupseteq (\kappa \circ \alpha \circ f)(\tilde{x}) \\ &= \alpha(f(\tilde{x})).\end{aligned}$$

  We must show that $f(\tilde{x}) \sqsubseteq_{\tilde{X}} \alpha(f(\tilde{x}))$ when $\gamma(\alpha(f(\tilde{x}))) \sqsupseteq_X f(\tilde{x})$, and this side condition directly satisfies the definition of subsumption.

- Case $\tilde{x} \in \hat{X}$: Observe that:

$$\begin{aligned}\tilde{f}(\tilde{x}) &= (\mu \circ f \circ \nu)(\tilde{x}) \\ &= f(\gamma(\tilde{x})) \text{ when } f(\gamma(\tilde{x})) \notin P \text{ or } \alpha(f(\gamma(\tilde{x}))) \\ &\qquad \text{when } f(\gamma(\tilde{x})) \in P,\end{aligned}$$

  and that:

$$\begin{aligned}\tilde{\hat{f}}(\tilde{x}) &= (\kappa \circ \hat{f} \circ \eta)(\tilde{x}) \\ \tilde{\hat{f}}(\tilde{x}) &= (\kappa \circ \hat{f})(\tilde{x}) \\ &= (\kappa \circ \alpha \circ f \circ \gamma)(\tilde{x}) \\ &= \alpha(f(\gamma(\tilde{x}))),\end{aligned}$$

  which leads to a resolution identical to the prior case.

□

Knowing that the concrete semantics is weaker than the abstract semantics under the partial order on the union space, we know that the ordered interval $[\tilde{f}, \tilde{\hat{f}}]$[2] will be nonempty. Furthermore, if $\tilde{X}$ is a lattice (and it nearly always will be in static analysis), then we can define the join of two semantics function $g, h \in [\tilde{f}, \tilde{\hat{f}}]$:

$$g \sqcup h = \lambda \tilde{x}. g(\tilde{x}) \sqcup h(\tilde{x}),$$

which means that the function space $[\tilde{f}, \tilde{\hat{f}}]$ is itself a lattice.

4.4 Exploiting the lattice: Abstractable interpretation

Practically speaking, the lattice of semantics means that a static analysis may choose to transition using any member of that lattice. This, in turn, leads to a tactic for static analysis that we term *abstractable interpretation*. Under this tactic, analysis begins execution with the concrete semantics. The analysis continues execution with the concrete semantics until it encounters non-determinism (I/O) or until a conservative heuristic detects non-terminating behavior.[3] At this point, the analysis then widens the semantics itself (rather than widening the state of the analysis) to a point higher up the lattice of semantics.

This simple tactic offers practical benefits for at least one common static analysis problem: the global data initialization problem [2]. Consider all of the data that is written once during or shortly after a program's initialization, *e.g.*, virtual function tables in a C++ executable and top-level defines in a Scheme program. In an ordinary abstract interpretation, global data is seen as having two possible values simultaneously: the uninitialized value and then the value it holds for

[2] This is the interval construction from Tarski's proof of his lattice-theoretic fixed point theorem [13], whereby $[a, b] = \{c : a \sqsubseteq c \sqsubseteq b\}$.

[3] For example, a suitable conservative heuristic is "the program may be non-terminating if it has taken more than n transitions."

the program's lifetime. Initializing a static analysis by first executing the concrete semantics for as long as possible allows all of this uninitialized data to be set to its final value with strong updates before the true static analysis phase.

5. Case Study: CPS λ-calculus/k-CFA

To demonstrate the applicability of Galois unions, we will construct a structural Galois-connection-based abstract interpretation of continuation-passing style (CPS) λ-calculus that was examined in section 2, which yields k-CFA. We will then construct a Galois union for a single substructure within this abstract interpretation: abstract addresses. Mechanically, this change is small, yet it allows the allocation function parameter to promote the abstract semantics back into a concrete semantics, in addition to determining the context-sensitivity of k-CFA.

We use a CPS λ-calculus:

$$\begin{array}{rcll} e \in \mathsf{Exp} &=& \mathsf{Lam} + \mathsf{Var} & \text{[expressions]} \\ v \in \mathsf{Var} &=& \langle\text{variables}\rangle & \text{[variables]} \\ \mathit{lam} \in \mathsf{Lam} &::=& \lambda v_1 \ldots v_n . \mathit{call} & [\lambda\text{-terms}] \\ \mathit{call} \in \mathsf{Call} &::=& e_0\ e_1 \ldots e_n & \text{[function application].} \end{array}$$

5.1 Constructing the Galois connection

We define a concrete state-space for continuation-passing style λ-calculus:

$$\begin{array}{rcl} \varsigma \in \Sigma &=& \mathsf{Call} \times \mathit{Env} \times \mathit{Store} \\ \rho \in \mathit{Env} &=& \mathsf{Var} \rightarrow \mathit{Addr} \\ \sigma \in \mathit{Store} &=& \mathit{Addr} \rightarrow D \\ d \in D &=& \mathit{Clo} \\ \mathit{clo} \in \mathit{Clo} &=& \mathsf{Lam} \times \mathit{Env} \\ a \in \mathit{Addr} & \text{is} & \text{an infinite set of addresses,} \end{array}$$

and an abstract state-space:

$$\begin{array}{rcl} \hat{\varsigma} \in \hat{\Sigma} &=& \mathsf{Call}_{\bot}^{\top} \times \widehat{\mathit{Env}} \times \widehat{\mathit{Store}} \\ \hat{\rho} \in \widehat{\mathit{Env}} &=& \mathsf{Var} \rightarrow \widehat{\mathit{Addr}} \\ \hat{\sigma} \in \widehat{\mathit{Store}} &=& \widehat{\mathit{Addr}} \rightarrow \mathcal{P}\left(\widehat{D}\right) \\ \hat{d} \in \hat{D} &=& \widehat{\mathit{Clo}} \\ \widehat{\mathit{clo}} \in \widehat{\mathit{Clo}} &=& \mathsf{Lam}_{\bot}^{\top} \times \widehat{\mathit{Env}} \\ \hat{a} \in \widehat{\mathit{Addr}} & \text{is} & \text{a finite set of addresses.} \end{array}$$

The Galois connection process begins by examining the leaves of the state-space. In this case, the key leaf is the set of addresses: Addr. We assume some address-abstractor $\beta : \mathit{Addr} \rightarrow \widehat{\mathit{Addr}}$, β maps infinite address spaces to the finite set of abstract spaces. We then use it to define a Galois connection:

$$(\mathcal{P}(\mathit{Addr}), \alpha, \gamma, \widehat{\mathit{Addr}}).$$

This is the particular Galois connection that we will revisit when constructing the Galois union. We can lift this Galois connection to a function space:

$$(\mathcal{P}(\mathsf{Var} \rightarrow \mathit{Addr}), \alpha_1, \gamma_1, \mathsf{Var} \rightarrow \widehat{\mathit{Addr}})$$

$$= (\mathcal{P}(\mathit{Env}), \alpha_1, \gamma_1, \widehat{\mathit{Env}}).$$

λ-terms lift into a flat Galois connection:

$$(\mathcal{P}(\mathsf{Lam}), \alpha_2, \gamma_2, \mathsf{Lam}_{\bot}^{\top}),$$

which makes it easy to construct a Galois connection over closures:

$$(\mathcal{P}(\mathsf{Lam} \times \mathit{Env}), \alpha_3, \gamma_3, \mathsf{Lam}_{\bot}^{\top} \times \widehat{\mathit{Env}})$$
$$= (\mathcal{P}(\mathit{Clo}), \alpha_3, \gamma_3, \widehat{\mathit{Clo}}).$$

By promoting closures to a fully relational Galois connection, we have a Galois connection for values:

$$(\mathcal{P}(\mathit{Clo}), \alpha_4, \gamma_4, \mathcal{P}(\widehat{\mathit{Clo}})) = (\mathcal{P}(D), \alpha_4, \gamma_4, \hat{D}).$$

Lifting once again yields a Galois connection over stores:

$$(\mathcal{P}(\mathit{Addr} \rightarrow D), \alpha_5, \gamma_5, \widehat{\mathit{Addr}} \rightarrow \hat{D})$$
$$= (\mathcal{P}(\mathit{Store}), \alpha_5, \gamma_5, \widehat{\mathit{Store}}).$$

The Galois connection for call sites is flat:

$$(\mathcal{P}(\mathsf{Call}), \alpha_6, \gamma_6, \mathsf{Call}_{\bot}^{\top}).$$

Combining all of the above yields a Galois connection on states:

$$(\mathcal{P}(\mathsf{Call} \times \mathit{Env} \times \mathit{Store}), \alpha_7, \gamma_7, \mathsf{Call}_{\bot}^{\top} \times \widehat{\mathit{Env}} \times \widehat{\mathit{Store}})$$
$$= (\mathcal{P}(\mathit{State}), \alpha_7, \gamma_7, \widehat{\mathit{State}}),$$

which can be lifted into a more precise abstraction:

$$(\mathcal{P}(\mathit{State}), \alpha_8, \gamma_8, \mathcal{P}(\widehat{\mathit{State}})).$$

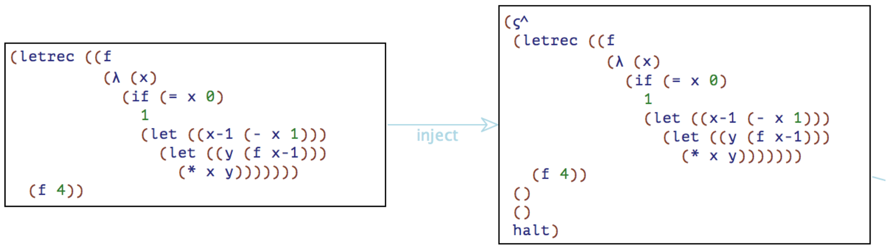
5.2 Calculating an abstract semantics

The transfer function $f : \Sigma \rightarrow \Sigma$ describes the concrete semantics:

$$\begin{array}{rl} f\,(\overbrace{[\![e_0\ e_1 \ldots e_n]\!], \rho, \sigma}^{\varsigma}) &= (\mathit{call}, \rho'', \sigma'), \text{ where:} \\ ([\![\lambda v_1 \ldots v_n . \mathit{call}]\!], \rho') &= \mathcal{A}(e_0, \rho, \sigma) \\ a_i &= \mathit{alloc}(v_i, \varsigma) \\ \rho'' &= \rho'[v_i \mapsto a_i] \\ \sigma' &= \sigma[a_i \mapsto d_i] \\ d_i &= \mathcal{A}(e_i, \rho, \sigma), \end{array}$$

where the function $\mathcal{A} : \mathsf{Exp} \times \mathit{Env} \times \mathit{Store} \rightarrow D$ is the argument evaluator:

$$\mathcal{A}(v, \rho, \sigma) = \sigma(\rho(v))$$
$$\mathcal{A}(\mathit{lam}, \rho, \sigma) = (\mathit{lam}, \rho),$$



```
(letrec ((f
          (λ (x)
            (if (= x 0)
              1
              (let ((x-1 (- x 1)))
                (let ((y (f x-1)))
                  (* x y)))))))
  (f 4))
```

```
(ς^
 (letrec ((f
           (λ (x)
             (if (= x 0)
               1
               (let ((x-1 (- x 1)))
                 (let ((y (f x-1)))
                   (* x y)))))))
   (f 4))
 ()
 ()
 halt)
```

Figure 1. Injection Into the Start State

and the allocator $alloc : \mathsf{Var} \times \Sigma \to Addr$ allocates a fresh address.

Promoting the transfer function to sets gives the function $F : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$:

$$F(S) = f.S = \{f(\varsigma) : \varsigma \in S\},$$

which allows us to calculate the optimal analysis, $\hat{F} = \alpha_8 \circ F \circ \gamma_8$.

A series of calculations (Appendix A.1) then finds a computable static analysis:

$$\hat{F}\{\overbrace{(call, \hat\rho, \hat\sigma)}^{\hat\varsigma}\} \sqsubseteq \left\{ (call', \hat\rho'', \hat\sigma') : \left\{ \begin{array}{l} \hat{a}_i = \widehat{alloc}(v_i, \hat\varsigma) \\ ([\![\lambda v_1 \ldots v_n.call]\!], \hat\rho') \\ \quad \in \hat{\mathcal{A}}(e_0, \hat\rho, \hat\sigma) \\ \hat\rho'' = \hat\rho'[v_i \mapsto \hat{a}_i] \\ \hat\sigma' = \hat\sigma \sqcup [\hat{a}_i \mapsto \hat{d}_i] \\ \hat{d}_i = \hat{\mathcal{A}}(e_i, \hat\rho, \hat\sigma) \end{array} \right\} \right\},$$

where the function $\hat{\mathcal{A}} : \mathsf{Exp} \times \widehat{Env} \times \widehat{Store} \to \hat{D}$ is the abstract evaluator:

$$\hat{\mathcal{A}}(v, \hat\rho, \hat\sigma) = \hat\sigma(\hat\rho(v))$$
$$\hat{\mathcal{A}}(lam, \hat\rho, \hat\sigma) = \{(lam, \hat\rho)\}.$$

And, we have the constraint whereby any function $\widehat{alloc}$ such that:

$$\alpha_7\{\varsigma\} \sqsubseteq \hat\varsigma \text{ implies } \alpha\{alloc(v, \varsigma)\} = \widehat{alloc}(v, \hat\varsigma),$$

leads to a sound analysis.

5.3 Constructing the Galois union

As it turns out, constructing the Galois union for the *entire* Galois connection over state-spaces $\mathcal{P}(\Sigma) \underset{\alpha_8}{\overset{\gamma_8}{\leftrightarrows}} \mathcal{P}(\hat\Sigma)$, while sufficient, is not necessary. Rather, as is often the case in practice, it is sufficient (and easier) to construct the Galois union for only the leaves of the state-spaces. And, in this case, the only leaf of consequence is the Galois connection over addresses: $\mathcal{P}(Addr) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \widehat{Addr}$.[4] The natural Galois union-space for addresses is the set $\widetilde{Addr} \subset \mathcal{P}(Addr) + \widehat{Addr}$, which then percolates up to create a union-space for states:

$$\begin{aligned} \tilde\varsigma \in \tilde\Sigma &= \mathsf{Call}_\perp^\top \times \widetilde{Env} \times \widetilde{Store} \\ \tilde\rho \in \widetilde{Env} &= \mathsf{Var} \to \widetilde{Addr} \\ \tilde\sigma \in \widetilde{Store} &= \widetilde{Addr} \to \tilde{D} \\ \tilde{d} \in \tilde{D} &= \mathcal{P}(\widetilde{Clo}) \\ \widetilde{clo} \in \widetilde{Clo} &= \mathsf{Lam}_\perp^\top \times \widetilde{Env}. \end{aligned}$$

5.4 Calculating a unified implementation

To extract the unified implementation, we replace the set of abstract addresses with the set of unioned addresses, and repeat the prior projection process exactly. This results in a "new" unified transfer function:

$$\tilde{F}\{\overbrace{(call, \tilde\rho, \tilde\sigma)}^{\tilde\varsigma}\} \sqsubseteq \left\{ (call', \tilde\rho'', \tilde\sigma') : \left\{ \begin{array}{l} \tilde{a}_i = \widetilde{alloc}(v_i, \tilde\varsigma) \\ ([\![\lambda v_1 \ldots v_n.call]\!], \tilde\rho') \\ \quad \in \tilde{\mathcal{A}}(e_0, \tilde\rho, \tilde\sigma) \\ \tilde\rho'' = \tilde\rho'[v_i \mapsto \tilde{a}_i] \\ \tilde\sigma' = \tilde\sigma \sqcup [\tilde{a}_i \mapsto \tilde{d}_i] \\ \tilde{d}_i = \tilde{\mathcal{A}}(e_i, \tilde\rho, \tilde\sigma) \end{array} \right\} \right\}.$$

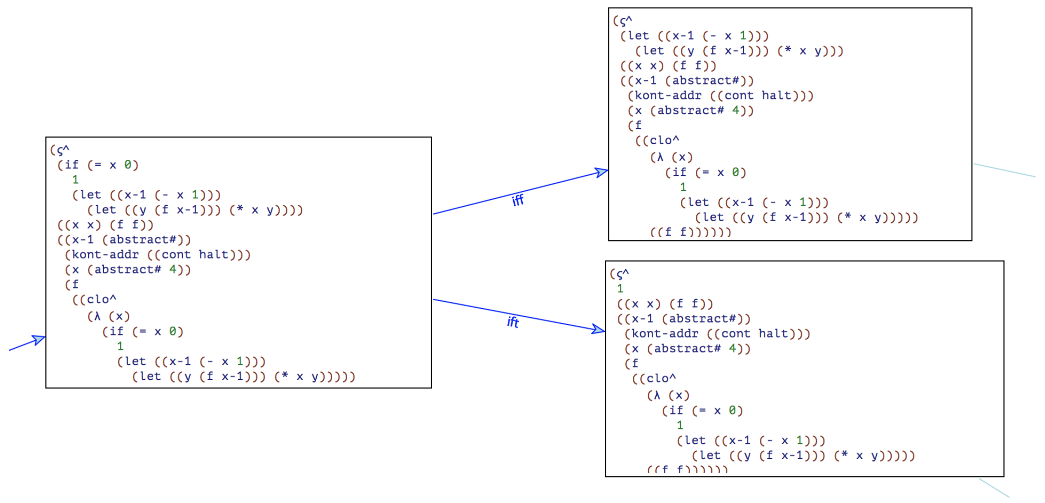
Of course, these transfer functions looks identical (modulo tildes and hats) to the previously derived transfer function. The difference comes in that the allocation function, $\widetilde{alloc} : \mathsf{Var} \times \tilde\Sigma \to \widetilde{Addr}$—which allocates addresses—is now free to allocate sets of concrete addresses alongside abstract addresses. If this allocation function mimics the behavior of the original concrete allocator (by allocating singletons), then the result is a sound and complete simulation of the concrete semantics; but if this allocation function mimics the behavior of the abstract allocator, the result is k-CFA. In practice, the implementations of either allocator takes about one line of code, which means that for the cost of the static analysis plus one line of code, we also obtain the concrete semantics.

6. Implementation: CESK

To show how the unified representation works in practice we developed an implementation[5] of ANF λ-Calculus based on the CESK machine [6]modified to use galois unions. To implement it we used the domain specific language PLT Redex[7]. The implementation will use the concrete semantics for an arbitrarily large yet finite amount of states and

[4] Constructing the Galois unions of the other leaves—Var and Call—yields exactly the abstract space again.

[5] http://github.com/LeifAndersen/CESKRedex/



```
(ς^
 (if (= x 0)
   1
   (let ((x-1 (- x 1)))
     (let ((y (f x-1))) (* x y))))
 ((x x) (f f))
 ((x-1 (abstract#))
  (kont-addr ((cont halt)))
  (x (abstract# 4))
  (f
   ((clo^
     (λ (x)
       (if (= x 0)
         1
         (let ((x-1 (- x 1)))
           (let ((y (f x-1))) (* x y)))))
```

```
(ς^
 (let ((x-1 (- x 1)))
   (let ((y (f x-1))) (* x y)))
 ((x x) (f f))
 ((x-1 (abstract#))
  (kont-addr ((cont halt)))
  (x (abstract# 4))
  (f
   ((clo^
     (λ (x)
       (if (= x 0)
         1
         (let ((x-1 (- x 1)))
           (let ((y (f x-1))) (* x y)))))
     ((f f))))))
```

```
(ς^
 1
 ((x x) (f f))
 ((x-1 (abstract#))
  (kont-addr ((cont halt)))
  (x (abstract# 4))
  (f
   ((clo^
     (λ (x)
       (if (= x 0)
         1
         (let ((x-1 (- x 1)))
           (let ((y (f x-1))) (* x y)))))
     ((f f))))))
```

Figure 2. Branching to Multiple States

conclude with the abstract semantics. The user may specify the threshold at which it switches from the concrete to the abstract machine. The machine can also run completely concretely or completely abstractly.

```
(define-metafunction CESK˜
  alloc˜ : store˜ x -> addr˜
  [(alloc˜ store˜ x)
   ,(if
     ((length (flatten (term store˜))) . < . 100)
        (variable-not-in (term store˜) (term x))
        (term x))])
```

Figure 3. Allocation Metafunction

The abstract semantics and the unified semantics differ in the allocation function. The abstract semantics allocates abstract addresses and abstract values. The unified semantics generates concrete values in addition to abstract values but limits the allocation such that it still contains a finite amount of space. This small change allows the allocation function parameter to promote the abstract semantics back into a concrete semantics.

An example makes it clear to see the machine abstract itself and allocate concretely and abstractly.

6.1 Example: Factorial

The code under analysis in Figure 2 is Factorial of four. To demonstrate the effectiveness of the CESK machine, as shown in Figure 1, the program is injected into a start state with an empty environment, an empty store, and the halt continuation. The program continues in the concrete semantics for a finite amount of states.

Eventually the code will reach Figure 3 in which it will switch to the abstract semantics and branch into multiple states until the analysis terminates.The ability to switch from concrete to abstract and visualize the states gives analysts using this tool a clearer picture of their analyses.

```
(letrec
      ((f (lambda (x)
              (if (= x 0)
                   1
                   (let ((x-1 (- x 1)))
                      (let ((y (f x-1)))
                         (* x y)))))))
        (f 4))
```

Figure 4. Factorial

In our implementation we built the concrete interpreter, an abstract interpreter and a unified interpreter to compare the amount of engineering effort that was needed. In terms of lines of code the abstract machine was 14 more lines of code than the concrete machine. The unified machine was the same number of lines of code as the abstract machine. For 14 extra lines of code we get a static analyzer that doubles as a concrete interpreter.

7. Related Work

The idea that a concrete interpreter is also an incomputable static analysis is at least as old the Cousots' original work [3] on abstract interpretation. The inverse of that idea—that a static analysis can be systematically engineered to also serve as a concrete interpreter in addition to its regular duties—is, to the best of our knowledge, novel. Our definition of Galois Union, a property of a Galois connection, is novel as well.

The Cousots' early work details using Galois connections to systematically design static analyses [4]. The Cousots' later work on higher-order abstract interpretation [5] and Nielson, Nielson and Hankin's work [10] provide a complete treatement of both abstract interpretation and Galois connections. k-CFA, in the form that we derive it here, is closely related to Shivers's original formualtion [12].

8. Conclusion

Our goal was to use Galois unions to guide an implementation of a static analyzer that doubles as a concrete interpreter. We provided a framework to systematically enhance an abstract interpretation to also behave as a concrete interpreter.

This material is partially based on research sponsored by DARPA under agreement number FA8750-12- 2-0106 and by NSF under CAREER grant 1350344. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] A. W. Appel. *Compiling with Continuations*. Cambridge University Press, February 2007. ISBN 052103311X.

[2] G. Balakrishnan and T. Reps. Recency-abstraction for heap-allocated storage. In *Proceedings of the Static Analysis Symposium*, Seoul, Korea, 2006.

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, 1977. ACM Press, New York.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, TX, 1979. ACM Press, New York.

[5] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *n Proceedings of the 1994 International Conference on Computer Languages*, pages 238–252, 1994.

[6] M. Felleisen and D. P. Friedman. A calculus for assignments in higher-order languages. In *Proceedings of the Symposium on Principles of Programming Languages*, page 314, New York, NY, 1987. ACM.

[7] M. Felleisen, R. B. Findler, and M. Flatt. *Semantics Engineering with PLT Redex*. MIT Press, 2009. ISBN 0262062755 9780262062756.

[8] D. V. Horn and M. Might. Abstracting abstract machines. In *Proceedings of the International Conference on Functional Programming*, September 2010.

[9] M. Might. Abstract interpreters for free. In *17th International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010. Proceedings*, volume 6337, pages 407–421, 2010.

[10] F. Nielson and C. H. Hanne R Nielson. *Principles of Program Analysis*. Springer, 1999.

[11] O. Shivers. Control flow analysis in scheme. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 164–174, New York, NY, 1988. ACM.

[12] O. Shivers. *Control-Flow Analysis of Higher-Order Languages*. PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[13] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. pages 285–309, 1955.

A. Appendix

A.1 Calculation of k-CFA

We include the remainder of the calculation of k-CFA here:

$$\begin{aligned}
&\hat{F}\left\{(\mathit{call},\hat\rho,\hat\sigma)\right\} = (\alpha_8 \circ F \circ \gamma_8)\left\{(\mathit{call},\hat\rho,\hat\sigma)\right\}\\
&\quad = (\alpha_8 \circ F)(\gamma_8\left\{(\mathit{call},\hat\rho,\hat\sigma)\right\})\\
&\quad = (\alpha_8 \circ F)\left\{(\mathit{call},\rho,\sigma) : \alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho \text{ and } \alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\right\}\\
&\quad = (\alpha_8)\left\{f(\mathit{call},\rho,\sigma) : \alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho \text{ and } \alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\right\}\\
&\quad = \bigsqcup\left\{\left\{\alpha_7\left\{f(\mathit{call},\rho,\sigma)\right\}\right\} : \alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho \text{ and } \alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\right\}.
\end{aligned}$$

An inconsequential weakening makes the last line easier to understand:

$$\hat{F}\left\{(\mathit{call},\hat\rho,\hat\sigma)\right\} \sqsubseteq$$
$$\left\{\alpha_7\left\{f(\mathit{call},\rho,\sigma)\right\} : \alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho \text{ and } \alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\right\}.$$

To proceed, we can expand the transfer function and the abstraction function:

$$\hat{F}\{(\mathit{call},\hat\rho,\hat\sigma)\} \sqsubseteq$$
$$\left\{ (\mathit{call}',\alpha_1\left\{\rho''\right\},\alpha_5\left\{\sigma'\right\}) : \left\{\begin{array}{l}
\alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho\\
\alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\\
([\![\lambda v_1 \ldots v_n.\mathit{call}]\!],\rho')\\
\quad = \mathcal{A}(e_0,\rho,\sigma)\\
a_i = \mathit{alloc}(v_i,\varsigma)\\
\rho'' = \rho'[v_i \mapsto a_i]\\
\sigma' = \sigma[a_i \mapsto d_i]\\
d_i = \mathcal{A}(e_i,\rho,\sigma)
\end{array}\right\}\right\}. \tag{A.1}$$

We make a series of observations. Suppose that $\alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho$ and $\alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma$. Then let $\mathit{clo} = ([\![\lambda v_1 \ldots v_n.\mathit{call}]\!],\rho') = \mathcal{A}(e_0,\rho,\sigma)$. By cases, we can show that for any expression e:

$$\alpha_3\left\{\mathcal{A}(e,\rho,\sigma)\right\} \sqsubseteq \hat{\mathcal{A}}(e,\hat\rho,\hat\sigma).$$

There must exist a closure $\widehat{\mathit{clo}} = (\mathit{lam},\hat\rho) \in \hat{\mathcal{A}}(\exp_0,\hat\rho,\hat\sigma)$ such that:

$$\alpha_3\left\{\mathit{clo}\right\} \sqsubseteq \widehat{\mathit{clo}}.$$

So, we may further weaken the function $\hat{F}$

$$\hat{F}\{(\mathit{call},\hat\rho,\hat\sigma)\} \sqsubseteq$$
$$\left\{ (\mathit{call}',\alpha_1\left\{\rho''\right\},\alpha_5\left\{\sigma'\right\}) : \left\{\begin{array}{l}
\alpha_1\left\{\rho\right\} \sqsubseteq \hat\rho\\
\alpha_5\left\{\sigma\right\} \sqsubseteq \hat\sigma\\
([\![\lambda v_1 \ldots v_n.\mathit{call}]\!],\hat\rho')\\
\quad \in \hat{\mathcal{A}}(e_0,\hat\rho,\hat\sigma)\\
\alpha_1\left\{\rho'\right\} \sqsubseteq \hat\rho'\\
a_i = \mathit{alloc}(v_i,\varsigma)\\
\rho'' = \rho'[v_i \mapsto a_i]\\
\sigma' = \sigma[a_i \mapsto d_i]\\
d_i = \mathcal{A}(e_i,\rho,\sigma)
\end{array}\right\}\right\}.$$

Thus, assuming $\alpha_1\left\{\mathit{env}'\right\} \sqsubseteq \hat\rho'$ and $\alpha\left\{a_i\right\} = \hat{a}_i$:

$$\alpha_1\left\{\rho''\right\} \sqsubseteq \hat\rho'' = \hat\rho'[v_i \mapsto a_i].$$

Then, we have:

$$\begin{aligned}\alpha_3\,\{d_i\} &\sqsubseteq \hat{d}_i = \hat{\mathcal{A}}(e_i, \hat{\rho}_i, \hat{\sigma}_i)\\ \alpha_5\,\{\sigma'\} &\sqsubseteq \hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{d}_i].\end{aligned}$$

All of this permits a further weakening:

$$\hat{F}\,\{(call, \hat{\rho}, \hat{\sigma})\} \sqsubseteq$$

$$\left\{ (call', \hat{\rho}'', \hat{\sigma}') : \left\{ \begin{array}{l} ([\![\lambda v_1 \ldots v_n.call]\!], \hat{\rho}') \\ \quad \in \hat{\mathcal{A}}(e_0, \hat{\rho}, \hat{\sigma}) \\ \hat{\rho}'' = \hat{\rho}'[v_i \mapsto \hat{a}_i] \\ \hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{d}_i] \\ \hat{d}_i = \hat{\mathcal{A}}(e_i, \hat{\rho}, \hat{\sigma}) \end{array} \right\} \right\}.$$