A Calculational Approach to Control-flow Analysis by Abstract Interpretation

Jan Midtgaard[1] and Thomas Jensen[2]

[1] INRIA Rennes - Bretagne Atlantique
[2] CNRS
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{jan.midtgaard,thomas.jensen}@irisa.fr

Abstract. We present a derivation of a control-flow analysis by abstract interpretation. Our starting point is a transition system semantics defined as an abstract machine for a small functional language in continuation-passing style. We obtain a Galois connection for abstracting the machine states by composing Galois connections, most notable an independent-attribute Galois connection on machine states and a Galois connection induced by a closure operator associated with a constituent-parts relation on environments. We calculate abstract transfer functions by applying the state abstraction to the collecting semantics, resulting in a novel characterization of demand-driven 0-CFA.

1 Introduction

Over twenty-five years ago Jones [16] statically approximated the flow of lambda-expressions. Since then control-flow analysis (CFA) has been the subject of immense research [2, 25, 30, 31]. Ten years ago Nielson and Nielson designed a co-inductive collecting semantics for control-flow analysis and at the same time asked [25, p.1]: *How does one exploit Galois connections and widenings to systematically coarsen [control-flow analysis]?"*

In this paper we take the first steps towards answering that question, by expressing a control-flow analysis as the composition of several well-known Galois connections, thereby spelling out the approximations taking place. Our approach thus follows Cousot's programme of *calculational abstract interpretation* [6] in which an abstract interpretation is calculated by systematically applying abstraction functions to a formal programming language semantics.

We develop our approach in the setting of CFA for functional languages. A substantial amount of work concerned with abstract interpretation of functional languages is based on denotational semantics [19] in which source-level functions are modelled with mathematical functions [29]. However, as CFA is concerned with operational information about source-level functions we believe that a denotational starting point is inadequate for a calculational derivation of control-flow analysis by abstract interpretation. Instead, we have chosen to base our derivation on an operational semantics in the form of an abstract machine (a transition system) in which source-level functions are modelled with *closures* which are pairs of expressions and environments. Closures were originally suggested by Landin to model functions in the SECD machine [20] and have since become a standard implementation method for functional languages [1].

$$\begin{tikzcd}
\wp(\mathit{SExp} \times \mathit{Env}) \arrow[d, shift right, "\alpha_\times"'] & \text{collecting semantics} & \text{(Figure 4)} \\
\wp(\mathit{SExp}) \times \wp(\mathit{Env}) \arrow[u, shift right, "\gamma_\times"'] \arrow[d, shift right, "\alpha_\otimes"'] & \text{intermediate transfer function} & \text{(Figure 5)} \\
\wp(\mathit{SExp}) \times \mathit{Env}^\sharp \arrow[u, shift right, "\gamma_\otimes"'] & \text{demand-driven 0-CFA} & \text{(Figure 6)}
\end{tikzcd}$$

Fig. 1: The big picture

The aim of a functional CFA is to determine which functions are bound to which variables during execution. This information is found in the environments during the execution of the program. Accordingly, an essential part of the abstraction technique that we propose is to express appropriate Galois connections for extracting the approximate environment components of a machine state. Environments are recursive structures that themselves may contain closures which in turn contain environments. A crucial step in the derivation is the definition of an upper closure operator on environments that induce an appropriate Galois connection. Figure 1 summarizes the two steps of the abstraction. From a reachable-states collecting semantics of the CE machine [14], defined in Section 5, we first abstract the machine components as independent attributes. Next we abstract the pair of sets by an environment abstraction based on the notion of *constituent relation* of Milner and Tofte [23]. These and other Galois connections for abstracting values are defined in Section 6. The composition of these Galois connections yields an abstraction function that is again applied to the transfer function to calculate a demand-driven 0-CFA. The calculations are given in Section 7. The contributions of the paper are as follows.

– We start from a well-known operational semantics, instead of instrumenting a collecting semantics.
– We apply Cousot-style abstract interpretation to CFA: reachable states of a transition system systematically abstracted through Galois connections.
– We explain the approximations of a CFA by spelling it out as the composition of several well-known Galois connections.
– We characterize demand-driven 0-CFA as a simple independent attribute abstraction of a reachable states semantics.
– Finally, we *calculate* the analysis rather than postulating it and verifying it a posteriori.

An implementation of the derived analysis and an example is briefly explained in Section 8. Section 9 compares the analysis to related work. Section 10 concludes and lists future work which notably consists in calculating a flow-sensitive CFA by using alternative Galois connections to approximate machine states. We assume the reader is familiar with operational semantics, continuation-passing style (CPS), control-flow analysis, complete lattices, and fixed points. The following sections provides a concise summary of well-known facts about abstract interpretation that will be used in the paper.

2 A short introduction to abstract interpretation

We recall a number of properties related to complete lattices, Galois connections, and closure operators. The section consists of known material from the research literature [7–9, 11, 13]. Readers familiar with the material can skip to the next section.

2.1 Galois connections etc.

The powerset of a set S is written $\wp(S)$. The powerset $\wp(S)$ ordered by set inclusion is a complete lattice $\langle \wp(S); \subseteq, \emptyset, S, \cup, \cap \rangle$. A *Galois connection* between two posets $\langle \mathcal{D}_1; \subseteq_1 \rangle$ and $\langle \mathcal{D}_2; \subseteq_2 \rangle$ (the *concrete* and the *abstract* domain) is a pair of maps $\alpha : \mathcal{D}_1 \to \mathcal{D}_2$ (the *abstraction* map) and $\gamma : \mathcal{D}_2 \to \mathcal{D}_1$ (the *concretisation* map) such that $\forall c \in \mathcal{D}_1 : \forall a \in \mathcal{D}_2 : \alpha(c) \subseteq_2 a \iff c \subseteq_1 \gamma(a)$. Equivalently α is monotone, γ is monotone, $\gamma \circ \alpha$ is extensive ($\forall c \in \mathcal{D}_1 : c \subseteq_1 \gamma \circ \alpha(c)$), and $\alpha \circ \gamma$ is reductive ($\forall a \in \mathcal{D}_2 : \alpha \circ \gamma(a) \subseteq_2 a$). Galois connections are typeset as $\langle \mathcal{D}_1; \subseteq_1 \rangle \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} \langle \mathcal{D}_2; \subseteq_2 \rangle$. When the domains are apparent from the context we use the lighter notation $\mathcal{D}_1 \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} \mathcal{D}_2$. Now let $\langle \mathcal{D}_1; \subseteq_1, \bot_1, \top_1, \cup_1, \cap_1 \rangle$ and $\langle \mathcal{D}_2; \subseteq_2, \bot_2, \top_2, \cup_2, \cap_2 \rangle$ be complete lattices. Given a Galois connection $\mathcal{D}_1 \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} \mathcal{D}_2$ then α is a complete join-morphism (CJM) ($\alpha(\cup_1 X) = \cup_2 \alpha(X) = \cup_2 \{\alpha(x) \mid x \in X\}$) (and γ is a complete meet-morphism). Given a complete join-morphism α and $\gamma(y) = \cup_1 \{x \in \mathcal{D}_1 \mid \alpha(x) \subseteq_2 y\}$ then they form a Galois connection.

Example 1 (Identity abstraction). Two identity functions $1_{\mathcal{D}} = \lambda x. x$ form a Galois connection on a poset $\langle \mathcal{D}; \subseteq \rangle \overset{1_{\mathcal{D}}}{\underset{1_{\mathcal{D}}}{\leftrightarrows}} \langle \mathcal{D}; \subseteq \rangle$.

Example 2 (Elementwise abstraction). Let an elementwise operator $@ : C \to A$ be given. Define $\alpha_{@}(P) = \{@(p) \mid p \in P\}$ and $\gamma_{@}(Q) = \{p \mid @(p) \in Q\}$. Then $\wp(C) \overset{\gamma_{@}}{\underset{\alpha_{@}}{\leftrightarrows}} \wp(A)$.

Example 3 (Pointwise abstraction of a set of functions). Assume an abstraction

$$\langle \wp(\mathcal{D}_2); \subseteq, \emptyset, \mathcal{D}_2, \cup, \cap \rangle \overset{\gamma_2}{\underset{\alpha_2}{\leftrightarrows}} \langle \mathcal{D}_2^\sharp; \subseteq_2^\sharp, \bot_2^\sharp, \top_2^\sharp, \cup_2^\sharp, \cap_2^\sharp \rangle$$

and let

$$\begin{aligned} \alpha_\Pi(F) &= \lambda x. \alpha_2(\{f(x) \mid x \in \mathcal{D}_1 \ \wedge \ f \in F\}) \\ \gamma_\Pi(\Phi) &= \{f \in \mathcal{D}_1 \to \mathcal{D}_2 \mid \forall x : f(x) \in \gamma_2(\Phi(x))\} \end{aligned}$$

Then $\langle \wp(\mathcal{D}_1 \to \mathcal{D}_2); \subseteq, \emptyset, \mathcal{D}_1 \to \mathcal{D}_2, \cup, \cap \rangle \overset{\gamma_\Pi}{\underset{\alpha_\Pi}{\leftrightarrows}} \langle \mathcal{D}_1 \to \mathcal{D}_2^\sharp; \dot{\subseteq}_2^\sharp, \dot{\bot}_2^\sharp, \dot{\top}_2^\sharp, \dot{\cup}_2^\sharp, \dot{\cap}_2^\sharp \rangle$ where we have used the pointwise notation $A \ \dot{r} \ B \iff \forall x.A(x) \ r \ B(x)$ and $\dot{c} = \lambda\_. c$ for relations and constants.

Cousot and Cousot [11, Sect.3] describe the pointwise abstraction as the composition of three abstractions.

One can abstract a set of pairs into a pair of sets, in turn performing an *attribute independent abstraction* [18], as relational information between the components of the individual pairs is lost.

Example 4 (Abstraction of a binary relation by a pair of sets). Let

$$\alpha_\times(r) = \langle \pi_1(r),\, \pi_2(r) \rangle \qquad\qquad \gamma_\times(\langle X,\, Y \rangle) = X \times Y$$

where $\pi_1(r) = \{x \mid \exists y : \langle x, y \rangle \in r\}$, $\pi_2(r) = \{y \mid \exists x : \langle x, y \rangle \in r\}$, and let $\subseteq_\times = \subseteq \times \subseteq$, $\bot_\times = \langle \emptyset, \emptyset \rangle$, $\top_\times = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$, $\cup_\times = \cup \times \cup$, and $\cap_\times = \cap \times \cap$. Then

$$\langle \wp(\mathcal{D}_1 \times \mathcal{D}_2); \subseteq, \emptyset, \mathcal{D}_1 \times \mathcal{D}_2, \cup, \cap \rangle \xleftarrow[\alpha_\times]{\gamma_\times} \langle \wp(\mathcal{D}_1) \times \wp(\mathcal{D}_2); \subseteq_\times, \bot_\times, \top_\times, \cup_\times, \cap_\times \rangle$$

An *upper closure operator* is a map $\rho : \mathcal{D} \to \mathcal{D}$ on a poset $\langle \mathcal{D}; \subseteq \rangle$ that is extensive ($\forall x \in \mathcal{D} : x \subseteq \rho(x)$), monotone ($\forall x, x' \in \mathcal{D} : x \subseteq x' \implies \rho(x) \subseteq \rho(x')$), and idempotent ($\forall x \in \mathcal{D} : \rho(x) = \rho(\rho(x))$). A closure operator ρ on a poset $\langle \mathcal{D}; \subseteq \rangle$ induces a Galois connection: $\langle \mathcal{D}; \subseteq \rangle \xleftarrow[\rho]{1_\mathcal{D}} \langle \rho(\mathcal{D}); \subseteq \rangle$. Finally the image of a complete lattice $\langle \mathcal{D}; \subseteq, \bot, \top, \cup, \cap \rangle$ by a closure operator ρ is itself a complete lattice $\langle \rho(\mathcal{D}); \subseteq, \rho(\bot), \rho(\top), \lambda X.\, \rho(\cup X), \cap \rangle$.

2.2 Composition of Galois connections

Galois connections enjoy a number of properties regarding composition. One can abstract a pair of sets by abstracting its components.

Example 5 (Abstraction of a pair of sets by an abstract pair). Assuming two Galois connections $\langle \wp(\mathcal{D}_1); \subseteq, \emptyset, \mathcal{D}_1, \cup, \cap \rangle \xleftarrow[\alpha_1]{\gamma_1} \langle \mathcal{D}_1^\sharp; \subseteq_1^\sharp, \bot_1^\sharp, \top_1^\sharp, \cup_1^\sharp, \cap_1^\sharp \rangle$ and $\langle \wp(\mathcal{D}_2); \subseteq, \emptyset, \mathcal{D}_2, \cup, \cap \rangle \xleftarrow[\alpha_2]{\gamma_2} \langle \mathcal{D}_2^\sharp; \subseteq_2^\sharp, \bot_2^\sharp, \top_2^\sharp, \cup_2^\sharp, \cap_2^\sharp \rangle$, define

$$\alpha_\otimes(\langle X,\, Y \rangle) = \langle \alpha_1(X),\, \alpha_2(Y) \rangle \qquad\qquad \gamma_\otimes(\langle x,\, y \rangle) = \langle \gamma_1(x),\, \gamma_2(y) \rangle$$

where $\subseteq_\otimes = \subseteq_1^\sharp \times \subseteq_2^\sharp$, $\bot_\otimes = \langle \bot_1^\sharp, \bot_2^\sharp \rangle$, $\top_\otimes = \langle \top_1^\sharp, \top_2^\sharp \rangle$, $\cup_\otimes = \cup_1^\sharp \times \cup_2^\sharp$, and $\cap_\otimes = \cap_1^\sharp \times \cap_2^\sharp$. Then

$$\langle \wp(\mathcal{D}_1) \times \wp(\mathcal{D}_2); \subseteq_\times, \bot_\times, \top_\times, \cup_\times, \cap_\times \rangle \xleftarrow[\alpha_\otimes]{\gamma_\otimes} \langle \mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp; \subseteq_\otimes, \bot_\otimes, \top_\otimes, \cup_\otimes, \cap_\otimes \rangle$$

Most importantly Galois connections compose sequentially.

Lemma 1 (Compositional abstraction). *Given two Galois connections between complete lattices* $\langle \mathcal{D}_0; \subseteq_0, \bot_0, \top_0, \cup_0, \cap_0 \rangle \xleftarrow[\alpha_1]{\gamma_1} \langle \mathcal{D}_1; \subseteq_1, \bot_1, \top_1, \cup_1, \cap_1 \rangle$ *and* $\langle \mathcal{D}_1; \subseteq_1, \bot_1, \top_1, \cup_1, \cap_1 \rangle \xleftarrow[\alpha_2]{\gamma_2} \langle \mathcal{D}_2; \subseteq_2, \bot_2, \top_2, \cup_2, \cap_2 \rangle$*, then*

$$\langle \mathcal{D}_0; \subseteq_0, \bot_0, \top_0, \cup_0, \cap_0 \rangle \xleftarrow[\alpha_2 \circ \alpha_1]{\gamma_1 \circ \gamma_2} \langle \mathcal{D}_2; \subseteq_2, \bot_2, \top_2, \cup_2, \cap_2 \rangle$$

3 Language

As our source language we take CPS expressions characterized by the grammar in Figure 2 [12]. The grammar distinguishes between *serious* expressions, denoting expressions whose evaluation may diverge, and *trivial* expressions, denoting expressions whose evaluation will terminate. We further distinguish three different forms of variables: source variables x, continuation variables k, and formal continuation parameters v, each drawn from disjoint countable sets of variables X, K, and V, respectively. We let Var denote the disjoint union of the three $\mathit{Var} = X \cup V \cup K$. When apparent from the context we will also use x as a generic meta-variable $x \in \mathit{Var}$.

$$\begin{array}{lll}
p & ::= \lambda k.\, e & \text{(CPS programs)} \\
e & ::= t_0\, t_1\, c \;\mid\; c\, t & \text{(serious CPS expressions)} \\
t & ::= x \;\mid\; v \;\mid\; \lambda x, k.\, e & \text{(trivial CPS expressions)} \\
c & ::= \lambda v.\, e \;\mid\; k & \text{(continuation expressions)}
\end{array}$$

Fig. 2: BNF of CPS language

For brevity we write $\lambda x, k.\, e$ for $\lambda x.\, \lambda k.\, e$. Furthermore we let SExp, TExp, and CExp denote the domain of serious expressions $\mathit{SExp} = \mathcal{L}(e)$, the domain of trivial expressions $\mathit{TExp} = \mathcal{L}(t)$, and the domain of continuation expressions $\mathit{CExp} = \mathcal{L}(c)$, respectively. [3] Slightly misusing notation we will furthermore use e, t, and c (with subscripts and primes) as meta-variables to denote a serious expression, a trivial expression, and a continuation expression, respectively. Their use will be apparent from the context. The language is Turing-complete, in that it is sufficient to express a CPS-version of the Ω-combinator $(\lambda x.\, x\, x\, \lambda y.\, y\, y)$: $\lambda k_0.\, (\lambda x, k_1.\, x\, x\, k_1)\, (\lambda y, k_2.\, y\, y\, k_2)\, k_0$.

4 Semantics

Our starting point semantics will be the CE machine of Flanagan et al. [14]. As opposed to Flanagan et al. we consider only unary functions in the CPS language. Furthermore, our starting-point grammar is a slightly refactored version of the grammar given by Flanagan et al., and the machine description has been refactored accordingly. As a consequence the number of machine transitions is thus cut down to two.

The values and environments of the machine are given in Figure 3a. Values can be either closures, continuation closures, or a special `stop` value that signals a machine halt. We let Val denote the set of values $\mathit{Val} = \mathcal{L}(w)$ and we let Env denote the set of environments $\mathit{Env} = \mathcal{L}(r)$. The environments in Env constitute partial functions, with $\bullet$ being the partial function nowhere defined. We use w and r (with subscripts and primes) as meta-variables to denote a machine value and a machine environment, respectively. Again their use will be apparent from the context. In the spirit of the original CE machines helper function μ, we formulate two helper functions μ_t and μ_c in Figure 3b for evaluating trivial expressions and continuation expressions, respectively.

[3] using the $\mathcal{L}(N)$ notation for the language generated by the non-terminal N.

$$\begin{array}{ll} w ::= [\lambda x, k.\, e,\ r] \ \mid\ [\lambda v.\, e,\ r] \ \mid\ \texttt{stop} & \text{(values)} \\ r\ ::= \bullet \ \mid\ r[x \mapsto w] & \text{(environments)} \end{array}$$

(a) Values and environments

$$\begin{aligned} \mu_t &: \mathit{TExp} \times \mathit{Env} \rightharpoonup \mathit{Val} \\ \mu_t(x, r) &= r(x) \\ \mu_t(v, r) &= r(v) \\ \mu_t(\lambda x, k.\, e, r) &= [\lambda x, k.\, e,\ r] \end{aligned}$$

$$\begin{aligned} \mu_c &: \mathit{CExp} \times \mathit{Env} \rightharpoonup \mathit{Val} \\ \mu_c(k, r) &= r(k) \\ \mu_c(\lambda v.\, e, r) &= [\lambda v.\, e,\ r] \end{aligned}$$

(b) Helper functions

$$\langle t_0\, t_1\ c,\ r\rangle \longrightarrow \langle e,\ r'[x \mapsto w][k \mapsto w_c]\rangle \qquad \text{where}\ \begin{aligned}[t] [\lambda x, k.\, e,\ r'] &= \mu_t(t_0, r) \\ w &= \mu_t(t_1, r) \\ w_c &= \mu_c(c, r) \end{aligned}$$

$$\langle c\, t,\ r\rangle \longrightarrow \langle e,\ r'[v \mapsto w]\rangle \qquad \text{where}\ \begin{aligned}[t] [\lambda v.\, e,\ r'] &= \mu_c(c, r) \\ w &= \mu_t(t, r) \end{aligned}$$

(c) Transition relation

$$\begin{aligned} &\mathit{eval}(\lambda k.\, e) = w \ \text{ iff} \\ &\langle e,\ \bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\ \bullet[k_r \mapsto \texttt{stop}]]]\rangle \longrightarrow^* \langle k_r\ v_r,\ \bullet[k_r \mapsto \texttt{stop}][v_r \mapsto w]\rangle \end{aligned}$$

(d) Machine evaluation

Fig. 3: The CE abstract machine

A machine state is a pair consisting of a serious expression and an environment. The transition relation of the machine is given in Figure 3c. The initial state of the machine binds the initial continuation variable k to a special continuation closure containing a `stop` value. When applied, the special continuation closure will first bind the final result to a special variable v_r, and afterwards attempt to apply the `stop` value (the latter indicating a final state). The machine evaluates CPS programs by repeatedly transitioning from state to state until it is either stuck or in a final state.

5 Collecting semantics

As traditional we consider the reachable states of the transition system as our collecting semantics [5, 10].

$$\begin{aligned} I_{\lambda k.\, e} &= \{\langle e,\ \bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\ \bullet[k_r \mapsto \texttt{stop}]]]\rangle\} && \text{(initial state)} \\ F_{\lambda k.\, e} &: \wp(\mathit{SExp} \times \mathit{Env}) \to \wp(\mathit{SExp} \times \mathit{Env}) \\ F_{\lambda k.\, e}(S) &= I_{\lambda k.\, e} \cup \{s \mid \exists s' \in S : s' \longrightarrow s\} && \text{(strongest post-condition)} \end{aligned}$$

The reachable states semantics is now given as the limit $\bigcup_{n\geq 0} F^n_{\lambda k.\, e}(\emptyset)$. Due to the notational overhead we shall refrain from subscripting with the program at hand from here onwards.

We give an equivalent formulation of the collecting semantics in Figure 4 with helper functions extended to operate on sets of environments. A simple case analysis reveals that $\mu_t^{\wp}$ and $\mu_c^{\wp}$ are monotone in their second argument. By another case analysis one can establish two equivalences between the helper functions $\forall t, R : \forall r \in R : \{w \mid w = \mu_t(t, r)\} = \mu_t^{\wp}(t, \{r\})$ and $\forall c, R : \forall r \in R : \{w \mid w = \mu_c(c, r)\} = \mu_c^{\wp}(c, \{r\})$. A final case analysis establishes the equivalence of the two: $\forall S : \langle e,\ r\rangle \in F(S) \iff \langle e,\ r\rangle \in F_c(S)$.

$$\begin{aligned}
\mu_t^{\wp} &: \mathit{TExp} \times \wp(\mathit{Env}) \to \wp(\mathit{Val}) \\
\mu_t^{\wp}(x, R) &= \{r(x) \mid r \in R\} \\
\mu_t^{\wp}(v, R) &= \{r(v) \mid r \in R\} \\
\mu_t^{\wp}(\lambda x, k.\, e, R) &= \{[\lambda x, k.\, e,\ r] \mid r \in R\}
\end{aligned}
\qquad
\begin{aligned}
\mu_c^{\wp} &: \mathit{CExp} \times \wp(\mathit{Env}) \to \wp(\mathit{Val}) \\
\mu_c^{\wp}(k, R) &= \{r(k) \mid r \in R\} \\
\mu_c^{\wp}(\lambda v.\, e, R) &= \{[\lambda v.\, e,\ r] \mid r \in R\}
\end{aligned}$$

(a) Collecting helper functions

$$\begin{aligned}
F_c &: \wp(\mathit{SExp} \times \mathit{Env}) \to \wp(\mathit{SExp} \times \mathit{Env}) \\
F_c(S) &= I_{\lambda k.\, e} \\
&\quad \cup \{\langle e',\ r'[x \mapsto w][k' \mapsto w_c]\rangle \mid \exists \langle t_0\, t_1\, c,\ r\rangle \in S : \\
&\qquad\qquad [\lambda x, k'.\, e',\ r'] \in \mu_t^{\wp}(t_0, \{r\}) \\
&\qquad\qquad \wedge\ w \in \mu_t^{\wp}(t_1, \{r\}) \\
&\qquad\qquad \wedge\ w_c \in \mu_c^{\wp}(c, \{r\})\} \\
&\quad \cup \{\langle e',\ r'[v \mapsto w]\rangle \mid \exists \langle c\, t,\ r\rangle \in S : \\
&\qquad\qquad [\lambda v.\, e',\ r'] \in \mu_c^{\wp}(c, \{r\}) \\
&\qquad\qquad \wedge\ w \in \mu_t^{\wp}(t, \{r\})\}
\end{aligned}$$

(b) Transition function

Fig. 4: Reachable states collecting semantics

6 Abstracting the collecting semantics

With the collecting semantics in place we are now in position to abstract it. We describe the abstractions for values, environments, and machine states in turn.

6.1 Abstraction of values

Values are abstracted using the elementwise abstraction of Example 2. First, the grammar of abstract values reads as follows.

$$w^{\sharp} ::= [\lambda x, k.\, e] \ \mid\ [\lambda v.\, e] \ \mid\ \texttt{stop} \quad \text{(abstract values)}$$

We let $Val^\sharp = \mathcal{L}(w^\sharp)$ denote the domain of abstract values. Secondly, we define an elementwise operator mapping a concrete value to its abstract counterpart. The operator abstracts away the captured environment component of closure values.

$$\begin{aligned} @ &: Val \to Val^\sharp \\ @([\lambda x, k.\, e,\ r]) &= [\lambda x, k.\, e] \\ @([\lambda v.\, e,\ r]) &= [\lambda v.\, e] \\ @(\mathtt{stop}) &= \mathtt{stop} \end{aligned}$$

6.2 Abstraction of environments

In order to perform environment extension (binding) we need to concretize an environment component from an abstract closure. Unfortunately a straight-forward pointwise extension of the above value abstraction will not suffice: concretization of an abstract closure would return *top* representing *any* environment component. We therefore prefix the pointwise environment abstraction by an abstraction based on a closure operator to ensure that any captured environment in a set of environments belongs to the set itself. We will use a *constituent relation* formulated by Milner and Tofte [23] to express the closure operator.

For a tuple $(x_1, \ldots, x_n)$ each entry x_i is a *constituent* of the tuple. For a partial function $[x_1 \mapsto w_1 \ldots x_n \mapsto w_n]$, each w_i is a *constituent* of the function.[4] We write $x \succ y$ if y is a constituent of x. We denote by $\succ^*$ the reflexive transitive closure of the constituent relation.[5] We can now formulate an appropriate closure operator which induces the first Galois connection.

Definition 1 (Closure operator).

$$\begin{aligned} \rho &: \wp(Env) \to \wp(Env) \\ \rho(R) &= \{r' \in Env \mid \exists r \in R : r \succ^* r'\} \end{aligned}$$

Intuitively, given a set of environments, the closure operator returns a larger set containing all the "enclosed" environments of its argument. For the set of environments in the reachable states semantics the closure operator acts as an identity function, since the set is already closed.

Lemma 2 (ρ is an upper closure operator).
ρ is extensive, monotone, and idempotent.

Proof. The proofs for extensiveness and monotonicity are straightforward. Idempotency follows from extensiveness and transitivity of the $\succ^*$ relation.

Example 6 (Applying ρ). We apply ρ to the singleton environment $\{\bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\ \bullet[k_r \mapsto \mathtt{stop}]]]\}$ originating from the initial state $I_{\lambda k.\, e}$. Besides the element itself, $\bullet[k_r \mapsto \mathtt{stop}]$ is also a constituent environment. Hence

$$\begin{aligned} \rho(\{\bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\ \bullet[k_r \mapsto \mathtt{stop}]]]\}) = \{&\bullet[k_r \mapsto \mathtt{stop}], \\ &\bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\ \bullet[k_r \mapsto \mathtt{stop}]]]\} \end{aligned}$$

[4] Milner and Tofte [23] define the constituent relation on finite maps, whereas we define it for partial functions.

[5] Milner and Tofte [23] instead introduce constituent sequences.

Next we apply the pointwise abstraction from Example 3, based on the value abstraction of Section 6.1.

$$\rho(\wp(Env)) \underset{\alpha_\Pi}{\overset{\gamma_\Pi}{\leftrightarrows}} Env^\sharp \quad \text{where} \quad Env^\sharp = Var \to \wp(Val^\sharp)$$

Strictly speaking this Galois connection applies to the complete lattice $\wp(Env)$, whereas in this case we have a specialized complete lattice $\rho(\wp(Env))$. As the latter is a subset of the former, the definition of α_Π still applies. As for the existing definition of γ_Π its image $\wp(Env)$ does not agree with $\rho(\wp(Env))$. However since α_Π is a CJM it uniquely determines a specialized γ_Π. We leave a direct definition of γ_Π unspecified. With this in mind, we compose the two Galois connections and get another Galois connection: $\wp(Env) \underset{\rho}{\overset{1_{\wp(Env)}}{\leftrightarrows}} \rho(\wp(Env)) \underset{\alpha_\Pi}{\overset{\gamma_\Pi}{\leftrightarrows}} Env^\sharp$.

6.3 Abstraction of machine states

As outlined in Figure 1 the abstraction of machine states is staged in two. We first abstract the independent attributes of the reachable states using Example 4. Next we abstract the pair of sets using Example 5 on the environment abstraction from Section 6.2. The first component, i.e., the set of reachable expressions, is not abstracted, hence instantiated with the identity abstraction.

As traditional [9–11] we consider the *reduced product* of both the abstract domains, i.e., all abstract pairs with an empty expression set or an empty abstract environment implicitly represent *bottom*.

7 Calculating the analysis

We calculate an abstract transition function using a traditional recipe [10], by applying the independent attributes abstraction to the transition function from the collecting semantics. The following lemma determines the result as the best abstraction. It furthermore states the strategy for the calculation of a new transition function when read directionally from left to right. The resulting $F_\times$ appears in Figure 5.

Lemma 3 (Transition function as best abstraction).

$$\forall S : \alpha_\times(F_c(\gamma_\times(S))) = F_\times(S)$$

Proof. Let $S = \langle E, R\rangle$ be given. Since $\alpha_\times$ is a complete join morphism, it distributes onto the three sets in F_c's definition. We consider the case of the last of the three sets:

$$\begin{aligned}
&\alpha_\times(\{\langle e', r'[v \mapsto w]\rangle \mid \exists \langle c\ t, r\rangle \in \gamma_\times(\langle E, R\rangle) : \\
&\qquad\qquad\qquad [\lambda v.\, e', r'] \in \mu_c^\wp(c, \{r\}) \\
&\qquad\qquad\qquad \wedge\ w \in \mu_t^\wp(t, \{r\})\}) \\
&= \alpha_\times(\bigcup_{\substack{\langle c\ t, r\rangle \in \gamma_\times(\langle E, R\rangle) \\ [\lambda v.\, e', r'] \in \mu_c^\wp(c,\{r\}) \\ w \in \mu_t^\wp(t,\{r\})}} \{\langle e', r'[v \mapsto w]\rangle\}) \qquad \text{(formulate as join)}
\end{aligned}$$

$$= \bigcup\nolimits_{\times \, \substack{\langle c\ t,\, r\rangle \in \gamma_\times(\langle E,\, R\rangle) \\ [\lambda v.\, e',\, r'] \in \mu_c^\wp(c,\{r\}) \\ w \in \mu_t^\wp(t,\{r\})}} \alpha_\times(\{\langle e',\, r'[v \mapsto w]\rangle\}) \qquad (\alpha_\times \text{ a CJM})$$

using the definitions of $\alpha_\times$ and $\gamma_\times$ we arrive at the last set of $F_\times$'s definition. □

For a given program $\lambda k.\, e$:

$$F_\times : \wp(SExp) \times \wp(Env) \to \wp(SExp) \times \wp(Env)$$
$$F_\times(\langle E,\, R\rangle) = \langle\{e\},\, \{\bullet[k \mapsto [\lambda v_r.\, k_r\ v_r,\, \bullet[k_r \mapsto \texttt{stop}]]]\}\rangle$$
$$\cup_\times \bigcup\nolimits_{\times \, \substack{t_0\ t_1\ c \in E \quad r \in R \\ [\lambda x, k'.\, e',\, r'] \in \mu_t^\wp(t_0,\{r\}) \\ w \in \mu_t^\wp(t_1,\{r\}) \\ w_c \in \mu_c^\wp(c,\{r\})}} \langle\{e'\},\, \{r'[x \mapsto w][k' \mapsto w_c]\}\rangle$$
$$\cup_\times \bigcup\nolimits_{\times \, \substack{c\ t \in E \quad r \in R \\ [\lambda v.\, e',\, r'] \in \mu_c^\wp(c,\{r\}) \\ w \in \mu_t^\wp(t,\{r\})}} \langle\{e'\},\, \{r'[v \mapsto w]\}\rangle$$

Fig. 5: Independent attributes transition function

Next we abstract the pair of sets by an abstract pair again following a recipe. We first calculate abstract helper functions $\mu_t^\sharp$ and $\mu_c^\sharp$. By construction they satisfy the following lemma. Furthermore the lemma states the strategy for the calculations when read directionally from left to right. The calculations proceed by case analysis.

Lemma 4 (Equivalence of helper functions).

$$\forall t, R : \alpha_@(\mu_t^\wp(t, R)) = \mu_t^\sharp(t, \alpha_\Pi(R)) \ \wedge\ \forall c, R : \alpha_@(\mu_c^\wp(c, R)) = \mu_c^\sharp(c, \alpha_\Pi(R))$$

Another case analysis reveals that $\mu_t^\sharp$ and $\mu_c^\sharp$ are monotone in their second argument. By a third case analysis one can prove the following lemma concerning applying helper functions to "closed" environments.

Lemma 5 (Helper functions on closed environments).

$$\forall t, R, w : w \in \mu_t^\wp(t, \rho(R)) \implies \{r \mid w \succ^* r\} \subseteq \rho(R)$$
$$\forall c, R, w : w \in \mu_c^\wp(c, \rho(R)) \implies \{r \mid w \succ^* r\} \subseteq \rho(R)$$

Finally we need a lemma formulating abstraction of an extended environment in terms of the abstraction of its subparts.

Lemma 6 (Abstraction of environment extension).

$$\forall v, w, R : \{r \mid w \succ^* r\} \subseteq \rho(R)$$
$$\implies \alpha_\Pi \circ \rho(\{r[v \mapsto w] \mid r \in \rho(R)\}) \mathrel{\dot\subseteq} \alpha_\Pi \circ \rho(R)[v \mapsto \alpha_@(\{w\})]^\sharp$$

where we have used the shorthand notation $R^\sharp[v \mapsto \{\ldots\}]^\sharp = R^\sharp \,\dot\cup\, \dot\emptyset[v \mapsto \{\ldots\}]$. We omit the proof due to lack of space.

We are now in position to calculate the abstract transition function. By construction, the abstract transition function satisfies the following lemma. Again, when read directionally from left to right, the lemma states the strategy for the calculation. The resulting analysis appears in Figure 6.

Lemma 7. $\forall S : \alpha_\otimes(F_\times(S)) \subseteq_\otimes F^\sharp(\alpha_\otimes(S))$

Proof. Let $S = \langle E,\, R\rangle$ be given. Again $\alpha_\otimes$ is a complete join morphism and hence distributes onto the three sets from $F_\times$'s definition. We consider the case of the last of the three sets. First observe

$$\begin{aligned}
& r \in R \wedge [\lambda v.\, e',\, r'] \in \mu_c^\wp(c, \{r\}) \wedge w \in \mu_t^\wp(t, \{r\}) \\
\Longrightarrow\ & r \in \rho(R) \wedge [\lambda v.\, e',\, r'] \in \mu_c^\wp(c, \{r\}) \wedge w \in \mu_t^\wp(t, \{r\}) && (\rho \text{ extensive}) \\
\Longrightarrow\ & [\lambda v.\, e',\, r'] \in \mu_c^\wp(c, \rho(R)) \wedge w \in \mu_t^\wp(t, \rho(R)) && (\mu_c^\wp,\, \mu_t^\wp \text{ monotone}) \\
\Longrightarrow\ & r' \in \rho(R) \wedge [\lambda v.\, e',\, r'] \in \mu_c^\wp(c, \rho(R)) \wedge w \in \mu_t^\wp(t, \rho(R)) && (\text{by Lemma 5}) \\
\Longrightarrow\ & r' \in \rho(R) \wedge \alpha_@(\{[\lambda v.\, e',\, r']\}) \subseteq \alpha_@(\mu_c^\wp(c, \rho(R))) \wedge w \in \mu_t^\wp(t, \rho(R)) \\
& && (\alpha_@ \text{ monotone}) \\
\Longleftrightarrow\ & r' \in \rho(R) \wedge [\lambda v.\, e'] \in \alpha_@(\mu_c^\wp(c, \rho(R))) \wedge w \in \mu_t^\wp(t, \rho(R)) && (\text{def. of } \alpha_@) \\
\Longleftrightarrow\ & r' \in \rho(R) \wedge [\lambda v.\, e'] \in \mu_c^\sharp(c, \alpha_\Pi \circ \rho(R)) \wedge w \in \mu_t^\wp(t, \rho(R)) && (\text{by Lemma 4})
\end{aligned}$$

Secondly observe if $w \in \mu_t^\wp(t, \rho(R))$ then

$$\begin{aligned}
& \alpha_\otimes\Big(\bigcup\nolimits_{\times}\limits_{r' \in \rho(R)} \langle \{e'\},\, \{r'[v \mapsto w]\}\rangle \Big) \\
&= \alpha_\otimes(\langle \{e'\},\, \{r'[v \mapsto w] \mid r' \in \rho(R)\}\rangle) && (\text{def. } \cup_\times) \\
&= \langle \{e'\},\, \alpha_\Pi \circ \rho(\{r'[v \mapsto w] \mid r' \in \rho(R)\})\rangle && (\text{def. } \alpha_\otimes) \\
&\subseteq_\otimes \langle \{e'\},\, \alpha_\Pi \circ \rho(R)[v \mapsto \alpha_@(\{w\})]^\sharp\rangle && (\text{by Lemma 5, 6})
\end{aligned}$$

Thirdly observe

$$\begin{aligned}
& \bigcup\nolimits_{\otimes}\limits_{w \in \mu_t^\wp(t, \rho(R))} \langle \{e'\},\, \alpha_\Pi \circ \rho(R)[v \mapsto \alpha_@(\{w\})]^\sharp\rangle \\
&= \langle \{e'\},\, \dot{\bigcup_{w \in \mu_t^\wp(t, \rho(R))}} \alpha_\Pi \circ \rho(R)[v \mapsto \alpha_@(\{w\})]^\sharp\rangle && (\text{def. } \cup_\otimes) \\
&= \langle \{e'\},\, \alpha_\Pi \circ \rho(R) \,\dot\cup\, \dot{\bigcup_{w \in \mu_t^\wp(t, \rho(R))}} \dot\emptyset[v \mapsto \alpha_@(\{w\})]\rangle && (\text{def. } -[-]^\sharp) \\
&= \langle \{e'\},\, \alpha_\Pi \circ \rho(R)[v \mapsto \bigcup_{w \in \mu_t^\wp(t, \rho(R))} \alpha_@(\{w\})]^\sharp\rangle && (\text{def. } \dot\cup) \\
&= \langle \{e'\},\, \alpha_\Pi \circ \rho(R)[v \mapsto \alpha_@(\mu_t^\wp(t, \rho(R)))]^\sharp\rangle && (\alpha_@ \text{ a CJM}) \\
&= \langle \{e'\},\, \alpha_\Pi \circ \rho(R)[v \mapsto \mu_t^\sharp(t, \alpha_\Pi \circ \rho(R))]^\sharp\rangle && (\text{by Lemma 4})
\end{aligned}$$

Hence

$$\begin{aligned}
&\alpha_{\otimes}(\bigcup_{\times\; \substack{c\,t \in E \quad r \in R \\ [\lambda v.\, e',\, r'] \in \mu^{\wp}_{c}(c,\{r\}) \\ w \in \mu^{\wp}_{t}(t,\{r\})}} \langle \{e'\},\, \{r'[v \mapsto w]\} \rangle) \\
&\subseteq_{\otimes} \alpha_{\otimes}(\bigcup_{\times\; \substack{c\,t \in E \quad r' \in \rho(R) \\ [\lambda v.\, e'] \in \mu^{\sharp}_{c}(c,\alpha_{\Pi}\circ\rho(R)) \\ w \in \mu^{\wp}_{t}(t,\rho(R))}} \langle \{e'\},\, \{r'[v \mapsto w]\} \rangle) && \text{(first obs.)} \\
&= \bigcup_{\otimes\; \substack{c\,t \in E \\ [\lambda v.\, e'] \in \mu^{\sharp}_{c}(c,\alpha_{\Pi}\circ\rho(R)) \\ w \in \mu^{\wp}_{t}(t,\rho(R))}} \alpha_{\otimes}(\bigcup_{\times\; r' \in \rho(R)} \langle \{e'\},\, \{r'[v \mapsto w]\} \rangle) && (\alpha_{\otimes} \text{ a CJM}) \\
&\subseteq_{\otimes} \bigcup_{\otimes\; \substack{c\,t \in E \\ [\lambda v.\, e'] \in \mu^{\sharp}_{c}(c,\alpha_{\Pi}\circ\rho(R)) \\ w \in \mu^{\wp}_{t}(t,\rho(R))}} \langle \{e'\},\, \alpha_{\Pi} \circ \rho(R)[v \mapsto \alpha_{@}(\{w\})]^{\sharp} \rangle && \text{(second obs.)} \\
&= \bigcup_{\otimes\; \substack{c\,t \in E \\ [\lambda v.\, e'] \in \mu^{\sharp}_{c}(c,\alpha_{\Pi}\circ\rho(R))}} \langle \{e'\},\, \alpha_{\Pi} \circ \rho(R)[v \mapsto \mu^{\sharp}_{t}(t, \alpha_{\Pi} \circ \rho(R))]^{\sharp} \rangle && \text{(third obs.)}
\end{aligned}$$

Since $\alpha_{\otimes}(S) = \langle E,\, \alpha_{\Pi} \circ \rho(R) \rangle$ we define the third set of $F^{\sharp}$ as this set (with $R^{\sharp}$ for $\alpha_{\Pi} \circ \rho(R)$). By construction $\alpha_{\otimes}(F_{\times}(S)) \subseteq_{\otimes} F^{\sharp}(\alpha_{\otimes}(S))$ holds. $\square$

This result in turn enables us to prove the following (standard) theorem stating the correctness of the analysis [8].

Theorem 1 (Fixed-point transfer theorem). $\alpha_{\otimes} \circ \alpha_{\times}(\text{lfp}\, F_c) \subseteq \text{lfp}\, F^{\sharp}$

The resulting analysis in Figure 6 is striking. By an independent attribute abstraction of a standard collecting semantics, we have encountered a demand-driven CFA. Demand-driven CFA has been discovered independently [2, 3, 15], and is usually presented as an extension (or improvement) to 0-CFA. Our result on the other hand explains it as a natural abstraction of a reachable states collection semantics.

Expressing a semantics for a CPS language as a transition system lends itself to abstract interpretation as originally expressed by Cousot [5]. Since all intermediate results in CPS are already named, i.e., bound to an identifier, a control-flow analysis merely becomes a question of computing an abstract environment. As both the continuations and closures live in the environment there is no need for an explicit stack, as it lives as a chain of closures in the environment. We have found no need to introduce new concepts such as *labels* or *caches* [4,27].

8 Implementation and example

We have implemented a prototype of the derived analysis in OCaml.[6] The core of the algorithm constitutes 60 lines of source code. To illustrate the 0-CFA

[6] available at `<http://www.brics.dk/~jmi/Midtgaard-Jensen:SAS08/>`

$$\mu_t^\sharp : TExp \times Env^\sharp \rightarrow \wp(Val^\sharp)$$
$$\mu_t^\sharp(x, R^\sharp) = R^\sharp(x)$$
$$\mu_t^\sharp(v, R^\sharp) = R^\sharp(v)$$
$$\mu_t^\sharp(\lambda x, k.\, e, R^\sharp) = \{[\lambda x, k.\, e]\}$$

$$\mu_c^\sharp : CExp \times Env^\sharp \rightarrow \wp(Val^\sharp)$$
$$\mu_c^\sharp(k, R^\sharp) = R^\sharp(k)$$
$$\mu_c^\sharp(\lambda v.\, e, R^\sharp) = \{[\lambda v.\, e]\}$$

(a) Abstract helper functions

For a given program $\lambda k.\, e$:

$$F^\sharp : \wp(SExp) \times Env^\sharp \rightarrow \wp(SExp) \times Env^\sharp$$
$$\begin{aligned} F^\sharp(\langle E^\sharp,\, R^\sharp\rangle) = {} & \langle \{e\},\, \dot{\emptyset}[k_r \mapsto \{\texttt{stop}\}, k \mapsto \{[\lambda v_r.\, k_r\, v_r]\}]^\sharp\rangle \\ & \cup_\otimes \bigcup_{\substack{t_0\ t_1\ c \,\in\, E^\sharp \\ [\lambda x, k'.\, e'] \,\in\, \mu_t^\sharp(t_0, R^\sharp)}}{}_{\otimes} \langle \{e'\},\, R^\sharp[x \mapsto \mu_t^\sharp(t_1, R^\sharp), k' \mapsto \mu_c^\sharp(c, R^\sharp)]^\sharp\rangle \\ & \cup_\otimes \bigcup_{\substack{c\ t \,\in\, E^\sharp \\ [\lambda v.\, e'] \,\in\, \mu_c^\sharp(c, R^\sharp)}}{}_{\otimes} \langle \{e'\},\, R^\sharp[v \mapsto \mu_t^\sharp(t, R^\sharp)]^\sharp\rangle \end{aligned}$$

(b) Abstract transition function

Fig. 6: Demand-driven 0-CFA

nature of the analysis we recall an example from Nielson, Nielson, and Hankin's textbook [26]: `let f = (fn x => x) in f f (fn y => y)`. The expression $\lambda k_0.\, (\lambda f, k_2.\, f f\, (\lambda v_4.\, v_4\, (\lambda y, k_5.\, k_5\, y)\, k_2))\, (\lambda x, k_6.\, k_6\, x)\, k_0$ is a CPS version of the same example. After 8 iterations the analysis reaches a fixed point determining that all serious expressions of the example are reachable. The inferred abstract environment is given in Figure 7. Just as the textbook 0-CFA the derived analysis merges all bindings to x, which affects the final answer v_r, and results in the overly approximate answer of two abstract closures.

$k_r \mapsto \{\texttt{stop}\}$	$f \mapsto \{[\lambda x, k_6.\, k_6\, x]\}$
$k_0, k_2, k_5 \mapsto \{[\lambda v_r.\, k_r\, v_r]\}$	$y \mapsto \{[\lambda y, k_5.\, k_5\, y]\}$
$k_6 \mapsto \{[\lambda v_4.\, v_4\, (\lambda y, k_5.\, k_5\, y)\, k_2], [\lambda v_r.\, k_r\, v_r]\}$	$x, v_4, v_r \mapsto \{[\lambda x, k_6.\, k_6\, x], [\lambda y, k_5.\, k_5\, y]\}$

Fig. 7: Inferred abstract environment

9 Related work

The only existing demand-driven 0-CFA for a CPS language that we are aware of is that of Ayers [2], who used Galois connections to express the correctness of 0-CFA for a CPS language. After formally proving his 0-CFA correct he suggests a number of improvements. One of these is *use-maps*, the idea of which is to only re-analyse parts of the program where recent additions will have an effect on the fixed-point computation. A later refinement of use-maps incorporates reachability, resulting in an algorithm which will only re-analyse *reachable* parts of the program where recent additions will have an effect. Ayers's work differs from our result in that: (a) it does not use an off-the-shelf starting point,[7] (b) it does not use off-the-shelf Galois connections, and (c) reachability is added afterwards as an extension (but not formally proved, e.g., expressed with Galois connections).

Cousot and Cousot have championed the calculational approach to program analysis for three decades [6, 8–11]. Cousot has provided a comprehensive set of lecture notes [6], in which he calculates various abstract interpreters for a simple imperative language. Nevertheless, the calculational approach to control-flow analysis of functional programs has received little attention so far.

Shivers [22, 31, 32] has long argued that basing a CFA on a CPS language simplifies matters as it captures all control flow in one unifying construct. In his thesis [32] he developed control-flow analyses for Scheme including (control and state) side effects. Shivers [32] did not consider demand-driven analysis, nor formulate correctness using Galois connections. Initially the development was based on an instrumented denotational semantics, however the more recent work with Might is based on instrumented abstract machines [22]. In contrast we have developed an analysis starting from a well-known and non-instrumented abstract machine.

Sabry and Felleisen [28] have formulated interpreters and corresponding program analysers for languages in direct style and CPS to compare formally their output when run on equivalent input. Their analyses are formulated as inference rules, and as such an analysis may diverge when implemented directly. They therefore detect loops in the analyser and return top when encountering one. In contrast our calculated analysis needs no such ad-hoc modifications. For a further discussion of related work we refer to a recent survey by the first author [21].

10 Conclusion and further work

To the best of our knowledge we have given the first calculated 0-CFA derivation. The calculations reveal a strikingly simple derivation of a demand-driven 0-CFA, a variant which has been discovered independently. Our derivation spells out the approximation by expressing it as a combination of several known Galois connections, thereby capturing the essence of the CFA approximation as an independent attributes abstraction. We have derived the analysis from the reachable states of a well-known abstract machine without resorting to instrumentation.

Cousot and Cousot [11] have pointed out several alternative abstractions to sets of pairs, one of which is a *pointwise coding*. When applied to the states of the CE machine the abstraction may be the key to calculating a *flow-sensitive* CFA. We plan to investigate such a calculation. A natural next step is to consider the calculation of the *context-sensitive* k-CFA hierarchy. Finally it would be

[7] though to be fair, our starting point, the CE machine in Flanagan et al. [14], and Ayers's thesis are both from 1993

interesting to investigate whether proof assistants can aid in the calculation of future analyses.

Acknowledgements: The authors thank Tiphaine Turpin, Mads Sig Ager, Tristan Le Gall, David Pichardie, and the anonymous referees for their comments on this paper.

References

[1] A. W. Appel. *Compiling with Continuations.* Cambridge University Press, New York, 1992.

[2] A. E. Ayers. *Abstract Analysis and Optimization of Scheme.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, Sept. 1993.

[3] S. K. Biswas. A demand-driven set-based analysis. In Jones [17], pages 372–385.

[4] A. Bondorf. Automatic autoprojection of higher-order recursive equations. *Science of Computer Programming*, 17(1-3):3–34, 1991.

[5] P. Cousot. Semantic foundations of program analysis. In Muchnick and Jones [24], chapter 10, pages 303–342.

[6] P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design.* NATO ASI Series F. IOS Press, Amsterdam, 1999.

[7] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1–2):47–103, 2002.

[8] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. Sethi, editor, *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, Jan. 1977.

[9] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In B. K. Rosen, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, Jan. 1979.

[10] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, Aug. 1992.

[11] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In H. Bal, editor, *Proceedings of the Fifth IEEE International Conference on Computer Languages*, pages 95–112, Toulouse, France, May 1994.

[12] O. Danvy, B. Dzafic, and F. Pfenning. On proving syntactic properties of CPS programs. In *Third International Workshop on Higher-Order Operational Techniques in Semantics*, volume 26 of *Electronic Notes in Theoretical Computer Science*, pages 19–31, Paris, France, Sept. 1999.

[13] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, England, second edition, 2002.

[14] C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The essence of compiling with continuations. In D. W. Wall, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Languages Design and Implementation*, pages 237–247, Albuquerque, New Mexico, June 1993.

[15] K. L. S. Gasser, F. Nielson, and H. R. Nielson. Systematic realisation of control flow analyses for CML. In M. Tofte, editor, *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 38–51, Amsterdam, The Netherlands, June 1997.

[16] N. D. Jones. Flow analysis of lambda expressions (preliminary version). In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 114–128, London, UK, 1981.

[17] N. D. Jones, editor. *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Programming Languages*, Paris, France, Jan. 1997.

[18] N. D. Jones and S. S. Muchnick. Complexity of flow analysis, inductive assertion synthesis and a language due to Dijkstra. In Muchnick and Jones [24], pages 380–393.

[19] N. D. Jones and F. Nielson. Abstract interpretation: a semantics-based tool for program analysis. In *Handbook of logic in computer science*, volume 4, pages 527–636. Oxford University Press, 1995.

[20] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[21] J. Midtgaard. Control-flow analysis of functional programs. Technical Report BRICS RS-07-18, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, Dec. 2007.

[22] M. Might and O. Shivers. Improving flow analyses via ΓCFA: abstract garbage collection and counting. In J. Lawall, editor, *Proceedings of the Eleventh ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*, pages 13–25, Portland, Oregon, Sept. 2006.

[23] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87(1):209–220, 1991.

[24] S. S. Muchnick and N. D. Jones, editors. *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.

[25] F. Nielson and H. R. Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In Jones [17], pages 332–345.

[26] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[27] J. Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 17(1):47–62, 1995.

[28] A. Sabry and M. Felleisen. Is continuation-passing useful for data flow analysis? In V. Sarkar, editor, *Proceedings of the ACM SIGPLAN'94 Conference on Programming Languages Design and Implementation*, pages 1–12, Orlando, Florida, June 1994.

[29] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, Inc., 1986.

[30] P. Sestoft. Replacing function parameters by global variables. Master's thesis, DIKU, Computer Science Department, University of Copenhagen, Copenhagen, Denmark, Oct. 1988.

[31] O. Shivers. Control-flow analysis in Scheme. In M. D. Schwartz, editor, *Proceedings of the ACM SIGPLAN'88 Conference on Programming Languages Design and Implementation*, pages 164–174, Atlanta, Georgia, June 1988.

[32] O. Shivers. *Control-Flow Analysis of Higher-Order Languages or Taming Lambda*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1991. Technical Report CMU-CS-91-145.