HAL
open science

Control-flow analysis of function calls and returns by abstract interpretation

Jan Midtgaard, Thomas Jensen

To cite this version:

Jan Midtgaard, Thomas Jensen. Control-flow analysis of function calls and returns by abstract interpretation. Information and Computation, 2012, 2012, pp.49-76. 10.1016/j.ic.2011.11.005 . hal-00758152

HAL Id: hal-00758152

https://inria.hal.science/hal-00758152

Submitted on 29 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Control-Flow Analysis of Function Calls and Returns by Abstract Interpretation

Jan Midtgaard[a], Thomas P. Jensen[b]

[a] *Department of Computer Science*
Aarhus University
Åbogade 34, DK-8200 Aarhus N, Denmark
[b] *INRIA*
Campus de Beaulieu
F-35042 Rennes, France

Abstract

Abstract interpretation techniques are used to derive a control-flow analysis for a simple higher-order functional language. The analysis approximates the interprocedural control-flow of both function calls and returns in the presence of first-class functions and tail-call optimization. In addition to an abstract environment, the analysis computes for each expression an abstract call-stack, effectively approximating where function calls return. The analysis is systematically derived by abstract interpretation of the stack-based C_aEK abstract machine of Flanagan et al. using a series of Galois connections. We prove that the analysis is equivalent to an analysis obtained by first transforming the program into continuation-passing style and then performing control flow analysis of the transfored program. We then show how the analysis induces an equivalent constraint-based formulation, thereby providing a rational reconstruction of a constraint-based CFA from abstract interpretation principles.

1. Introduction

Control-flow analysis (CFA) of functional programs is concerned with determining how the program's functions call each other. In the case of the lambda calculus, this amounts to computing the flow of lambda expressions in order to determine what functions are effectively called in an application $(e_1\ e_2)$. The result of a CFA can be visualized as an oriented *control flow graph* (CFG) linking sub-expression e_i to sub-expression e_j if evaluation of e_i may entail the immediate evaluation of e_j. A CFA computes an approximation of the actual behaviour of the program and can be more or less accurate depending on the technique employed.

Email addresses: `jmi@cs.au.dk` (Jan Midtgaard), `Thomas.Jensen@inria.fr` (Thomas P. Jensen)

In his seminal work, Jones [1, 2] proposed to use program analysis techniques to statically approximate the flow of lambda-expressions under both call-by-value and call-by-name evaluation in the lambda calculus. Since then CFA has been the subject of an immense research effort [3, 4, 5, 6]—see the recent survey by Midtgaard [7] for a complete list. CFA has been expressed using a variety of formalisms including data flow equations, type systems and constraint-based analysis. Surprisingly, nobody has employed Cousot's programme of *calculational abstract interpretation* [8] in which a program analysis is calculated by systematically applying abstraction functions to a formal programming language semantics. The purpose of this article is to show that such a derivation is indeed feasible and that a number of advantages follow from taking this approach:

- The systematic derivation of a CFA for a higher-order functional language from a well-known operational semantics provides the resulting analysis with strong mathematical foundations. Its correctness follows directly from the general theorems of abstract interpretation.
- The approach is easily adapted to different variants of the source language. We demonstrate this by deriving a CFA for functional programs written in continuation-passing style.
- The common framework of these analyses enables their comparison. We take advantage of this to settle a question about the equivalence between the analysis of programs in direct and continuation-passing style.
- The resulting equations can be given an equivalent constraint-based presentation, providing *ipso facto* a rational reconstruction and a correctness proof of constraint-based CFA.

The article is organized as follows. Section 2 provides a concise enumeration of fundamental notions from abstract interpretation used in the rest of the article. In Section 3 we define the language of study (the lambda calculus in administrative normal form) and its semantics, and give an example of CFA of programs written in this language. Sections 4 and 5 contain the derivation of a 0-CFA from an operational semantics: the C_aEK machine of Flanagan *et al.* [9]. In Section 4 we define a series of Galois connections that each specifies one aspect of the abstraction in the analysis. In Section 5 we calculate the analysis as the result of composing the collecting semantics induced by the abstract machine with these Galois connections. Section 6 uses the same technical machinery to derive a CFA for a language in continuation-passing style and sets up a relation between the two abstract domains that enables to prove a lock-step equivalence of the analysis of programs in direct style and the CPS analysis of their CPS counterparts. In Section 7 we show how the recursive equations defining the CFA of a program induce an equivalent formulation of the analysis, where the result of the analysis now is expressed as a solution to a set of constraints. Section 8 compares with related approaches and Section 9 concludes.

Preliminary versions of the results reported in this article were published at SAS 2008 [10] and ICFP 2009 [11]. The present article is a revised version

of the latter paper, extending the lock-step relation between the direct and continuation-passing style analyses to include integer constants. The paper has furthermore been expanded with proofs and details of derivations of the abstract interpretations.

2. Abstract interpretation

This section recalls basic notions of lattice theory and abstract interpretation [12, 13, 14, 15, 16] on which we base our developments in the subsequent sections. In particular, we introduce the notion of *Galois connections* and provide a list of known Galois connections that will be used to design the abstraction underlying the CFA developed in Section 4.

A partially ordered set (poset) $\langle S; \sqsubseteq \rangle$ is a set S equipped with a partial order $\sqsubseteq$. A complete lattice is a poset $\langle C; \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$, such that the least upper bound $\sqcup S$ and the greatest lower bound $\sqcap S$ exists for every subset S of C. $\bot = \sqcap C$ denotes the infimum of C and $\top = \sqcup C$ denotes the supremum of C. The set of total functions $D \to C$, whose codomain is a complete lattice $\langle C; \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$, is itself a complete lattice $\langle D \to C; \dot{\sqsubseteq}, \dot{\bot}, \dot{\top}, \dot{\sqcup}, \dot{\sqcap} \rangle$ under the pointwise ordering $f \,\dot{\sqsubseteq}\, f' \iff \forall x. f(x) \sqsubseteq f'(x)$, and with bottom, top, join, and meet extended similarly. The powersets $\wp(S)$ of a set S ordered by set inclusion is a complete lattice $\langle \wp(S); \subseteq, \emptyset, S, \cup, \cap \rangle$.

2.1. Galois connections

A Galois connection is a pair of functions α, γ between two posets $\langle C; \sqsubseteq \rangle$ and $\langle A; \leq \rangle$ such that for all $a \in A, c \in C : \alpha(c) \leq a \iff c \sqsubseteq \gamma(a)$. Equivalently a Galois connection can be defined as a pair of functions satisfying

(a) α and γ are monotone.

(b) $\alpha \circ \gamma$ is reductive (for all $a \in A : \alpha \circ \gamma(a) \leq a$),

(c) $\gamma \circ \alpha$ is extensive (for all $c \in C : c \sqsubseteq \gamma \circ \alpha(c)$).

Galois connections are typeset as $\langle C; \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A; \leq \rangle$. We omit the orderings when they are clear from the context. For a Galois connection between two complete lattices $\langle C; \sqsubseteq, \bot_c, \top_c, \sqcup, \sqcap \rangle$ and $\langle A; \leq, \bot_a, \top_a, \vee, \wedge \rangle$, α is a complete join-morphism (CJM) (for all $S_c \subseteq C : \alpha(\sqcup S_c) = \vee\alpha(S_c) = \vee\{\alpha(c) \mid c \in S_c\}$) and γ is a complete meet morphism (for all $S_a \subseteq A : \gamma(\wedge S_a) = \sqcap\gamma(S_a) = \sqcap\{\gamma(a) \mid a \in S_a\}$). The composition of two Galois connections $\langle C; \sqsubseteq \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle B; \subseteq \rangle$ and $\langle B; \subseteq \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle A; \leq \rangle$ is itself a Galois connection $\langle C; \sqsubseteq \rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle A; \leq \rangle$. Galois connections in which α is surjective (or equivalently γ is injective) are typeset as: $\langle C; \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A; \leq \rangle$. Galois connections in which γ is surjective (or equivalently α is injective) are typeset as: $\langle C; \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A; \leq \rangle$. When both α and γ are surjective, the two domains are isomorphic.

The following Galois connections will be used in the article:

Elementwise Abstraction [17]:. A function $@ : C \to A$ induces a Galois connection:

$$\langle \wp(C); \subseteq \rangle \underset{\alpha_@}{\overset{\gamma_@}{\leftrightarrows}} \langle \wp(A); \subseteq \rangle$$
$$\alpha_@(P) = \{@(p) \mid p \in P\}$$
$$\gamma_@(Q) = \{p \mid @(p) \in Q\}$$

Cartesian abstraction [15]. One can approximate a set of tuples by a tuple of sets, by projecting out each component into a separate set:

$$\langle \wp(C_1 \times \ldots \times C_n); \subseteq \rangle \leftrightarrows \langle \wp(C_1) \times \ldots \times \wp(C_n); \subseteq_\times \rangle$$
$$\alpha_\times(r) = \langle \pi_1(r), \ldots, \pi_n(r) \rangle$$
$$\gamma_\times(\langle X_1, \ldots, X_n \rangle) = X_1 \times \ldots \times X_n$$

Componentwise Abstraction [15]:. Assuming a series of Galois connections: $\wp(C_i) \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} A_i$ for $i \in \{1, \ldots, n\}$, their componentwise composition induces a Galois connection on tuples:

$$\langle \wp(C_1) \times \ldots \times \wp(C_n); \subseteq_\times \rangle \underset{\alpha_\otimes}{\overset{\gamma_\otimes}{\leftrightarrows}} \langle A_1 \times \ldots \times A_n; \subseteq_\otimes \rangle$$
$$\alpha_\otimes(\langle X_1, \ldots, X_n \rangle) = \langle \alpha_1(X_1), \ldots, \alpha_n(X_n) \rangle$$
$$\gamma_\otimes(\langle x_1, \ldots, x_n \rangle) = \langle \gamma_1(x_1), \ldots, \gamma_n(x_n) \rangle$$

We write $\cup_\otimes$ and $\subseteq_\otimes$ for componentwise join and inclusion, respectively.

Pointwise Encoding of a Relation [15]:. A relation can be isomorphically encoded as a set-valued function by a Galois connection:

$$\langle \wp(A \times B); \subseteq \rangle \underset{\alpha_\omega}{\overset{\gamma_\omega}{\leftrightarrows}} \langle A \to \wp(B); \dot{\subseteq} \rangle$$
$$\alpha_\omega(r) = \lambda a.\, \{b \mid \langle a, b \rangle \in r\}$$
$$\gamma_\omega(f) = \{\langle a, b \rangle \mid b \in f(a)\}$$

Pointwise Abstraction of a Set of Functions [15]:. A given Galois connection on the co-domain $\langle \wp(C); \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle C^\sharp; \sqsubseteq \rangle$ induces a Galois connection on a set of functions:

$$\langle \wp(D \to C); \subseteq \rangle \underset{\alpha_\Pi}{\overset{\gamma_\Pi}{\leftrightarrows}} \langle D \to C^\sharp; \dot{\sqsubseteq} \rangle$$
$$\alpha_\Pi(F) = \lambda d.\, \alpha(\{f(d) \mid f \in F\})$$
$$\gamma_\Pi(A) = \{f \mid \forall d : f(d) \in \gamma(A(d))\}$$

Subset Abstraction [17]:. Given a set C and a strict subset $A \subset C$ hereof, the restriction to the subset induces a Galois connection:

$$\langle \wp(C); \subseteq \rangle \underset{\alpha_\subset}{\overset{\gamma_\subset}{\leftrightarrows}} \langle \wp(A); \subseteq \rangle$$
$$\alpha_\subset(X) = X \cap A$$
$$\gamma_\subset(Y) = Y \cup (C \setminus A)$$

A(n upper) closure operator ρ is map $\rho : S \to S$ on a poset $\langle S; \sqsubseteq \rangle$, that is (a) monotone: (for all $s, s' \in S : s \sqsubseteq s' \implies \rho(s) \sqsubseteq \rho(s')$), (b) extensive (for all $s \in S : s \sqsubseteq \rho(s)$), and (c) idempotent, (for all $s \in S : \rho(s) = \rho(\rho(s))$). A closure operator ρ induces a Galois connection $\langle S; \sqsubseteq \rangle \underset{\rho}{\overset{1}{\leftrightarrows}} \langle \rho(S); \sqsubseteq \rangle$, writing $\rho(S)$ for $\{\rho(s) \mid s \in S\}$ and 1 for the identity function. Furthermore the image of a complete lattice $\langle C; \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ by an upper closure operator is itself a complete lattice $\langle \rho(C); \sqsubseteq, \rho(\bot), \top, \lambda X.\, \rho(\sqcup X), \sqcap \rangle$.

2.2. Abstract interpretation basics

Canonical abstract interpretation approximates the *collecting semantics* of a transition system [13]. A standard example of a collecting semantics is the *reachable states* from a given set of initial states I. Given a transition function T defined as:

$$T(\Sigma) = I \cup \{\sigma \mid \exists \sigma' \in \Sigma : \sigma' \to \sigma\}$$

we can compute the reachable states of T as the least fixed-point $\operatorname{lfp} T$ of T. The collecting semantics is ideal, in that it is the most precise analysis. Unfortunately it is in general uncomputable. Abstract interpretation therefore approximates the collecting semantics, by instead computing a fixed-point over an alternative and perhaps simpler domain. For this reason, abstract interpretation is also referred to as a theory of fixed-point approximation.

Abstractions are formally represented as Galois connections which connect complete lattices through a pair of adjoint functions α and γ. Galois connection-based abstract interpretation suggests that one may derive an analysis systematically by composing the transition function with these adjoints: $\alpha \circ T \circ \gamma$. The function so obtained is called the *best correct aproximation* with respect to T and α. In this setting Galois connections allow us to gradually refine the collecting semantics into a computable analysis function by mere calculation. Cousot [8] has shown how to systematically construct a static analyser for a first-order imperative language using calculational abstract interpretation. An alternative "recipe" consists in rewriting the composition of the abstraction function and transition function $\alpha \circ T$ into something of the form $T^\sharp \circ \alpha$, from which the analysis function $T^\sharp$ can be read off [18]. We will use the former approach in Section 4 and the latter approach in Section 5 for deriving a CFA.

An analysis function $T^\sharp$ is said to be *complete* with respect to an abstraction α if $T^\sharp \circ \alpha = \alpha \circ T$. Intuitively, this means that the analysis is able to take full advantage of the information present in the abstract domain. The best correct approximation is not always a complete analysis function. The notion of completeness generalizes in a straightforward manner to the setting where T has different domain and codomain that are abstracted by different α's—see Giacobazzi et al. [19].

3. Language and semantics

Our source language is a simple call-by-value core language known as *administrative normal form* (ANF). The grammar of ANF terms is given in Fig. 1.

$$
\begin{array}{lll}
P \ni \mathtt{p} ::= \mathtt{s} & & \text{(programs)} \\
T \ni \mathtt{t} ::= n \mid \mathtt{x} \mid \mathtt{fn\ x\ =>\ s} & & \text{(trivial expressions)} \\
C \ni \mathtt{s} ::= & & \text{(serious expressions)} \\
\quad \mathtt{t} & & \text{(return)} \\
\quad \mid \mathtt{let\ x{=}t\ in\ s} & & \text{(let-binding)} \\
\quad \mid \mathtt{t_0\ t_1} & & \text{(tail call)} \\
\quad \mid \mathtt{let\ x{=}t_0\ t_1\ in\ s} & & \text{(non-tail call)}
\end{array}
$$

Figure 1: ANF grammar

Following Reynolds [20], the grammar distinguishes *serious* expressions, i.e., terms whose evaluation may diverge, from *trivial* expressions, i.e., terms without risk of divergence. Trivial expressions include constants, variables, and functions, and serious expressions include returns, let-bindings, tail calls, and non-tail calls. Programs are serious expressions. For more explanations about the ANF, we refer to Danvy [31] and Flanagan et al. [9].

Throughout the rest of the paper we implicitly distinguish between syntactically identical sub-terms that occur at different places in an expression. This can be achieved, *e.g.*, through a labelling of all sub-terms as is standard [21].

The analysis is calculated from a simple operational semantics in the form of an abstract machine. We use the environment-based C_aEK abstract machine of Flanagan et al. [9] given in Fig. 2. The machine represents functional values as *closures* [22], i.e., pairs of a lambda-expression and an environment. The environment-component captures the (values of the) free variables of the lambda. Machine states are triples consisting of a serious expression, an environment and a control stack. The control stack is composed of elements ("stack frames") of the form $[\mathtt{x}, \mathtt{s}, e]$ where $\mathtt{x}$ is the variable receiving the return value w of the current function call, and $\mathtt{s}$ is a serious expression whose evaluation in the environment $e[\mathtt{x} \mapsto w]$ represents the rest of the computation in that stack frame. The empty stack is represented by `stop`. The machine has a helper function μ for evaluation of trivial expressions. The machine is initialized with the input program, with an empty environment, and with an initial stack, that will bind the result of the program to a special variable $\mathtt{x_r}$ before halting. Evaluation follows by repeated application of the machine transitions.

For example, the program (which we abbreviate `p` below)

```
let f=fn x => x in
  let a1=f 1 in
    let a2=f 2 in a2
```

taken from Sabry and Felleisen [23] gives rise to the following sequence of tran-

$$\mu : T \times Env \rightharpoonup Val$$

$$Val \ni w ::= n \mid [\texttt{fn x => s},\ e] \qquad \mu(n, e) = n$$

$$Env \ni e ::= \bullet \mid e[\texttt{x} \mapsto w] \qquad \mu(\texttt{x}, e) = e(\texttt{x})$$

$$K \ni k ::= \texttt{stop} \mid [\texttt{x},\ \texttt{s},\ e] :: k \qquad \mu(\texttt{fn x => s}, e) = [\texttt{fn x => s},\ e]$$

(a) Values, environments, and stacks (b) Helper function

$$\begin{aligned}
\langle \texttt{t},\ e,\ [\texttt{x},\ \texttt{s}',\ e'] :: k' \rangle &\longrightarrow \langle \texttt{s}',\ e'[\texttt{x} \mapsto \mu(\texttt{t}, e)],\ k' \rangle \\
\langle \texttt{let x=t in s},\ e,\ k \rangle &\longrightarrow \langle \texttt{s},\ e[\texttt{x} \mapsto \mu(\texttt{t}, e)],\ k \rangle \\
\langle \texttt{t}_0\ \texttt{t}_1,\ e,\ k \rangle &\longrightarrow \langle \texttt{s}',\ e'[\texttt{x} \mapsto w],\ k \rangle \\
&\qquad \text{if } [\texttt{fn x => s}',\ e'] = \mu(\texttt{t}_0, e) \text{ and } w = \mu(\texttt{t}_1, e) \\
\langle \texttt{let x=t}_0\ \texttt{t}_1\ \texttt{in s},\ e,\ k \rangle &\longrightarrow \langle \texttt{s}',\ e'[\texttt{y} \mapsto w],\ [\texttt{x},\ \texttt{s},\ e] :: k \rangle \\
&\qquad \text{if } [\texttt{fn y => s}',\ e'] = \mu(\texttt{t}_0, e) \text{ and } w = \mu(\texttt{t}_1, e)
\end{aligned}$$

(c) Machine transitions

$$eval(\texttt{p}) = w \ \text{ iff } \ \langle \texttt{p},\ \bullet,\ [\texttt{x}_\texttt{r},\ \texttt{x}_\texttt{r},\ \bullet] :: \texttt{stop} \rangle \longrightarrow^* \langle \texttt{x}_\texttt{r},\ \bullet[\texttt{x}_\texttt{r} \mapsto w],\ \texttt{stop} \rangle$$

(d) Machine evaluation

Figure 2: The C_aEK abstract machine

sitions:

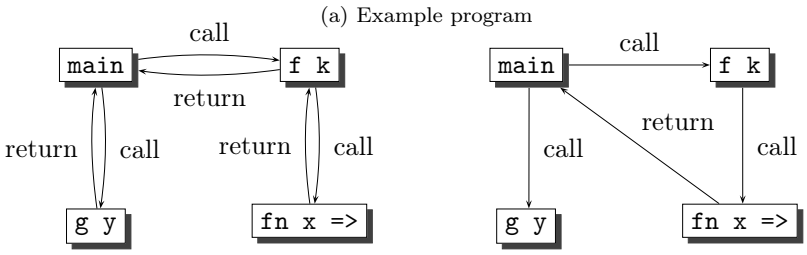
$$\begin{aligned}
&\langle \texttt{p},\ \bullet,\ \overbrace{[\texttt{x}_\texttt{r},\ \texttt{x}_\texttt{r},\ \bullet] :: \texttt{stop}}^{k_{init}} \rangle \\
\longrightarrow\ &\langle \texttt{let a}_1\texttt{=f 1 in let a}_2\texttt{=f 2 in a}_2,\ \overbrace{\bullet[\texttt{f} \mapsto [\texttt{fn x => x},\ \bullet]]}^{e_\texttt{f}},\ k_{init} \rangle && \text{(let-binding)} \\
\longrightarrow\ &\langle \texttt{x},\ \bullet[\texttt{x} \mapsto 1],\ [\texttt{a}_1,\ \texttt{let a}_2\texttt{=f 2 in a}_2,\ e_\texttt{f}] :: k_{init} \rangle && \text{(non-tail call)} \\
\longrightarrow\ &\langle \texttt{let a}_2\texttt{=f 2 in a}_2,\ e_\texttt{f}[\texttt{a}_1 \mapsto 1],\ k_{init} \rangle && \text{(return)} \\
\longrightarrow\ &\langle \texttt{x},\ \bullet[\texttt{x} \mapsto 2],\ [\texttt{a}_2,\ \texttt{a}_2,\ e_\texttt{f}[\texttt{a}_1 \mapsto 1]] :: k_{init} \rangle && \text{(non-tail call)} \\
\longrightarrow\ &\langle \texttt{a}_2,\ e_\texttt{f}[\texttt{a}_1 \mapsto 1][\texttt{a}_2 \mapsto 2],\ k_{init} \rangle && \text{(return)} \\
\longrightarrow\ &\langle \texttt{x}_\texttt{r},\ \bullet[\texttt{x}_\texttt{r} \mapsto 2],\ \texttt{stop} \rangle && \text{(return)}
\end{aligned}$$

and hence $eval(\texttt{p}) = 2$ as one would expect.

Now consider the example program in Fig. 3. The program contains three functions: two named function `g` and `f` and an anonymous function `fn x => x`. A standard direct-style CFA can determine that the applications of `k` in each branch of the conditional will call the anonymous function `fn x => x` at run time. Building a control-flow graph based on the output of a standard direct-

```
let g z = z in
 let f k = if b then k 1 else k 2 in
  let y = f (fn x => x) in
     g y
```

(a) Example program



(b) Call-return call graph (c) Optimized call graph

Figure 3: The corresponding call graphs

style CFA gives rise to Figure 3b, where we have named the main expression of the program `main`. In addition to the above resolved call, our analysis will determine that the anonymous function returns to the let-binding of `y` in `main` upon completion, rather than to its caller. The analysis hence gives rise to the more precise control-flow graph in Figure 3c.

4. Control-flow analysis

As our collecting semantics we consider the reachable states of the C_aEK machine, expressed as the least fixed point $\operatorname{lfp} F_{\mathtt{p}}$ of the following transition function.

$$F : P \to \wp(C \times Env \times K) \to \wp(C \times Env \times K)$$
$$F_{\mathtt{p}}(S) = I_{\mathtt{p}} \cup \{s \mid \exists s' \in S : s' \longrightarrow s\}$$
$$\text{where} \quad I_{\mathtt{p}} = \{\langle \mathtt{p}, \bullet, [\mathtt{x_r}, \mathtt{x_r}, \bullet] :: \mathtt{stop}\rangle\}$$

First we formulate in Fig. 4(a) an equivalent helper function μ_c extended to work on sets of environments.

Lemma 4.1. $\forall \mathtt{t}, e : \{\mu(\mathtt{t}, e)\} = \mu_c(\mathtt{t}, \{e\})$

Proof. By case analysis on $\mathtt{t}$. For constants n, we have

$$\mu_c(n, \{e\}) = \{n\} = \{\mu(n, e)\}$$

For variables $\mathtt{x}$, we have

$$\mu_c(\mathtt{x}, \{e\}) = \{w \mid \exists e' \in \{e\} : w = e'(\mathtt{x})\} = \{w \mid w = e(\mathtt{x})\} = \{\mu(\mathtt{x}, e)\}$$

For abstractions, we have

$$\mu_c(\texttt{fn x => s}, \{e\}) = \{[\texttt{fn x => s},\ e'] \mid \exists e' \in \{e\}\} = \{[\texttt{fn x => s},\ e]\} = \{\mu(\texttt{fn x => s}, e)\}$$

□

The equivalence of the two helper functions follows straightforwardly. This lemma enables us to express an equivalent collecting semantics based on μ_c, which appears in Fig. 4.

Lemma 4.2. $\forall \texttt{p}, S : F_{\texttt{p}}(S) = F^c_{\texttt{p}}(S)$

Proof. The definition of $F_{\texttt{p}}(S)$ yields

$$I_{\texttt{p}} \cup \bigcup_{s' \in S} \{s \mid s' \longrightarrow s\}$$

which can be specialized into the four set expressions defining F^c by case analysis of the transition relation $\longrightarrow$. For example, if the state is of the form $\langle \texttt{t}, e, [\texttt{x}, \texttt{s}', e']{::}k' \rangle$ then the resulting state after a $\longrightarrow$ transition will be of the form $\langle \texttt{s}',\ e'[\texttt{x} \mapsto \mu(\texttt{t}, e)],\ k' \rangle$. This state belongs to the set expression

$$\bigcup_{w \in \{\mu(\texttt{t}, e)\}} \{\langle \texttt{s}',\ e'[\texttt{x} \mapsto w],\ k' \rangle\}$$

which by Lemma 4.1 is equivalent to

$$\bigcup_{w \in \mu_c(\texttt{t}, \{e\})} \{\langle \texttt{s}',\ e'[\texttt{x} \mapsto w],\ k' \rangle\}.$$

The other cases of the proof of this lemma follow similar reasoning. □

The abstraction of the collecting semantics is staged in several steps. Figure 5 provides an overview. Intuitively, the analysis extracts three pieces of information from the set of reachable states.

1. An approximation of the set of reachable expressions.
2. A relation between expressions and control stacks that represents where the values of expressions are returned to.
3. An abstract environment mapping variables to the expressions that may be bound to that variable. This is standard in CFA and allows to determine which functions are called at a given call site.

Keeping an explicit set of reachable expressions is more precise than leaving it out, once we further approximate the expression-stack pairs. Alternatively the reachable expressions would be approximated by the expressions present in the expression-stack relation. However expressions may be in the expression-stack

$$\mu_c : T \times \wp(Env) \to \wp(Val)$$

$$\mu_c(n, E) = \{n\}$$

$$\mu_c(\mathtt{x}, E) = \{w \mid \exists e \in E : w = e(\mathtt{x})\}$$

$$\mu_c(\mathtt{fn\ x\ =>\ s}, E) = \{[\mathtt{fn\ x\ =>\ s},\ e] \mid \exists e \in E\}$$

(a) Helper function

$$F^c : P \to \wp(C \times Env \times K) \to \wp(C \times Env \times K)$$

$$\begin{aligned}
F^c_{\mathtt{p}}(S) = I_{\mathtt{p}} & \\
& \cup \bigcup_{\substack{\langle \mathtt{t},\, e,\, [\mathtt{x},\, \mathtt{s}',\, e'] :: k' \rangle \in S \\ w \in \mu_c(\mathtt{t}, \{e\})}} \{\langle \mathtt{s}',\ e'[\mathtt{x} \mapsto w],\ k' \rangle\} \\
& \cup \bigcup_{\substack{\langle \mathtt{let\ x=t\ in\ s},\, e,\, k \rangle \in S \\ w \in \mu_c(\mathtt{t}, \{e\})}} \{\langle \mathtt{s},\ e[\mathtt{x} \mapsto w],\ k \rangle\} \\
& \cup \bigcup_{\substack{\langle \mathtt{t_0\ t_1},\, e,\, k \rangle \in S \\ [\mathtt{fn\ x\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}} \{\langle \mathtt{s}',\ e'[\mathtt{x} \mapsto w],\ k \rangle\} \\
& \cup \bigcup_{\substack{\langle \mathtt{let\ x=t_0\ t_1\ in\ s},\, e,\, k \rangle \in S \\ [\mathtt{fn\ y\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}} \{\langle \mathtt{s}',\ e'[\mathtt{y} \mapsto w],\ [\mathtt{x},\ \mathtt{s},\ e] :: k \rangle\}
\end{aligned}$$

(b) Transition function

Figure 4: Collecting semantics

$$
\begin{array}{ccc}
\wp(C \times Env \times K) & \text{collecting semantics} & F^c \\
\alpha_\times \Big\downarrow \Big\uparrow \gamma_\times & & \\
\wp(C) \times \wp(C \times K) \times \wp(Env) & \text{-} & F^\times \\
\rho \Big\downarrow \Big\uparrow 1 & & \\
\rho(\wp(C) \times \wp(C \times K) \times \wp(Env)) & \text{-} & F^\rho \\
\alpha_\otimes \Big\downarrow \Big\uparrow \gamma_\otimes & & \\
\wp(C) \times (C/_\equiv \to \wp(K^\sharp)) \times Env^\sharp & \text{0-CFA} & F^\sharp
\end{array}
$$

Figure 5: Overview of abstraction

relation without ever being reached. An example hereof would be a diverging non-tail call.

To formalize this analysis, we first perform a Cartesian abstraction of the machine states, however keeping the relation between expressions and their corresponding control stacks. The next step in the approximation consists in closing the triples by a closure operator, to ensure that (a) any saved environment on the stack or nested within another environment is itself part of the environment set, and (b) that all expression-control stack pairs that appear further down in a control stack are also contained in the expression-stack relation. We explain this in more detail below (Section 4.2). Finally, we approximate stacks by their top element, we merge expressions with the same return point into equivalence classes, and we approximate closure values by their lambda expression.

In the following sections we provide a detailed explanation of each abstraction in turn. In order to illustrate the systematic calculation and still remain of a manageable size, we only provide the calculations for the return case `t`. Since we calculate with Galois connections on complete lattices, the abstraction functions are complete join morphisms (CJMs), and hence distribute over each element of a join, permitting us to do such case division. The remaining cases are proved similarly.

4.1. Projecting machine states

The mapping that extracts the three kinds of information described above is defined formally as follows.

$$
\wp(C \times Env \times K) \underset{\alpha_\times}{\overset{\gamma_\times}{\leftrightarrows}} \wp(C) \times \wp(C \times K) \times \wp(Env)
$$

$$
\alpha_\times(S) = \langle \pi_1 S, \{\langle \mathtt{s},\ e,\ k\rangle \mid \exists e : \langle \mathtt{s},\ e,\ k\rangle \in S\}, \pi_2 S\rangle
$$

$$
\gamma_\times(\langle C,\ F,\ E\rangle) = \{\langle \mathtt{s},\ e,\ k\rangle \mid \mathtt{s} \in C \ \wedge\ \langle \mathtt{s},\ k\rangle \in F \ \wedge\ e \in E\}
$$

Lemma 4.3. $\alpha_\times$, $\gamma_\times$ is a Galois connection.

Proof. The projection of a tuple space onto a sub-space of smaller dimension forms a Galois connection with its inverse. The function pair $\alpha_\times$, $\gamma_\times$ can therefore be characterized as the component-wise abstraction of three Galois connections and thus constitutes a Galois connection itself. □

We use the notation $\cup_\times$ and $\subseteq_\times$ for the componentwise join and componentwise inclusion of triples. As traditional [12, 18, 15], we will assume that the abstract product domains throughout this article have been *reduced* with respect to the empty set, i.e., all triples $\langle A, B, C\rangle$ representing the empty set $\emptyset$ ($\gamma_a(A) = \emptyset \ \vee \ \gamma_b(B) = \emptyset \ \vee \ \gamma_c(C) = \emptyset$) have been eliminated and replaced by a single bottom element $\langle \bot_a, \bot_b, \bot_c\rangle$.

We now calculate a new transfer function by composing the partly-relational abstraction with the collecting semantics. As explained above, this amounts to applying the abstraction to each of the set expressions that define the collecting semantics. For the return case $\mathtt{t}$, we obtain the following derivation:

Let $\langle C, F, E\rangle \in \wp(C) \times \wp(C \times K) \times \wp(Env)$ be given.

$$
\begin{aligned}
&\alpha_\times\Big(\bigcup_{\substack{\langle \mathtt{t}, e, [\mathtt{x}, \mathtt{s}', e']::k'\rangle \in \gamma_\times(\langle C, F, E\rangle)\\ w \in \mu_c(\mathtt{t}, \{e\})}} \{\langle \mathtt{s}', e'[\mathtt{x} \mapsto w], k'\rangle\}\Big) \\
&= \bigcup_{\substack{\langle \mathtt{t}, e, [\mathtt{x}, \mathtt{s}', e']::k'\rangle \in \gamma_\times(\langle C, F, E\rangle)\\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \ \alpha_\times(\{\langle \mathtt{s}', e'[\mathtt{x} \mapsto w], k'\rangle\}) && (\alpha_\times \text{ a CJM}) \\
&= \bigcup_{\substack{\langle \mathtt{t}, e, [\mathtt{x}, \mathtt{s}', e']::k'\rangle \in \gamma_\times(\langle C, F, E\rangle)\\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \ \langle\{\mathtt{s}'\}, \{\langle \mathtt{s}', k'\rangle\}, \{e'[\mathtt{x} \mapsto w]\}\rangle && (\text{def. } \alpha_\times) \\
&= \bigcup_{\substack{\alpha_\times(\{\langle \mathtt{t}, e, [\mathtt{x}, \mathtt{s}', e']::k'\rangle\}) \subseteq_\times \langle C, F, E\rangle\\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \ \langle\{\mathtt{s}'\}, \{\langle \mathtt{s}', k'\rangle\}, \{e'[\mathtt{x} \mapsto w]\}\rangle && (\text{Galois conn.}) \\
&= \bigcup_{\substack{\langle \{\mathtt{t}\}, \{\langle \mathtt{t}, [\mathtt{x}, \mathtt{s}', e']::k'\rangle\}, \{e\}\rangle \subseteq_\times \langle C, F, E\rangle\\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \ \langle\{\mathtt{s}'\}, \{\langle \mathtt{s}', k'\rangle\}, \{e'[\mathtt{x} \mapsto w]\}\rangle && (\text{def. } \alpha_\times)
\end{aligned}
$$

By similar calculations, we obtain the transition function given in Fig. 6. Because the transition function has been obtained by equational reasoning, it is the best correct approximation with respect to the partly-relational approximation $\alpha_\times$, as stated by the following theorem.

Theorem 4.1.

$$\forall \mathtt{p}, C, F, E : \alpha_\times(F^c_{\mathtt{p}}(\gamma_\times(\langle C, F, E\rangle))) = F^\times_{\mathtt{p}}(\langle C, F, E\rangle)$$

4.2. A closure operator on machine states

For the final analysis, we are only interested in an abstraction of the information present in an expression-stack pair. More precisely, we aim at only

$$
\begin{aligned}
F^{\times} : P &\to \wp(C) \times \wp(C \times K) \times \wp(Env) \\
&\qquad\qquad\qquad \to \wp(C) \times \wp(C \times K) \times \wp(Env) \\
F^{\times}_{\mathtt{p}}(\langle C,\, F,\, E\rangle) &= \langle \{\mathtt{p}\},\, \{\langle \mathtt{p},\, [\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop}\rangle\},\, \{\bullet\}\rangle \\
&\quad \cup_{\times} \bigcup_{\substack{\langle \{\mathtt{t}\},\, \{\langle \mathtt{t},\, [\mathtt{x},\, \mathtt{s}',\, e'] :: k'\rangle\},\, \{e\}\rangle \subseteq_{\times} \langle C,\, F,\, E\rangle \\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_{\times} \langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x} \mapsto w]\}\rangle \\
&\quad \cup_{\times} \bigcup_{\substack{\langle \{\mathtt{let\ x{=}t\ in\ s}\},\, \{\langle \mathtt{let\ x{=}t\ in\ s},\, k\rangle\},\, \{e\}\rangle \subseteq_{\times} \langle C,\, F,\, E\rangle \\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_{\times} \langle \{\mathtt{s}\},\, \{\langle \mathtt{s},\, k\rangle\},\, \{e[\mathtt{x} \mapsto w]\}\rangle \\
&\quad \cup_{\times} \bigcup_{\substack{\langle \{\mathtt{t_0\ t_1}\},\, \{\langle \mathtt{t_0\ t_1},\, k\rangle\},\, \{e\}\rangle \subseteq_{\times} \langle C,\, F,\, E\rangle \\ [\mathtt{fn\ x\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}}{}_{\times} \langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k\rangle\},\, \{e'[\mathtt{x} \mapsto w]\}\rangle \\
&\quad \cup_{\times} \bigcup_{\substack{\langle \{\mathtt{let\ x{=}t_0\ t_1\ in\ s}\},\, \{\langle \mathtt{let\ x{=}t_0\ t_1\ in\ s},\, k\rangle\},\, \{e\}\rangle \subseteq_{\times} \langle C,\, F,\, E\rangle \\ [\mathtt{fn\ y\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}}{}_{\times} \langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, [\mathtt{x},\, \mathtt{s},\, e] :: k\rangle\},\, \{e'[\mathtt{y} \mapsto w]\}\rangle
\end{aligned}
$$

Figure 6: Abstract transition function

keeping track of the link between an expression and the top stack frame in effect during its evaluation, throwing away everything below. However, we need to make this information explicit for all expressions appearing on the control stack, *i.e.*, for a pair $\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle$ we also want to retain that $\mathtt{s}'$ will eventually be evaluated with control stack k. Similarly, environments can be stored on the stack or inside other environments and will have to be extracted. We achieve this by defining a suitable *closure operator* on these nested structures. For environments, we adapt the definition of a *constituent relation* due to Milner and Tofte [24]. To deal with the control stack, we extend this to a structural order on expression-stack pairs.

Definition 4.1 (Milner and Tofte's constituent relation)**.** For each component x_i of a tuple $\langle x_0, \ldots, x_n\rangle$ we say that x_i is a *constituent* of the tuple, written $\langle x_0, \ldots, x_n\rangle \succ x_i$. For a partial function[1] $f = [x_0 \mapsto w_0, \ldots, x_n \mapsto w_n]$, we say that each w_i is a constituent of the function, written $f \succ w_i$.

For example, the empty stack `stop` is a constituent of the non-empty stack $[\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop}$ which we write as $[\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop} \succ \mathtt{stop}$. The empty environment is also a constituent thereof: $[\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop} \succ \bullet$. Similarly the closure $w = [\mathtt{fn\ i\ =>\ i},\, \bullet]$ is a constituent of the extended environment

[1] Milner and Tofte define the constituent relation for finite functions.

$\bullet[\mathtt{x} \mapsto w]$: $\bullet[\mathtt{x} \mapsto w] \succ w$ of which the empty environment is a constituent: $w \succ \bullet$. We write $\succ^*$ for the reflexive, transitive closure of the constituent relation. For example, $\bullet[\mathtt{x} \mapsto w] \succ^* \bullet$.

Definition 4.2 (Relation on expression-stack pairs)**.** Let the binary relation $\gtrdot$ on expression-stack pairs be defined by

$$\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle \gtrdot \langle \mathtt{s}',\, k\rangle.$$

Informally, expression-stack pairs are related iff

(a) the stack component of the second pair is the tail of the first pair's stack component, and

(b) the expression component of the second, resides on the top stack frame of the stack component of the first pair.

We write $\gtrdot^*$ for the expression-stack ordering induced by the reflexive, transitive closure of the expression-stack pair relation. The following lemma relates this expression-stack ordering to the constituent relation.

Lemma 4.4. $\forall \langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s},\, k\rangle \gtrdot^* \langle \mathtt{s}',\, k'\rangle \implies k \succ^* k'$

Proof. By induction. Assume that $\langle \mathtt{s},\, k\rangle = \langle \mathtt{s_0},\, k_0\rangle \gtrdot \langle \mathtt{s_1},\, k_1\rangle \gtrdot \ldots \gtrdot \langle \mathtt{s_n},\, k_n\rangle = \langle \mathtt{s}',\, k'\rangle$. If $n = 0$ then $k_0 = k_n$ and hence $k \succ^* k'$. Otherwise, we use the induction hypothesis to deduce that $k_0 \succ^* k_{n-1}$. In addition, we have that condition (a) of Definition 4.2 yields $\langle \mathtt{s_{n\text{-}1}},\, k_{n-1}\rangle \gtrdot \langle \mathtt{s_n},\, k_n\rangle \implies k_{n-1} \succ k_n$. Hence, $k_0 \succ^* k_n$ □

Next, we define an operator ρ, defined in terms of the constituent relation and the expression-stack pair ordering. This operator takes triples consisting of sets of expressions, sets of expression-stack pairs and sets of environemnts. Its purpose is

1. to "extract" all environments residing on the stacks (condition $\langle \mathtt{s},\, k\rangle \succ^* e$ in the Definition below) or nested within another environment (condition $e' \succ^* e$) and add them to the set of environments, and

2. to ensure that any expression-stack pair that appears inside a control stack is added to the expression-stack relation (condition $\langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle$).

Definition 4.3. Let ρ be the endo-function

$$\rho : \wp(C) \times \wp(C \times K) \times \wp(Env) \to \wp(C) \times \wp(C \times K) \times \wp(Env)$$

defined by

$$\begin{aligned}\rho(\langle C,\, F,\, E\rangle) = \langle C, &\{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in F : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\},\\ &\{e \mid \exists \langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \;\vee\; \exists e' \in E : e' \succ^* e\}\rangle\end{aligned}$$

Lemma 4.5. ρ is a closure operator

Proof. There are three properties to prove of ρ: monotonicity, extensiveness and idempotence.

Monotonicity. Assume $\langle C,\, F,\, E\rangle \sqsubseteq \langle C',\, F',\, E'\rangle$. Then $C \subseteq C'$, $F \subseteq F'$ and $E \subseteq E'$. Hence

$$\begin{aligned}
&\rho(\langle C,\, F,\, E\rangle)\\
&= \langle C, \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in F : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\},\\
&\qquad \{e \mid \exists \langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \ \vee\ \exists e' \in E : e' \succ^* e\}\rangle\\
&\subseteq \langle C', \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in F' : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\},\\
&\qquad \{e \mid \exists \langle \mathtt{s},\, k\rangle \in F' : \langle \mathtt{s},\, k\rangle \succ^* e \ \vee\ \exists e' \in E' : e' \succ^* e\}\rangle\\
&= \rho(\langle C',\, F',\, E'\rangle)
\end{aligned}$$

Extensiveness. This amounts to proving $\langle C,\, F,\, E\rangle \sqsubseteq \rho(\langle C,\, F,\, E\rangle)$ which is shown by proving the inclusion component-wise. ρ is the identity on the first component, so this inclusion is trivial. For the second component, it suffices to observe that

$$F \subseteq \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in F : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}$$

due to the reflexivity of $\gtrdot^*$, and for the third component, we have that

$$\begin{aligned}
E &\subseteq \{e \mid \exists e' \in E : e' \succ^* e\}\\
&\subseteq \{e \mid \exists \langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \ \vee\ \exists e' \in E : e' \succ^* e\}.
\end{aligned}$$

Idempotence. We need to show that $\rho(\rho(\langle C,\, F,\, E\rangle)) = \rho(\langle C,\, F,\, E\rangle)$. As ρ is extensive we have $\langle C,\, F,\, E\rangle \sqsubseteq \rho(\langle C,\, F,\, E\rangle)$, and, as ρ is monotone, therefore also $\rho(\langle C,\, F,\, E\rangle) \sqsubseteq \rho(\rho(\langle C,\, F,\, E\rangle))$. So, it remains to show the other inclusion $\sqsupseteq$.

Since ρ is the identity on the first component, the desired inclusion for this component is immediate. In the following, we write $(\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!2$ and $(\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3$ for the second and third component of the triple $\rho(\langle C,\, F,\, E\rangle)$. By unfolding the definition of ρ for the second component, we have

$$\begin{aligned}
&\{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!2 : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}\\
&= \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle \in \{\langle \mathtt{s}'',\, k''\rangle \in F : \langle \mathtt{s}'',\, k''\rangle \gtrdot^* \langle \mathtt{s}',\, k'\rangle\} : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}\\
&= \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}',\, k'\rangle, \langle \mathtt{s}'',\, k''\rangle : \langle \mathtt{s}'',\, k''\rangle \in F \wedge \langle \mathtt{s}'',\, k''\rangle \gtrdot^* \langle \mathtt{s}',\, k'\rangle \wedge \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}\\
&= \{\langle \mathtt{s},\, k\rangle \mid \exists \langle \mathtt{s}'',\, k''\rangle \in F : \langle \mathtt{s}'',\, k''\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}
\end{aligned}$$

which is exactly the second component of $\rho(\langle C,\, F,\, E\rangle)$.

For the third component, we need to show the inclusion

$$\begin{aligned}
&\{e \mid \exists \langle \mathtt{s},\, k\rangle \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!2 : \langle \mathtt{s},\, k\rangle \succ^* e \ \vee\ \exists e' \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3 : e' \succ^* e\}\\
&\subseteq (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3
\end{aligned}$$

where the left hand side of the inclusion is obtained by unfolding the expression $\rho(\rho(\langle C,\, F,\, E\rangle))$. Now, pick an e belonging to the left hand side. There are two

cases to consider.

$$
\begin{aligned}
&\exists\langle \mathtt{s},\, k\rangle \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!2 : \langle \mathtt{s},\, k\rangle \succ^* e \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle \in \{\langle \mathtt{s},\, k\rangle \mid \exists\langle \mathtt{s}',\, k'\rangle \in F : \langle \mathtt{s}',\, k'\rangle \succcurlyeq^* \langle \mathtt{s},\, k\rangle\} : \langle \mathtt{s},\, k\rangle \succ^* e \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s}',\, k'\rangle \in F \wedge \langle \mathtt{s}',\, k'\rangle \succcurlyeq^* \langle \mathtt{s},\, k\rangle \wedge \langle \mathtt{s},\, k\rangle \succ^* e \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s}',\, k'\rangle \in F \wedge k' \succ^* k \wedge \langle \mathtt{s},\, k\rangle \succ^* e \qquad \text{(Lemma 4.4)} \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s}',\, k'\rangle \in F \wedge \langle \mathtt{s}',\, k'\rangle \succ^* k \wedge \langle \mathtt{s},\, k\rangle \succ^* e \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s}',\, k'\rangle \in F \wedge \langle \mathtt{s}',\, k'\rangle \succ^* k \wedge k \succ^* e \\
&\Rightarrow \exists\langle \mathtt{s},\, k\rangle, \langle \mathtt{s}',\, k'\rangle : \langle \mathtt{s}',\, k'\rangle \in F \wedge \langle \mathtt{s}',\, k'\rangle \succ^* e \\
&\Rightarrow e \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3
\end{aligned}
$$

$$
\begin{aligned}
&\exists e' \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3 : e' \succ^* e \\
&\Rightarrow \exists e' \in \{e''' \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e''' \ \vee\ \exists e'' \in E : e'' \succ^* e'''\} : e' \succ^* e \\
&\Rightarrow \exists e' \in \{e''' \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e'''\} \cup \{e''' \mid \exists e'' \in E : e'' \succ^* e'''\} : e' \succ^* e \\
&\Rightarrow \exists e' : (\exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e' \wedge e' \succ^* e) \ \vee\ (\exists e'' \in E : e'' \succ^* e' \wedge e' \succ^* e) \\
&\Rightarrow (\exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e) \ \vee\ (\exists e'' \in E : e'' \succ^* e' \wedge e' \succ^* e) \\
&\Rightarrow e \in (\rho(\langle C,\, F,\, E\rangle))\!\downarrow\!3
\end{aligned}
$$

In either case, we show that e belongs to the right hand side of the inclusion. This proves the inclusion for the third component, and concludes the proof of idempotence.

□

We note without proof that ρ preserves least upper bounds, *i.e.*, it is a disjunctive closure. We can now formulate an abstraction on the triples:

$$\wp(C)\times\wp(C\times K)\times\wp(Env) \underset{\rho}{\overset{1}{\leftrightarrows}} \rho(\wp(C)\times\wp(C\times K)\times\wp(Env))$$

We use the notation $\cup_\rho$ for the join operation $\lambda X.\, \rho(\cup_\times X)$ on the closure operator-induced complete lattice. First observe that in our case:

$$\cup_\rho = \lambda X.\, \rho(\bigcup_i{}_\times\, X_i) = \lambda X.\, \bigcup_i{}_\times\, \rho(X_i) = \lambda X.\, \bigcup_i{}_\times\, X_i = \cup_\times$$

Based on the closure operator-based Galois connection, we calculate a new intermediate transfer function F^ρ. Now let $\langle C,\, F,\, E\rangle \in \rho(\wp(C)\times\wp(C\times K)\times\wp(Env))$

be given.

$$\begin{aligned}
&\rho\Big(\bigcup_{\substack{\times \\ \langle\{\mathtt{t}\},\{\langle\mathtt{t},[\mathtt{x},\mathtt{s}',e']::k'\rangle\},\{e\}\rangle\subseteq_\times\langle C,F,E\rangle \\ w\in\mu_c(\mathtt{t},\{e\})}} \langle\{\mathtt{s}'\},\ \{\langle\mathtt{s}',\ k'\rangle\},\ \{e'[\mathtt{x}\mapsto w]\}\rangle\Big) \\
&= \bigcup_{\substack{\rho \\ \langle\{\mathtt{t}\},\{\langle\mathtt{t},[\mathtt{x},\mathtt{s}',e']::k'\rangle\},\{e\}\rangle\subseteq_\times\langle C,F,E\rangle \\ w\in\mu_c(\mathtt{t},\{e\})}} \rho(\langle\{\mathtt{s}'\},\ \{\langle\mathtt{s}',\ k'\rangle\},\ \{e'[\mathtt{x}\mapsto w]\}\rangle) \qquad (\rho \text{ a CJM}) \\
&= \bigcup_{\substack{\times \\ \langle\{\mathtt{t}\},\{\langle\mathtt{t},[\mathtt{x},\mathtt{s}',e']::k'\rangle\},\{e\}\rangle\subseteq_\times\langle C,F,E\rangle \\ w\in\mu_c(\mathtt{t},\{e\})}} \rho(\langle\{\mathtt{s}'\},\ \{\langle\mathtt{s}',\ k'\rangle\},\ \{e'[\mathtt{x}\mapsto w]\}\rangle) \quad \text{(by observation)}
\end{aligned}$$

The resulting transfer function appears in Fig. 7. This transfer function differs only minimally from the one in Fig. 6, in that (a) the signature has changed, (b) the set of initial states has been "closed" and now contains the structurally smaller pair $\langle \mathtt{x_r},\ \mathtt{stop}\rangle$, and (c) the four indexed joins now each join "closed" triples in the image of the closure operator.

By construction, the new transition function satisfies the following theorem.

Theorem 4.2.

$$\forall \mathtt{p}, C, F, E : \rho \circ F^{\times}_{\mathtt{p}} \circ 1(\langle C,\ F,\ E\rangle) = F^{\rho}_{\mathtt{p}}(\langle C,\ F,\ E\rangle)$$

4.3. Abstracting the expression-stack relation

Since stacks can grow unbounded (for non-tail recursive programs), we need to approximate the stack component and hereby the expression-stack relation. The approximation that we shall be using is simply to keep only the top element of the stack and throw away the rest. We formalize this through a grammar of abstract stacks and an elementwise operator $@ : C \times K \to C \times K^\sharp$ operating on expression-stack pairs.

$$\begin{aligned}
K^\sharp \ni k^\sharp ::= \mathtt{stop} \ \mid\ [\mathtt{x},\ \mathtt{s}] \qquad & \text{(abstract stacks)} \\
@(\langle \mathtt{s},\ \mathtt{stop}\rangle) &= \langle \mathtt{s},\ \mathtt{stop}\rangle \\
@(\langle \mathtt{s},\ [\mathtt{x},\ \mathtt{s}',\ e] :: k\rangle) &= \langle \mathtt{s},\ [\mathtt{x},\ \mathtt{s}']\rangle
\end{aligned}$$

This elementwise operator constitutes an elementwise abstraction that gives rise to a Galois connection as described in Section 2.1.

$$\begin{aligned}
\alpha_@ &: \wp(C \times K) \to \wp(C \times K^\sharp) \\
\alpha_@(F) &= \{@(\langle \mathtt{s},\ k\rangle) \mid \langle \mathtt{s},\ k\rangle \in F\} \\
\gamma_@(F^\sharp) &= \{\langle \mathtt{s},\ k\rangle \mid @(\langle \mathtt{s},\ k\rangle) \in F^\sharp\}
\end{aligned}$$

$$\wp(C \times K) \underset{\alpha_@}{\overset{\gamma_@}{\leftrightarrows}} \wp(C \times K^\sharp)$$

$$
\begin{aligned}
F^\rho : P &\to \rho(\wp(C) \times \wp(C \times K) \times \wp(Env)) \\
&\to \rho(\wp(C) \times \wp(C \times K) \times \wp(Env))
\end{aligned}
$$

$$
\begin{aligned}
F^\rho_{\mathtt{p}}(\langle C,\, F,\, E\rangle) = \langle \{\mathtt{p}\},\, \{\langle \mathtt{p},\, [\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop}\rangle, \langle \mathtt{x_r},\, \mathtt{stop}\rangle\},\, \{\bullet\}\rangle \\
\cup_\times \bigcup_{\substack{\langle \{\mathtt{t}\},\, \{\langle \mathtt{t},\, [\mathtt{x},\, \mathtt{s}',\, e']::k'\rangle\},\, \{e\}\rangle \subseteq_\times \langle C,\, F,\, E\rangle \\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \rho(\langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x} \mapsto w]\}\rangle) \\
\cup_\times \bigcup_{\substack{\langle \{\mathtt{let\ x{=}t\ in\ s}\},\, \{\langle \mathtt{let\ x{=}t\ in\ s},\, k\rangle\},\, \{e\}\rangle \subseteq_\times \langle C,\, F,\, E\rangle \\ w \in \mu_c(\mathtt{t}, \{e\})}}{}_\times \rho(\langle \{\mathtt{s}\},\, \{\langle \mathtt{s},\, k\rangle\},\, \{e[\mathtt{x} \mapsto w]\}\rangle) \\
\cup_\times \bigcup_{\substack{\langle \{\mathtt{t_0\ t_1}\},\, \{\langle \mathtt{t_0\ t_1},\, k\rangle\},\, \{e\}\rangle \subseteq_\times \langle C,\, F,\, E\rangle \\ [\mathtt{fn\ x\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}}{}_\times \rho(\langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k\rangle\},\, \{e'[\mathtt{x} \mapsto w]\}\rangle) \\
\cup_\times \bigcup_{\substack{\langle \{\mathtt{let\ x{=}t_0\ t_1\ in\ s}\},\, \{\langle \mathtt{let\ x{=}t_0\ t_1\ in\ s},\, k\rangle\},\, \{e\}\rangle \subseteq_\times \langle C,\, F,\, E\rangle \\ [\mathtt{fn\ y\ =>\ s}',\, e'] \in \mu_c(\mathtt{t_0}, \{e\}) \\ w \in \mu_c(\mathtt{t_1}, \{e\})}}{}_\times \rho(\langle \{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, [\mathtt{x},\, \mathtt{s},\, e] :: k\rangle\},\, \{e'[\mathtt{y} \mapsto w]\}\rangle)
\end{aligned}
$$

Figure 7: The second abstract transition function

Some expressions share the same return point (read: same stack): the expression `let x=t in s` and the expression `s` share the same return point, and `let x=t`$_0$ `t`$_1$ `in s` and `s` share the same return point. In order to eliminate this redundancy we define an equivalence relation on serious expressions grouping together expressions sharing the same return point. We define the smallest equivalence relation $\equiv$ satisfying:

$$
\begin{aligned}
\mathtt{let\ x{=}t\ in\ s} &\equiv \mathtt{s} \\
\mathtt{let\ x{=}t_0\ t_1\ in\ s} &\equiv \mathtt{s}
\end{aligned}
$$

Based hereon we define a second elementwise operator $@' : C \times K^\sharp \to C/_\equiv \times K^\sharp$ mapping the first component of an expression-stack pair to a representative of its corresponding equivalence class:

$$
@'(\langle \mathtt{s},\, k^\sharp\rangle) = \langle [\mathtt{s}]_\equiv,\, k^\sharp\rangle
$$

We can choose the outermost expression as a representative for each equivalence class by a linear top-down traversal of the input program.
By composing the above Galois connection with a Galois connection α_ω for pointwise encoding of a relation (Section 2.1) we obtain our abstraction of the expression-stack relation:

$$
\wp(C \times K) \underset{\alpha_{st}}{\overset{\gamma_{st}}{\leftrightarrows}} C/_\equiv \to \wp(K^\sharp)
$$

where $\alpha_{st} = \alpha_\omega \circ \alpha_{@'} \circ \alpha_{@} = \lambda F. \dot{\bigcup}_{\langle \mathtt{s},\, k\rangle \in F} \alpha_\omega(\{@' \circ @(\langle \mathtt{s},\, k\rangle)\})$ and $\gamma_{st} = \gamma_{@} \circ \gamma_{@'} \circ \gamma_\omega$. We can now prove a lemma relating the concrete and abstract expression-stack relations.

Lemma 4.6. Control stack and saved environments
Let $\langle C,\, F,\, E\rangle \in \rho(\wp(C) \times \wp(C \times K) \times \wp(Env))$ be given.

$$\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle \in F \implies e \in E \;\wedge\; \{\langle \mathtt{s}',\, k\rangle\} \subseteq F \;\wedge\; \{[\mathtt{x},\, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{s}]_\equiv)$$

Proof. Assume $\{\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle\} \subseteq F$. Now $\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle \succ^* e$ and hence $e \in E$ by the assumption on E. Furthermore $\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle \gtrdot \langle \mathtt{s}',\, k\rangle$ hence $\{\langle \mathtt{s}',\, k\rangle\} \subseteq F$ by the assumption on F. For the last part we reason as follows:

$$\begin{aligned}
&\implies \alpha_{st}(\{\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle\}) \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\alpha_{st} \text{ monotone})\\
&\iff \dot{\bigcup}_{\langle \mathtt{s}'',\, k''\rangle \in \{\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle\}} \alpha_\omega(\{@' \circ @(\langle \mathtt{s}'',\, k''\rangle)\}) \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\text{def. } \alpha_{st})\\
&\iff \alpha_\omega(\{@' \circ @(\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}',\, e] :: k\rangle)\}) \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\text{def. } \dot{\cup})\\
&\iff \alpha_\omega(\{@'(\langle \mathtt{s},\, [\mathtt{x},\, \mathtt{s}']\rangle)\}) \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\text{def. } @)\\
&\iff \alpha_\omega(\{\langle [\mathtt{s}]_\equiv,\, [\mathtt{x},\, \mathtt{s}']\rangle\}) \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\text{def. } @')\\
&\iff \lambda_.\, \emptyset[[\mathtt{s}]_\equiv \mapsto \{[\mathtt{x},\, \mathtt{s}']\}] \mathrel{\dot{\subseteq}} \alpha_{st}(F) && (\text{def. } \alpha_\omega)\\
&\iff \{[\mathtt{x},\, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{s}]_\equiv) && (\text{def. } \dot{\subseteq})
\end{aligned}$$

□

4.4. Abstracting environments

We also abstract values using an elementwise abstraction. Again we formulate a grammar of abstract values and an elementwise operator $@ : Val \to Val^\sharp$ mapping concrete to abstract values.

$$\begin{gathered}
Val^\sharp \ni w^\sharp ::= n \;\mid\; [\mathtt{fn\ x\ =>\ s}]\\
@(n) = n\\
@([\mathtt{fn\ x\ =>\ s},\, e]) = [\mathtt{fn\ x\ =>\ s}]
\end{gathered}$$

The abstraction of environments, which are themselves partial functions, can be obtained by composing the two Galois connections *Pointwise Abstraction of a Set of Functions* and *Subset Abstraction* (see Section 2.1) as follows.

A standard trick is to regard partial functions $r : D \rightharpoonup C$ as total functions $r_\perp : D \to (C \cup \perp)$ where $\perp \sqsubseteq \perp \sqsubseteq c$, for all $c \in C$. Now consider environments $e \in Var \rightharpoonup Val$ to be total functions $Var \to (Val \cup \perp)$ using this idea. In this context the bottom element $\perp$ will denote variable lookup failure. Now compose a subset abstraction $\wp(Val \cup \perp) \underset{\alpha_\subset}{\overset{\gamma_\subset}{\leftrightarrows}} \wp(Val)$ with the value abstraction from the previous section, and feed the result to the pointwise abstraction above. The

result is a pointwise abstraction of a set of environments, that does not explicitly model variable lookup failure:

$$\wp(Env) \underset{\alpha_\Pi}{\overset{\gamma_\Pi}{\leftrightarrows}} Var \to \wp(Val^\sharp)$$

By considering only closed programs, we statically ensure against failure of variable-lookup, hence disregarding $\perp$ loses no information.

Given the abstraction of environments, we can calculate the corresponding abstract versions of the helper function used in the semantics, by "pushing α's" under the function definition, and reading off a resulting abstract definition. The resulting helper function reads:

$$\begin{aligned} \mu^\sharp &: T \times Env^\sharp \to \wp(Val^\sharp) \\ \mu^\sharp(n, E^\sharp) &= \{n\} \\ \mu^\sharp(\mathtt{x}, E^\sharp) &= E^\sharp(\mathtt{x}) \\ \mu^\sharp(\mathtt{fn\ x\ =>\ s}, E^\sharp) &= \{[\mathtt{fn\ x\ =>\ s}]\} \end{aligned}$$

where we write $Env^\sharp$ as shorthand for $Var \to \wp(Val^\sharp)$. The calculation can be done without introducing any additional approximations and leads to a complete abstraction of the helper function $\mu^\sharp$.

Lemma 4.7. Completeness of abstract helper function

$$\forall \mathtt{t}, E : \alpha_@(\mu_c(\mathtt{t}, E)) = \mu^\sharp(\mathtt{t}, \alpha_\Pi(E))$$

Proof. By a simple case analysis on `t` and an unfolding of the remaining definitions. □

We shall need a lemma relating the two helper function definitions on closed environments.

Lemma 4.8. Helper function on closed environments (1)
Let $\langle C,\ F,\ E\rangle \in \rho(\wp(C) \times \wp(C \times K) \times \wp(Env))$ be given.

$$\{[\mathtt{fn\ x\ =>\ s},\ e]\} \subseteq \mu_c(\mathtt{t}, E) \implies e \in E \ \wedge\ \{[\mathtt{fn\ x\ =>\ s}]\} \subseteq \mu^\sharp(\mathtt{t}, \alpha_\Pi(E))$$

Proof. By a simple case analysis on `t`. The constant and variable cases are straightforward. For the lambda case $\mathtt{t} = \mathtt{fn\ x'\ =>\ s'}$ there are two subcases to consider: $\mathtt{fn\ x'\ =>\ s'} \neq \mathtt{fn\ x\ =>\ s}$ in which case we reach a contradiction, and $\mathtt{fn\ x'\ =>\ s'} = \mathtt{fn\ x\ =>\ s}$ which follows straightforwardly. □

The above lemma is easily extended to capture nested environments in all values returned by the helper function:

Lemma 4.9. Helper function on closed environments (2)
Let $\langle C,\ F,\ E\rangle \in \rho(\wp(C) \times \wp(C \times K) \times \wp(Env))$ be given.

$$\{w\} \subseteq \mu_c(\mathtt{t}, E) \ \wedge\ w \succ^* e'' \implies e'' \in E$$

Proof. By a simple case analysis on w. In case w is a closure we apply the above lemma and the closed environment assumption. □

4.5. Putting it all together

We can now calculate the analysis as the abstraction of triplet of sets into abstract triples by a componentwise abstraction.

For the set of expressions $\wp(C)$ we use the identity abstraction consisting of two identity functions. For the expression-stack relation $\wp(C \times K)$ we use the expression-stack abstraction α_{st} developed in Section 4.3. For the set of environments $\wp(Env)$ we use the environment abstraction α_Π developed in Section 4.4. These can be combined using the Componentwise Abstraction Galois connection into the last abstraction step depicted in Figure 5.

5. Calculating the analysis

Using the alternative "recipe" we can calculate the analysis by "pushing α's" under the intermediate transition function:

$$\alpha_\otimes(F^\rho_{\mathtt{p}}(\langle C,\, F,\, E\rangle)) \subseteq_\otimes F^\sharp_{\mathtt{p}}(\langle C,\, \alpha_{st}(F),\, \alpha_\Pi(E)\rangle)$$

from which the final definition of $F^\sharp_{\mathtt{p}}$ can be read off. We recall that α_{st} was defined in Section 4.4 and α_Π in Section 4.4. For space-saving purposes the calculation is divided into a number of observations, on which the derivation relies. Let $\langle C,\, F,\, E\rangle \in \rho(\wp(C) \times \wp(C \times K) \times \wp(Env))$ be given. First observe that:

$$\begin{aligned}
&\{e \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \\
&\qquad \vee\ \exists e' \in (\bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}} \{e'[\mathtt{x} \mapsto w]\}) : e' \succ^* e\} \\
&= \{e \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \\
&\qquad \vee\ \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] \succ^* e\} && \text{(def. } \cup) \\
&= \{e \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e\} \\
&\qquad \cup \{e \mid \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] \succ^* e\} && \text{(def. } \vee) \\
&\subseteq E \cup \{e \mid \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] \succ^* e\} && \text{(assumption on } E) \\
&= E \cup \{e \mid \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] = e \\
&\qquad \vee\ e' \succ^* e\ \vee\ w \succ^* e\} && \text{(case analysis)} \\
&= E \cup \{e \mid \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] = e \\
&\qquad \vee\ e' \succ^* e\} && \text{(by Lemma 4.9)} \\
&= E \cup \{e \mid \exists e' \in E, w \in \mu_c(\mathtt{t}, E) : e'[\mathtt{x} \mapsto w] = e\} && \text{(assumption on } E) \\
&= E \cup \{e'[\mathtt{x} \mapsto w] \mid e' \in E, w \in \mu_c(\mathtt{t}, E)\} && \text{(def } =)
\end{aligned}$$

Secondly, observe that:

$$
\begin{aligned}
& \bigcup_{\substack{\{\langle \mathtt{s}',\, k'\rangle\}\subseteq F \\ \{e'\}\subseteq E \quad \{e\}\subseteq E \\ w\in\mu_c(\mathtt{t},\{e\})}}^{\times} \rho(\langle\{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x}\mapsto w]\}\rangle) \\
&= \bigcup_{\substack{\{\langle \mathtt{s}',\, k'\rangle\}\subseteq F \\ \{e'\}\subseteq E \quad w\in\mu_c(\mathtt{t},E)}}^{\times} \rho(\langle\{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x}\mapsto w]\}\rangle) && \text{(def. } \mu_c\text{)} \\
&= \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}}^{\times} \rho(\bigcup_{\{\langle \mathtt{s}',\, k'\rangle\}\subseteq F}^{\times} \langle\{\mathtt{s}'\},\, \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x}\mapsto w]\}\rangle) && (\rho \text{ a CJM}) \\
&= \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}}^{\times} \rho(\langle\{\mathtt{s}'\},\, \bigcup_{\{\langle \mathtt{s}',\, k'\rangle\}\subseteq F} \{\langle \mathtt{s}',\, k'\rangle\},\, \{e'[\mathtt{x}\mapsto w]\}\rangle) && \text{(def. } \cup_\times\text{)} \\
&\subseteq_\times \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}}^{\times} \rho(\langle\{\mathtt{s}'\},\, F,\, \{e'[\mathtt{x}\mapsto w]\}\rangle) && \text{(def. } \cup\text{)} \\
&= \rho(\bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}}^{\times} \langle\{\mathtt{s}'\},\, F,\, \{e'[\mathtt{x}\mapsto w]\}\rangle) && (\rho \text{ a CJM}) \\
&= \rho(\langle\{\mathtt{s}'\},\, F,\, \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}} \{e'[\mathtt{x}\mapsto w]\}\rangle) && \text{(def. } \cup_\times\text{)} \\
&= \begin{aligned}[l] \langle\{\mathtt{s}'\}, \{\langle \mathtt{s},\, k\rangle \mid \exists\langle \mathtt{s}',\, k'\rangle \in F : \langle \mathtt{s}',\, k'\rangle \gtrdot^* \langle \mathtt{s},\, k\rangle\}, \\ \{e \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \\ \qquad \vee\ \exists e' \in \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}} \{e'[\mathtt{x}\mapsto w]\} : e' \succ^* e\}\rangle \end{aligned} && \text{(def. } \rho\text{)} \\
&= \begin{aligned}[l] \langle\{\mathtt{s}'\}, F, \{e \mid \exists\langle \mathtt{s},\, k\rangle \in F : \langle \mathtt{s},\, k\rangle \succ^* e \\ \qquad \vee\ \exists e' \in \bigcup_{\substack{\{e'\}\subseteq E \\ w\in\mu_c(\mathtt{t},E)}} \{e'[\mathtt{x}\mapsto w]\} : e' \succ^* e\}\rangle \end{aligned} && \text{(assumption on } F\text{)} \\
&\subseteq_\times \langle\{\mathtt{s}'\},\, F,\, E\cup\{e'[\mathtt{x}\mapsto w] \mid e'\in E, w\in\mu_c(\mathtt{t},E)\}\rangle && \text{(First obs.)}
\end{aligned}
$$

Thirdly, observe that:

$$
\begin{aligned}
& \alpha_\Pi(E\cup\{e'[\mathtt{x}\mapsto w] \mid e'\in E, w\in\mu_c(\mathtt{t},E)\}) \\
&= \alpha_\Pi(E)\,\dot\cup\,\alpha_\Pi(\{e'[\mathtt{x}\mapsto w] \mid e'\in E, w\in\mu_c(\mathtt{t},E)\}) && (\alpha_\Pi \text{ a CJM}) \\
&= \alpha_\Pi(E)\,\dot\cup\,\alpha_\Pi(\{\lambda\mathtt{y}.\,\text{if } \mathtt{y}=\mathtt{x} \text{ then } w \text{ else } e'(\mathtt{y}) \mid e'\in E, \\
&\qquad\qquad w\in\mu_c(\mathtt{t},E)\}) && \text{(def. extend)} \\
&= \alpha_\Pi(E)\,\dot\cup\,\lambda\mathtt{y}.\,\text{if } \mathtt{y}=\mathtt{x} \text{ then } \alpha_@(\{w \mid w\in\mu_c(\mathtt{t},E)\}) \\
&\qquad\qquad \text{else } \alpha_@(\{e'(\mathtt{y}) \mid e'\in E\}) && \text{(def. } \alpha_\Pi\text{)} \\
&= \alpha_\Pi(E)\,\dot\cup\,\lambda\mathtt{y}.\,\text{if } \mathtt{y}=\mathtt{x} \text{ then } \alpha_@(\mu_c(\mathtt{t},E)) \text{ else } \alpha_\Pi(E)(\mathtt{y}) && \text{(def. } \alpha_\Pi\text{)}
\end{aligned}
$$

$$
\begin{aligned}
&= \alpha_\Pi(E) \,\dot\cup\, \lambda \mathtt{y}.\, \text{if } \mathtt{y} = \mathtt{x} \text{ then } \mu^\sharp(\mathtt{t}, \alpha_\Pi(E)) \text{ else } \alpha_\Pi(E)(\mathtt{y}) && \text{(by Lemma 4.7)}\\
&= \alpha_\Pi(E) \,\dot\cup\, \alpha_\Pi(E)[\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, \alpha_\Pi(E))] && \text{(def. extend)}\\
&= \alpha_\Pi(E) \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, \alpha_\Pi(E))] && \text{(def. } \dot\cup\text{)}
\end{aligned}
$$

where we have written $[\mathtt{x} \mapsto \ldots]$ as shorthand for $\lambda_.\, \emptyset[\mathtt{x} \mapsto \ldots]$. Now we can calculate the analysis:

$$
\begin{aligned}
&\alpha_\otimes\Big(\bigcup_{\substack{\times \\ \langle \{\mathtt{t}\}, \{\langle \mathtt{t}, [\mathtt{x}, \mathtt{s}', e']::k' \rangle\}, \{e\} \rangle \subseteq_\times \langle C, F, E \rangle \\ w \in \mu_c(\mathtt{t}, \{e\})}} \rho(\langle \{\mathtt{s}'\}, \{\langle \mathtt{s}', k' \rangle\}, \{e'[\mathtt{x} \mapsto w]\} \rangle) \Big) \\
&= \alpha_\otimes\Big(\bigcup_{\substack{\times \\ \{\mathtt{t}\} \subseteq C \\ \{\langle \mathtt{t}, [\mathtt{x}, \mathtt{s}', e']::k' \rangle\} \subseteq F \\ \{e\} \subseteq E \\ w \in \mu_c(\mathtt{t}, \{e\})}} \rho(\langle \{\mathtt{s}'\}, \{\langle \mathtt{s}', k' \rangle\}, \{e'[\mathtt{x} \mapsto w]\} \rangle) \Big) && \text{(def. } \subseteq_\times\text{)} \\
&\subseteq_\otimes \alpha_\otimes\Big(\bigcup_{\substack{\times \\ \{\mathtt{t}\} \subseteq C \\ \{[\mathtt{x}, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{t}]_\equiv) \\ \{\langle \mathtt{s}', k' \rangle\} \subseteq F \\ \{e'\} \subseteq E \quad \{e\} \subseteq E \\ w \in \mu_c(\mathtt{t}, \{e\})}} \rho(\langle \{\mathtt{s}'\}, \{\langle \mathtt{s}', k' \rangle\}, \{e'[\mathtt{x} \mapsto w]\} \rangle) \Big) && \text{(by Lemma 4.6)} \\
&\subseteq_\otimes \alpha_\otimes\Big(\bigcup_{\substack{\times \\ \{\mathtt{t}\} \subseteq C \\ \{[\mathtt{x}, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{t}]_\equiv)}} \langle \{\mathtt{s}'\}, F, E \cup \{e'[\mathtt{x} \mapsto w] \mid e' \in E, \\ & \qquad\qquad w \in \mu_c(\mathtt{t}, E)\} \rangle \Big) && \text{(Second obs.)} \\
&= \bigcup_{\substack{\otimes \\ \{\mathtt{t}\} \subseteq C \\ \{[\mathtt{x}, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{t}]_\equiv)}} \alpha_\otimes(\langle \{\mathtt{s}'\}, F, E \cup \{e'[\mathtt{x} \mapsto w] \mid e' \in E, \\ & \qquad\qquad w \in \mu_c(\mathtt{t}, E)\} \rangle) && \text{(}\alpha_\otimes\text{ a CJM)} \\
&= \bigcup_{\substack{\otimes \\ \{\mathtt{t}\} \subseteq C \\ \{[\mathtt{x}, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{t}]_\equiv)}} \langle \{\mathtt{s}'\}, \alpha_{st}(F), \alpha_\Pi(E \cup \{e'[\mathtt{x} \mapsto w] \mid e' \in E, \\ & \qquad\qquad w \in \mu_c(\mathtt{t}, E)\}) \rangle && \text{(def. } \alpha_\otimes\text{)} \\
&= \bigcup_{\substack{\times \\ \{\mathtt{t}\} \subseteq C \\ \{[\mathtt{x}, \mathtt{s}']\} \subseteq \alpha_{st}(F)([\mathtt{t}]_\equiv)}} \langle \{\mathtt{s}'\}, \alpha_{st}(F), \alpha_\Pi(E) \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, \alpha_\Pi(E))] \rangle && \text{(Third obs.)}
\end{aligned}
$$

The resulting analysis appears in Fig. 8. The alert reader may have noticed that this final abstraction is not *complete* in that the above derivation contains an inequality. The inequality is strict as illustrated by the following example. Consider two environments:

$$
\begin{aligned}
e_1 &= \bullet[\mathtt{x} \mapsto [\mathtt{fn\ i\ =>\ i},\ \bullet],\ \mathtt{y} \mapsto [\mathtt{fn\ j\ =>\ j},\ \bullet]] \\
e_2 &= \bullet[\mathtt{x} \mapsto [\mathtt{fn\ j\ =>\ j},\ \bullet],\ \mathtt{y} \mapsto [\mathtt{fn\ i\ =>\ i},\ \bullet]]
\end{aligned}
$$

and a triple containing the two: $\langle \{\mathtt{x\,y}\}, \{\langle \mathtt{x\,y}, \mathtt{stop} \rangle\}, \{e_1, e_2\} \rangle$. Technically this triple is not closed under the closure operator ρ as the two environments

$$
\begin{aligned}
F^\sharp : P &\to \wp(C) \times (C/_\equiv \to \wp(K^\sharp)) \times Env^\sharp \\
&\to \wp(C) \times (C/_\equiv \to \wp(K^\sharp)) \times Env^\sharp
\end{aligned}
$$

$$
\begin{aligned}
&F^\sharp_{\mathtt{p}}(\langle C,\, F^\sharp,\, E^\sharp\rangle) = \\
&\quad \langle \{\mathtt{p}\},\, [[\mathtt{p}]_\equiv \mapsto \{[\mathtt{x_r},\, \mathtt{x_r}]\},\, [\mathtt{x_r}]_\equiv \mapsto \{\mathtt{stop}\},\, \lambda_.\,\emptyset\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\{\mathtt{t}\}\subseteq C \\ \{[\mathtt{x},\mathtt{s'}]\}\subseteq F^\sharp([\mathtt{t}]_\equiv)}}^{\otimes} \langle \{\mathtt{s'}\},\, F^\sharp,\, E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)]\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\{\mathtt{let\ x{=}t\ in\ s}\}\subseteq C}}^{\otimes} \langle \{\mathtt{s}\},\, F^\sharp,\, E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)]\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\{\mathtt{t_0\ t_1}\}\subseteq C \\ \{[\mathtt{fn\ x\ {=}{>}\ s'}]\}\in\mu^\sharp(\mathtt{t_0},E^\sharp)}}^{\otimes} \langle \{\mathtt{s'}\},\, F^\sharp \,\dot\cup\, [[\mathtt{s'}]_\equiv \mapsto F^\sharp([\mathtt{t_0\ t_1}]_\equiv)],\, E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t_1}, E^\sharp)]\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\{\mathtt{let\ x{=}t_0\ t_1\ in\ s}\}\subseteq C \\ \{[\mathtt{fn\ y\ {=}{>}\ s'}]\}\in\mu^\sharp(\mathtt{t_0},E^\sharp)}}^{\otimes} \langle \{\mathtt{s'}\},\, F^\sharp \,\dot\cup\, [[\mathtt{s'}]_\equiv \mapsto \{[\mathtt{x},\, \mathtt{s}]\}],\, E^\sharp \,\dot\cup\, [\mathtt{y} \mapsto \mu^\sharp(\mathtt{t_1}, E^\sharp)]\rangle
\end{aligned}
$$

Figure 8: The resulting analysis function

contain constituent environments. Hence we include in the third component all such constituent environments: $\langle\{\mathtt{x\,y}\},\, \{\langle \mathtt{x\,y},\, \mathtt{stop}\rangle\},\, \{\bullet, e_1, e_2\}\rangle$. One can now verify that

$$
\begin{aligned}
&\alpha_\otimes(F^\rho_{\mathtt{p}}(\langle\{\mathtt{x\,y}\},\, \{\langle \mathtt{x\,y},\, \mathtt{stop}\rangle\},\, \{\bullet, e_1, e_2\}\rangle)) \\
&= \alpha_\otimes(\langle\{\mathtt{p}, \mathtt{i}, \mathtt{j}\}, \\
&\qquad \{\langle \mathtt{p},\, [\mathtt{x_r},\, \mathtt{x_r},\, \bullet] :: \mathtt{stop}\rangle, \langle \mathtt{x_r},\, \mathtt{stop}\rangle, \langle \mathtt{i},\, \mathtt{stop}\rangle, \langle \mathtt{j},\, \mathtt{stop}\rangle\}, \\
&\qquad \{\bullet,\, \bullet[\mathtt{i} \mapsto [\mathtt{fn\ j\ {=}{>}\ j},\, \bullet]],\, \bullet[\mathtt{j} \mapsto [\mathtt{fn\ i\ {=}{>}\ i},\, \bullet]]\}\rangle) \\
&= \langle\{\mathtt{p}, \mathtt{i}, \mathtt{j}\}, \\
&\quad [[\mathtt{p}]_\equiv \mapsto \{[\mathtt{x_r},\, \mathtt{x_r}]\},\, [\mathtt{x_r}]_\equiv \mapsto \{\mathtt{stop}\},\, [\mathtt{i}]_\equiv \mapsto \{\mathtt{stop}\},\, [\mathtt{j}]_\equiv \mapsto \{\mathtt{stop}\}], \\
&\quad [\mathtt{i} \mapsto \{[\mathtt{fn\ j\ {=}{>}\ j}]\},\, \mathtt{j} \mapsto \{[\mathtt{fn\ i\ {=}{>}\ i}]\}]\rangle
\end{aligned}
$$

whereas

$$
\begin{aligned}
&F^\sharp_{\mathtt{p}}(\langle\{\mathtt{x\,y}\},\, \alpha_{st}(\{\langle \mathtt{x\,y},\, \mathtt{stop}\rangle\}),\, \alpha_\Pi(\{\bullet, e_1, e_2\})\rangle) \\
&= F^\sharp_{\mathtt{p}}(\langle\{\mathtt{x\,y}\}, \\
&\qquad [[\mathtt{x\,y}]_\equiv \mapsto \mathtt{stop}], \\
&\qquad [\mathtt{x} \mapsto \{[\mathtt{fn\ i\ {=}{>}\ i}], [\mathtt{fn\ j\ {=}{>}\ j}]\},\, \mathtt{y} \mapsto \{[\mathtt{fn\ i\ {=}{>}\ i}], [\mathtt{fn\ j\ {=}{>}\ j}]\}]\rangle) \\
&= \langle\{\mathtt{p}, \mathtt{i}, \mathtt{j}\},
\end{aligned}
$$

$$[[\mathtt{p}]_\equiv \mapsto \{[\mathtt{x_r}, \mathtt{x_r}]\}, [\mathtt{x_r}]_\equiv \mapsto \{\mathtt{stop}\}, [\mathtt{i}]_\equiv \mapsto \{\mathtt{stop}\}, [\mathtt{j}]_\equiv \mapsto \{\mathtt{stop}\}],$$
$$[\mathtt{i} \mapsto \{[\mathtt{fn\ i\ =>\ i}], [\mathtt{fn\ j\ =>\ j}]\}, \mathtt{j} \mapsto \{[\mathtt{fn\ i\ =>\ i}], [\mathtt{fn\ j\ =>\ j}]\}]\rangle$$

for a program p. This example illustrates the information loss when abstracting the bindings of a set of environments as *independent attributes* to one global abstract environment: the analysis loses track of which bindings belong to the same environment. Whereas completeness is a desirable goal in an abstract interpretation it is not possible in general without refining the abstract domain [19]. As traditional [8], we instead limit upward judgements to a minimum.

As a corollary of the construction, the analysis safely approximates the reachable states of the abstract machine.

Corollary 5.1. $\forall \mathtt{p} : \alpha_\otimes \circ \rho \circ \alpha_\times(\mathrm{lfp}\, F_\mathtt{p}) \subseteq_\otimes \mathrm{lfp}\, F^\sharp_\mathtt{p}$

Proof. The only property that needs to be verified is that the resulting function is monotone (this does not follow automatically because of the upwards judgment in the derivation). The monotonicity follows from the fact that all operations involved in the definition of $F^\sharp$ are monotone. □

Table 1 in Section 6 contains an example trace of the analysis function and how it calculates an approximation of reachable states.

5.1. Characteristics of the derived analysis

First of all the analysis incorporates *reachability*: it computes an approximate set of reachable expressions and will only analyse those reachable program fragments. Reachability analyses have previously been discovered independently [25, 26, 27, 28]. In our case they arise naturally from a projecting abstraction of a reachable states collecting semantics.

Second the formulation materializes *monomorphism* into two mappings: (a) one mapping merging all bindings to the same variable, and (b) one mapping merging all calling contexts of the same function. Both characteristics are well known, but our presentation literally captures this phenomenon in two approximation functions.

Third the analysis handles returns inside-out (*"callee-restore"*), in that the called function restores control from the approximate control stack and propagates the obtained return values. This differs from the traditional direct-style presentations [29, 21] that handle returns outside-in (*"caller-restore"*) where the caller propagates the obtained return values from the body of the function to the call site (typically formulated as *conditional constraints*). Such caller-restore CFAs typically mimic the recursive nature of a corresponding interpreter, e.g., a big-step or denotational semantics. As a consequence they need not abstract the call stack. In our case the starting point was a callee-restore machine with an explicit call stack. Our systematic derivation of the "abstract interpreter" inherits this callee-restore strategy. We believe that the same strategy should be used by both a semantics and a corresponding analysis — an aspect that goes beyond analysing functional program. E.g., the Java byte code semantics of Cachera et al. [30] uses a callee-restore strategy, whereas their corresponding flow-logic

$$
\begin{array}{rll}
\mathit{CProg} \ni \mathtt{p} & ::= \mathtt{fn\ k\ =>\ e} & \text{(CPS programs)} \\
\mathit{SExp} \ni \mathtt{e} & ::= \mathtt{t}_0\,\mathtt{t}_1\,\mathtt{c} \mid \mathtt{c\ t} & \text{(serious CPS expressions)} \\
\mathit{TExp} \ni \mathtt{t} & ::= n \mid \mathtt{x} \mid \mathtt{v} \mid \mathtt{fn\ x, k\ =>\ e} & \text{(trivial CPS expressions)} \\
\mathit{CExp} \ni \mathtt{c} & ::= \mathtt{fn\ v\ =>\ e} \mid \mathtt{k} & \text{(continuation expressions)}
\end{array}
$$

Figure 9: BNF of CPS language

CFA is caller-restore. The mismatch of control transfer needlessly complicated the induction hypothesis of the machine-checked soundness proof [30]. In the words of Cachera et al. [30]: *"This is because the effect of the* `return` *is simulated by a constraint (...) attached to a different instruction"*.

In this presentation we did not include an explicit construct for recursive functions. Since our source language is untyped, it is possible to encode recursion through fixed-point operators. Explicit recursion is typically modelled by circular environments. The current formulation extends straight forwardly to handle those, because of our two-staged environment abstraction (closure operator and pointwise extended value abstraction).

6. Control-flow analysis of programs in continuation-passing style

In this section we present a CFA with reachability for a language in continuation-passing style (CPS). This analysis has been derived in the same way as the ANF CFA was derived in the previous section, using the stack-less CE-machine of Flanagan et al. [9] as operational semantics. Details of the derivation can be found in Midtgaard and Jensen [10]. We prove that the ANF analysis derived in this article achieves the same precision as obtained by first transforming a program into CPS and then using the CPS analysis. This is done by defining a relation that captures how the direct-style analysis and the CPS analysis operate in lock-step.

The grammar of CPS terms is given in Fig. 9. The grammar distinguishes variables in the original source program $\mathtt{x} \in X$, from intermediate variables $\mathtt{v} \in V$ and continuation variables $\mathtt{k} \in K$. We assume the three classes are non-overlapping. Their union constitute the domain of CPS variables $\mathit{Var} = X \cup V \cup K$. Trivial CPS expressions also include constants and functions.

6.1. CPS transformation and back again

In order to state the relation between the ANF and CPS analyses we first recall the relevant program transformations. The below presentation is based on Danvy [31], Flanagan et al. [9], and Sabry and Felleisen [23].

The CPS transformation given in Fig. 10(a) is defined by two mutually recursive functions for serious and trivial expressions, respectively. A continuation variable `k` is provided in the initial call to $\mathcal{F}$. A fresh `k` is generated in $\mathcal{V}$'s

$$\mathcal{C} : P \to \mathit{CProg}$$
$$\mathcal{C}[\mathtt{p}] = \mathtt{fn}\ \mathtt{k_p}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_p}}[\mathtt{p}]$$

$$\mathcal{F} : K \to C \to \mathit{SExp}$$
$$\mathcal{F}_{\mathtt{k}}[\mathtt{t}] = \mathtt{k}\ \mathcal{V}[\mathtt{t}]$$
$$\mathcal{F}_{\mathtt{k}}[\mathtt{let\ x{=}t\ in\ s}] = (\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k}}[\mathtt{s}])\ \mathcal{V}[\mathtt{t}]$$
$$\mathcal{F}_{\mathtt{k}}[\mathtt{t_0\ t_1}] = \mathcal{V}[\mathtt{t_0}]\ \mathcal{V}[\mathtt{t_1}]\ \mathtt{k}$$
$$\mathcal{F}_{\mathtt{k}}[\mathtt{let\ x{=}t_0\ t_1\ in\ s}] = \mathcal{V}[\mathtt{t_0}]\ \mathcal{V}[\mathtt{t_1}]\ (\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k}}[\mathtt{s}])$$

$$\mathcal{V} : T \to \mathit{TExp}$$
$$\mathcal{V}[n] = n$$
$$\mathcal{V}[\mathtt{x}] = \mathtt{x}$$
$$\mathcal{V}[\mathtt{fn\ x\ =>\ s}] = \mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]$$

(a) CPS transformation

$$\mathcal{D} : \mathit{CProg} \to P$$
$$\mathcal{D}[\mathtt{fn\ k\ =>\ e}] = \mathcal{U}[\mathtt{e}]$$

$$\mathcal{U} : \mathit{SExp} \to C$$
$$\mathcal{U}[\mathtt{k\ t}] = \mathcal{P}[\mathtt{t}]$$
$$\mathcal{U}[(\mathtt{fn\ v\ =>\ e})\ \mathtt{t}] = \mathtt{let\ v{=}}\mathcal{P}[\mathtt{t}]\ \mathtt{in}\ \mathcal{U}[\mathtt{e}]$$
$$\mathcal{U}[\mathtt{t_0\ t_1\ k}] = \mathcal{P}[\mathtt{t_0}]\ \mathcal{P}[\mathtt{t_1}]$$
$$\mathcal{U}[\mathtt{t_0\ t_1}\ (\mathtt{fn\ v\ =>\ e})] = \mathtt{let\ v{=}}\mathcal{P}[\mathtt{t_0}]\ \mathcal{P}[\mathtt{t_1}]\ \mathtt{in}\ \mathcal{U}[\mathtt{e}]$$

$$\mathcal{P} : \mathit{TExp} \to T$$
$$\mathcal{P}[n] = n$$
$$\mathcal{P}[\mathtt{x}] = \mathtt{x}$$
$$\mathcal{P}[\mathtt{v}] = \mathtt{v}$$
$$\mathcal{P}[\mathtt{fn\ x}, \mathtt{k\ =>\ e}] = \mathtt{fn\ x\ =>}\ \mathcal{U}[\mathtt{e}]$$

(b) Direct-style transformation

Figure 10: Transformations to and from CPS

lambda abstraction case. To ease the expression of the relation, we choose `k` unique to the serious expression `s` — $\mathtt{k_s}$. It follows that we only need one `k` per lambda abstraction in the original program + an additional `k` in the initial case.

It is immediate from the definition of $\mathcal{F}$ that the CPS transformation of a let-binding `let x=t in s` and the CPS transformation of its body `s` share the same continuation identifier — and similarly for non-tail calls. Hence we shall equate the two:

Definition 6.1. $\mathtt{k_s} \equiv \mathtt{k_{s'}}$ iff $\mathtt{s} \equiv \mathtt{s'}$

The direct-style transform given in Fig. 10(b) is defined by two mutually recursive functions over serious and trivial CPS expressions. We define the direct-style transformation of a program `fn k => e` as the direct-style transformation of its body $\mathcal{U}[\mathtt{e}]$. Transforming a program, a serious expression, or a trivial expression to CPS and back to direct style yields the original expression.

Lemma 6.1. $\mathcal{D}[\mathcal{C}[\mathtt{p}]] = \mathtt{p} \ \wedge \ \mathcal{U}[\mathcal{F}_{\mathtt{k}}[\mathtt{s}]] = \mathtt{s} \ \wedge \ \mathcal{P}[\mathcal{V}[\mathtt{t}]] = \mathtt{t}$

Proof. The proof follows by straightforward (mutual) structural induction on trivial and serious expressions. □

6.2. CPS analysis

Fig. 11 defines a CFA for CPS programs. It is defined as the least fixed point of a program specific transfer function $T^{\sharp}_{\mathtt{p}}$. The definition relies on two helper functions $\mu^{\sharp}_{t}$ and $\mu^{\sharp}_{c}$ for trivial and continuation expressions, respectively. The analysis computes a pair consisting of (a) a set of serious expressions (the reachable expressions) and (b) an abstract environment. Abstract environments map variables to abstract values. Abstract values can be either constants, the initial continuation `stop`, function closures [`fn x, k => e`], or continuation closures [`fn v => e`].

The definition relies on two special variables $\mathtt{k}_r$ and $\mathtt{v}_r$, the first of which names the initial continuation and the second of which names the result of the program. To ensure the most precise analysis result, variables in the source program can be renamed to be distinct as is traditional in control-flow analysis [21].

6.3. Analysis equivalence

Before formally stating the equivalence of the two analyses we will study an example run. As our example we use the ANF program:

```
let f=fn x => x in
  let a1=f 1 in
    let a2=f 2 in a2
```

taken from Sabry and Felleisen [23]. The analysis trace appears in the left column of Table 1. Similarly we study the CPS analysis of the CPS transformed

$$Env^\sharp = Var \to \wp(Val^\sharp) \qquad \text{(abstract environment)}$$

$$Val^\sharp \ni w^\sharp ::= n \;\mid\; \texttt{stop} \;\mid\; [\texttt{fn x}, \texttt{k => e}] \;\mid\; [\texttt{fn v => e}] \qquad \text{(abstract values)}$$

(a) Abstract domains

$$\mu_t^\sharp : TExp \times Env^\sharp \to \wp(Val^\sharp)$$

$$\begin{aligned}
\mu_t^\sharp(n, R^\sharp) &= \{n\} \\
\mu_t^\sharp(\texttt{x}, R^\sharp) &= R^\sharp(\texttt{x}) \\
\mu_t^\sharp(\texttt{v}, R^\sharp) &= R^\sharp(\texttt{v}) \\
\mu_t^\sharp(\texttt{fn x}, \texttt{k => e}, R^\sharp) &= \{[\texttt{fn x}, \texttt{k => e}]\}
\end{aligned}$$

$$\mu_c^\sharp : CExp \times Env^\sharp \to \wp(Val^\sharp)$$

$$\begin{aligned}
\mu_c^\sharp(\texttt{k}, R^\sharp) &= R^\sharp(\texttt{k}) \\
\mu_c^\sharp(\texttt{fn v => e}, R^\sharp) &= \{[\texttt{fn v => e}]\}
\end{aligned}$$

(b) Abstract helper functions

$$T^\sharp : CProg \to \wp(SExp) \times Env^\sharp \to \wp(SExp) \times Env^\sharp$$

$$\begin{aligned}
&T^\sharp_{\texttt{fn k => e}}(\langle Q^\sharp,\, R^\sharp\rangle) = \\
&\quad \langle \{\texttt{e}\},\, [\texttt{k}_r \mapsto \{\texttt{stop}\},\, \texttt{k} \mapsto \{[\texttt{fn v}_r \texttt{ => k}_r\, \texttt{v}_r]\}]\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\texttt{t}_0\, \texttt{t}_1\, \texttt{c} \in Q^\sharp \\ [\texttt{fn x},\texttt{k}' \texttt{ => e}'] \in \mu_t^\sharp(\texttt{t}_0, R^\sharp)}}^{\otimes} \langle \{\texttt{e}'\},\, R^\sharp \,\dot\cup\, [\texttt{x} \mapsto \mu_t^\sharp(\texttt{t}_1, R^\sharp), \texttt{k}' \mapsto \mu_c^\sharp(\texttt{c}, R^\sharp)]\rangle \\
&\quad \cup_\otimes \bigcup_{\substack{\texttt{c}\, \texttt{t} \in Q^\sharp \\ [\texttt{fn v => e}'] \in \mu_c^\sharp(\texttt{c}, R^\sharp)}}^{\otimes} \langle \{\texttt{e}'\},\, R^\sharp \,\dot\cup\, [\texttt{v} \mapsto \mu_t^\sharp(\texttt{t}, R^\sharp)]\rangle
\end{aligned}$$

(c) Abstract transition function

Figure 11: CPS analysis

program. The analysis trace appears in the right column of Table 1. Contrary to Sabry and Felleisen [23] both the ANF and the CPS analyses achieve the same precision on the example, determining that $\mathtt{a}_1$ will be bound to one of the two integer literals.

Note that integers are not approximated in either analysis. Doing so would be straightforward by utilizing the isomorphism between a mixed set of tagged elements and two separate sets for each tag [32]:

$$\wp(\mathit{Val}^\sharp) = \wp(\mathit{Const} + \mathit{Lam}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathit{Const}) \times \wp(\mathit{Lam})$$

One can now choose to approximate the set of integer constants further, e.g, by using intervals [33], or by the constant propagation lattice [34] as in the analyses of Sabry and Felleisen [23], as long as one applies the same abstraction of integers in both ANF and in CPS.

We are now in position to state our main theorem relating the ANF analysis to the CPS analysis. Intuitively the theorem relates:

- reachability in ANF to CPS reachability
- abstract stacks in ANF to CPS continuation closures
- abstract stack bottom in ANF to CPS initial continuation
- ANF closures to CPS function closures
- ANF constants to CPS constants

Theorem 6.1. Let $\mathtt{p}$ be given. Let $\langle C, F^\sharp, E^\sharp\rangle = \mathrm{lfp}\, F^\sharp_{\mathtt{p}}$ and $\langle Q^\sharp, R^\sharp\rangle = \mathrm{lfp}\, T^\sharp_{\mathcal{C}[\mathtt{p}]}$. Then

$$\begin{aligned}
\mathtt{s} \in C &\iff \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}] \in Q^\sharp \ \wedge \\
[\mathtt{x},\, \mathtt{s}'] \in F^\sharp([\mathtt{s}]_\equiv) &\iff [\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_s}) \ \wedge \\
\mathtt{stop} \in F^\sharp([\mathtt{s}]_\equiv) &\iff \mathtt{stop} \in R^\sharp(\mathtt{k_s}) \ \wedge \\
[\mathtt{fn\ x\ =>\ s}] \in E^\sharp(\mathtt{y}) &\iff [\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp(\mathtt{y}) \ \wedge \\
n \in E^\sharp(\mathtt{y}) &\iff n \in R^\sharp(\mathtt{y})
\end{aligned}$$

For the purpose of the equivalence we equate the special variables $\mathtt{x_r}$ and $\mathtt{v}_r$ both naming the result of the computations. We prove the theorem by combining an implication in each direction with the identity from Lemma 6.1. We formulate both implications as relations and prove that both relations are preserved by the transfer functions.

6.4. ANF-CPS equivalence

We formally define a relation $\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}$ that relates ANF analysis triples to CPS analysis pairs.

i	ANF trace: $\langle C_i,\ F_i^\sharp,\ E_i^\sharp\rangle$	CPS trace: $\langle Q_i^\sharp,\ R_i^\sharp\rangle$
0	$\{\texttt{let f=fn x => x in let a}_1\texttt{=f 1 in let a}_2\texttt{=f 2 in a}_2\}$ $\left[\begin{array}{l}[\texttt{x}_\texttt{r}]_\equiv \mapsto \{\texttt{stop}\},\\ [\texttt{let f=fn x => x in let a}_1\texttt{=f 1 in let a}_2\texttt{=f 2 in a}_2]_\equiv \mapsto \{[\texttt{x}_\texttt{r},\ \texttt{x}_\texttt{r}]\}\end{array}\right]$ $\lambda_.\,\emptyset$	$\{(\texttt{fn f => f 1 (fn a}_1\texttt{ => f 2 (fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2\texttt{)))}\ (\texttt{fn x, k}_\texttt{x}\texttt{ => k}_\texttt{x}\ \texttt{x})\}$ $\left[\begin{array}{l}\texttt{k}_r \mapsto \{\texttt{stop}\},\\ \texttt{k}_\texttt{p} \mapsto \{[\texttt{fn v}_r\texttt{ => k}_r\ \texttt{v}_r]\}\end{array}\right]$
1	$C_0 \cup \{\texttt{let a}_1\texttt{=f 1 in let a}_2\texttt{=f 2 in a}_2\}$ $F_0^\sharp$ $E_0^\sharp \ \dot\cup\ \left[\texttt{f} \mapsto \{[\texttt{fn x => x}]\}\right]$	$Q_0^\sharp \cup \{\texttt{f 1 (fn a}_1\texttt{ => f 2 (fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2\texttt{))}\}$ $R_0^\sharp \ \dot\cup\ \left[\texttt{f} \mapsto \{[\texttt{fn x, k}_\texttt{x}\texttt{ => k}_\texttt{x}\ \texttt{x}]\}\right]$
2	$C_1 \cup \{\texttt{x}\}$ $F_1^\sharp \ \dot\cup\ \left[[\texttt{x}]_\equiv \mapsto \{[\texttt{a}_1,\ \texttt{let a}_2\texttt{=f 2 in a}_2]\}\right]$ $E_1^\sharp \ \dot\cup\ \left[\texttt{x} \mapsto \{1\}\right]$	$Q_1^\sharp \cup \{\texttt{k}_\texttt{x}\ \texttt{x}\}$ $R_1^\sharp \ \dot\cup\ \left[\begin{array}{l}\texttt{k}_\texttt{x} \mapsto \{[\texttt{fn a}_1\texttt{ => f 2 (fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2\texttt{)}]\}\\ \texttt{x} \mapsto \{1\}\end{array}\right]$
3	$C_2 \cup \{\texttt{let a}_2\texttt{=f 2 in a}_2\}$ $F_2^\sharp$ $E_2^\sharp \ \dot\cup\ \left[\texttt{a}_1 \mapsto \{1\}\right]$	$Q_2^\sharp \cup \{\texttt{f 2 (fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2\texttt{)}\}$ $R_2^\sharp \ \dot\cup\ \left[\texttt{a}_1 \mapsto \{1\}\right]$
4	C_3 $F_3^\sharp \ \dot\cup\ \left[[\texttt{x}]_\equiv \mapsto \{[\texttt{a}_1,\ \texttt{let a}_2\texttt{=f 2 in a}_2], [\texttt{a}_2,\ \texttt{a}_2]\}\right]$ $E_3^\sharp \ \dot\cup\ \left[\texttt{x} \mapsto \{1, 2\}\right]$	$Q_3^\sharp$ $R_3^\sharp \ \dot\cup\ \left[\begin{array}{l}\texttt{k}_\texttt{x} \mapsto \{[\texttt{fn a}_1\texttt{ => f 2 (fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2\texttt{)}], [\texttt{fn a}_2\texttt{ => k}_\texttt{p}\ \texttt{a}_2]\}\\ \texttt{x} \mapsto \{1, 2\}\end{array}\right]$
5	$C_4 \cup \{\texttt{a}_2\}$ $F_4^\sharp$ $E_4^\sharp \ \dot\cup\ \left[\begin{array}{l}\texttt{a}_1 \mapsto \{1, 2\}\\ \texttt{a}_2 \mapsto \{1, 2\}\end{array}\right]$	$Q_4^\sharp \cup \{\texttt{k}_\texttt{p}\ \texttt{a}_2\}$ $R_4^\sharp \ \dot\cup\ \left[\begin{array}{l}\texttt{a}_1 \mapsto \{1, 2\}\\ \texttt{a}_2 \mapsto \{1, 2\}\end{array}\right]$
6	$C_5 \cup \{\texttt{x}_\texttt{r}\}$ $F_5^\sharp$ $E_5^\sharp \ \dot\cup\ \left[\texttt{x}_\texttt{r} \mapsto \{1, 2\}\right]$	$Q_5^\sharp \cup \{\texttt{k}_r\ \texttt{v}_r\}$ $R_5^\sharp \ \dot\cup\ \left[\texttt{v}_r \mapsto \{1, 2\}\right]$
7	$C_6 \quad F_6^\sharp \quad E_6^\sharp$	$Q_6^\sharp \quad R_6^\sharp$

Table 1: Analysis traces of `let f=fn x => x in let a₁=f 1 in let a₂=f 2 in a₂` and its CPS transformed counterpart

Definition 6.2. $\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle$ iff

$$\begin{aligned}
\mathtt{s} \in C &\implies \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}] \in Q^\sharp \;\wedge \\
[\mathtt{x},\, \mathtt{s}'] \in F^\sharp([\mathtt{s}]_\equiv) &\implies [\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_s}) \;\wedge \\
\mathtt{stop} \in F^\sharp([\mathtt{s}]_\equiv) &\implies \mathtt{stop} \in R^\sharp(\mathtt{k_s}) \;\wedge \\
[\mathtt{fn\ x\ =>\ s}] \in E^\sharp(\mathtt{y}) &\implies [\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp(\mathtt{y}) \;\wedge \\
n \in E^\sharp(\mathtt{y}) &\implies n \in R^\sharp(\mathtt{y})
\end{aligned}$$

First we need a small lemma relating the ANF helper function to one of the CPS helper functions.

Lemma 6.2.

$$\begin{aligned}
[\mathtt{fn\ x\ =>\ s}] \in \mu^\sharp(\mathtt{t}, E^\sharp) \;\wedge\; \langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle \\
\implies [\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in \mu^\sharp_t(\mathcal{V}[\mathtt{t}], R^\sharp)
\end{aligned}$$

and

$$n \in \mu^\sharp(\mathtt{t}, E^\sharp) \;\wedge\; \langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle \implies n \in \mu^\sharp_t(\mathcal{V}[\mathtt{t}], R^\sharp)$$

Proof. The proof for each part follows by a simple case analysis on $\mathtt{t}$. □

The relation is preserved by the transfer functions.

Theorem 6.2.

$$\begin{aligned}
&\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle \\
&\qquad\implies F^\sharp_{\mathtt{p}}(\langle C,\, F^\sharp,\, E^\sharp\rangle) \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; T^\sharp_{\mathcal{C}[\mathtt{p}]}(\langle Q^\sharp,\, R^\sharp\rangle)
\end{aligned}$$

Proof. First we name the individual triples of the union in the function body of $F^\sharp$. We name the first triple of results as initial:

$$\langle C_I,\, F^\sharp_I,\, E^\sharp_I\rangle = \langle\{\mathtt{p}\},\, [[\mathtt{p}]_\equiv \mapsto \{[\mathtt{x_r},\, \mathtt{x_r}]\},\, [\mathtt{x_r}]_\equiv \mapsto \{\mathtt{stop}\}],\, \lambda_.\,\emptyset\rangle$$

The results of the second, third, fourth, and fifth joined triples corresponding to return, binding, tail call, and non-tail call are named $\langle C_{ret},\, F^\sharp_{ret},\, E^\sharp_{ret}\rangle$, $\langle C_{bind},\, F^\sharp_{bind},\, E^\sharp_{bind}\rangle$, $\langle C_{tc},\, F^\sharp_{tc},\, E^\sharp_{tc}\rangle$ and $\langle C_{ntc},\, F^\sharp_{ntc},\, E^\sharp_{ntc}\rangle$, respectively. Similarly we name the first result pair in the function body of the CPS analysis as initial: $\langle Q^\sharp_I,\, R^\sharp_I\rangle = \langle\{\mathtt{e}\},\, [\mathtt{k}_r \mapsto \{\mathtt{stop}\},\, \mathtt{k} \mapsto \{[\mathtt{fn\ v}_r\ \mathtt{=>}\ \mathtt{k}_r\ \mathtt{v}_r]\}]\rangle$. The results of the second and third joined pair corresponding to call and return are named $\langle Q^\sharp_{call},\, R^\sharp_{call}\rangle$ and $\langle Q^\sharp_{ret},\, R^\sharp_{ret}\rangle$, respectively.

The proof proceeds by verifying five relations:

$$\langle C_I,\, F^\sharp_I,\, E^\sharp_I\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_I,\, R^\sharp_I\rangle \tag{1}$$

$$\langle C_{ret},\, F^\sharp_{ret},\, E^\sharp_{ret}\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_{ret},\, R^\sharp_{ret}\rangle \tag{2}$$

$$\langle C_{bind},\, F^\sharp_{bind},\, E^\sharp_{bind}\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_{ret},\, R^\sharp_{ret}\rangle \tag{3}$$

$$\langle C_{tc},\, F^\sharp_{tc},\, E^\sharp_{tc}\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_{call},\, R^\sharp_{call}\rangle \tag{4}$$

$$\langle C_{ntc},\, F^\sharp_{ntc},\, E^\sharp_{ntc}\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_{call},\, R^\sharp_{call}\rangle \tag{5}$$

We now prove the return case relation (2): $\langle C_{ret},\, F^\sharp_{ret},\, E^\sharp_{ret}\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp_{ret},\, R^\sharp_{ret}\rangle$. The remaining cases follow by similar reasoning.

(2a) Assume $\mathtt{s} \in C_{ret}$. Hence there exists $\mathtt{x}, \mathtt{s}', \mathtt{t}$ such that $\mathtt{s} = \mathtt{s}'$, $\{\mathtt{t}\} \subseteq C$, and $\{[\mathtt{x},\, \mathtt{s}']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$.

From the $\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle$ assumption we have $\mathcal{F}_{\mathtt{k_t}}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_t})$.

Hence $\mathtt{k_t}\ \mathcal{V}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in \mu^\sharp_c(\mathtt{k_t}, R^\sharp)$. As a consequence $\mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}'] \in Q^\sharp_{ret}$.

(2b) Assume $[\mathtt{x},\, \mathtt{s}'] \in F^\sharp_{ret}([\mathtt{s}]_\equiv)$. Hence there exists $\mathtt{x}'', \mathtt{s}'', \mathtt{t}$ such that $\{\mathtt{t}\} \subseteq C$, $\{[\mathtt{x}'',\, \mathtt{s}'']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$, and $[\mathtt{x},\, \mathtt{s}'] \in F^\sharp_{ret}([\mathtt{s}]_\equiv) = F^\sharp([\mathtt{s}]_\equiv)$.

From the $\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle$ assumption we have $\mathcal{F}_{\mathtt{k_t}}[\mathtt{t}] \in Q^\sharp$, $[\mathtt{fn\ x''\ =>}\ \mathcal{F}_{\mathtt{k_{s''}}}[\mathtt{s}'']] \in R^\sharp(\mathtt{k_t})$, and $[\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_s})$.

Hence $\mathtt{k_t}\ \mathcal{V}[\mathtt{t}] \in Q^\sharp$, $[\mathtt{fn\ x''\ =>}\ \mathcal{F}_{\mathtt{k_{s''}}}[\mathtt{s}'']] \in \mu^\sharp_c(\mathtt{k_t}, R^\sharp)$, and $[\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_s})$. Since $R^\sharp \dot{\subseteq} R^\sharp_{ret}$ we have $[\mathtt{fn\ x\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp_{ret}(\mathtt{k_s})$.

(2c) Assume $\mathtt{stop} \in F^\sharp_{ret}([\mathtt{s}]_\equiv)$. Hence there exists $\mathtt{x}'', \mathtt{s}'', \mathtt{t}$ such that $\{\mathtt{t}\} \subseteq C$, $\{[\mathtt{x}'',\, \mathtt{s}'']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$, and $\mathtt{stop} \in F^\sharp_{ret}([\mathtt{s}]_\equiv) = F^\sharp([\mathtt{s}]_\equiv)$.

From the $\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle$ assumption we have $\mathcal{F}_{\mathtt{k_t}}[\mathtt{t}] \in Q^\sharp$, $[\mathtt{fn\ x''\ =>}\ \mathcal{F}_{\mathtt{k_{s''}}}[\mathtt{s}'']] \in R^\sharp(\mathtt{k_t})$, and $\mathtt{stop} \in R^\sharp(\mathtt{k_s})$.

Hence $\mathtt{k_t}\ \mathcal{V}[\mathtt{t}] \in Q^\sharp$, $[\mathtt{fn\ x''\ =>}\ \mathcal{F}_{\mathtt{k_{s''}}}[\mathtt{s}'']] \in \mu^\sharp_c(\mathtt{k_t}, R^\sharp)$, and $\mathtt{stop} \in R^\sharp(\mathtt{k_s})$. Since $R^\sharp \dot{\subseteq} R^\sharp_{ret}$ we have $\mathtt{stop} \in R^\sharp_{ret}(\mathtt{k_s})$.

(2d) Assume $[\mathtt{fn\ x\ =>\ s}] \in E^\sharp_{ret}(\mathtt{y})$. Hence there exists $\mathtt{x}', \mathtt{s}', \mathtt{t}$ such that $\{\mathtt{t}\} \subseteq C$, $\{[\mathtt{x}',\, \mathtt{s}']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$, and $[\mathtt{fn\ x\ =>\ s}] \in (E^\sharp \dot{\cup} [\mathtt{x}' \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)])(\mathtt{y})$.

From the $\langle C,\, F^\sharp,\, E^\sharp\rangle \;\mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\; \langle Q^\sharp,\, R^\sharp\rangle$ assumption we have $\mathcal{F}_{\mathtt{k_t}}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x'\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_t})$.

Hence $\mathtt{k_t}\ \mathcal{V}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x'\ =>}\ \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in \mu^\sharp_c(\mathtt{k_t}, R^\sharp)$.

There are now two subcases:

1. $[\mathtt{fn\ x\ =>\ s}] \in E^\sharp(\mathtt{y})$. Hence $[\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp(\mathtt{y})$. Since $R^\sharp \dot{\subseteq} R^\sharp_{ret}$ we have $[\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp_{ret}(\mathtt{y})$.
2. $[\mathtt{fn\ x\ =>\ s}] \in [\mathtt{x}' \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)](\mathtt{y})$. If $\mathtt{y} \neq \mathtt{x}'$ our assumption reads $[\mathtt{fn\ x\ =>\ s}] \in \emptyset$. Hence $[\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp_{ret}(\mathtt{y})$ is trivially true.

   If $\mathtt{y} = \mathtt{x}'$ our assumption reads $[\mathtt{fn\ x\ =>\ s}] \in \mu^\sharp(\mathtt{t}, E^\sharp)$. By Lemma 6.2 it now follows that $[\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in \mu^\sharp_t(\mathcal{V}[\mathtt{t}], R^\sharp)$. As a consequence $[\mathtt{fn\ x}, \mathtt{k_s}\ \mathtt{=>}\ \mathcal{F}_{\mathtt{k_s}}[\mathtt{s}]] \in R^\sharp_{ret}(\mathtt{y})$.

(2e) Assume $n \in E^\sharp_{ret}(\mathtt{y})$. Hence there exists $\mathtt{x}', \mathtt{s}', \mathtt{t}$ such that $\{\mathtt{t}\} \subseteq C$, $\{[\mathtt{x}', \mathtt{s}']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$, and $n \in (E^\sharp \dot\cup [\mathtt{x}' \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)])(\mathtt{y})$.

From the $\langle C,\, F^\sharp,\, E^\sharp\rangle \ \mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\ \langle Q^\sharp,\, R^\sharp\rangle$ assumption we again have $\mathcal{F}_{\mathtt{k_t}}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x'} \Rightarrow \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in R^\sharp(\mathtt{k_t})$.

Hence $\mathtt{k_t}\ \mathcal{V}[\mathtt{t}] \in Q^\sharp$ and $[\mathtt{fn\ x'} \Rightarrow \mathcal{F}_{\mathtt{k_{s'}}}[\mathtt{s}']] \in \mu^\sharp_c(\mathtt{k_t}, R^\sharp)$.

There are now two subcases:

1. $n \in E^\sharp(\mathtt{y})$. Hence $n \in R^\sharp(\mathtt{y})$. Since $R^\sharp \dot\subseteq R^\sharp_{ret}$ we have $n \in R^\sharp_{ret}(\mathtt{y})$.
2. $n \in [\mathtt{x}' \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)](\mathtt{y})$. If $\mathtt{y} \neq \mathtt{x}'$ our assumption reads $n \in \emptyset$. Hence $n \in R^\sharp_{ret}(\mathtt{y})$ is trivially true.

   If $\mathtt{y} = \mathtt{x}'$ our assumption reads $n \in \mu^\sharp(\mathtt{t}, E^\sharp)$. By Lemma 6.2 it now follows that $n \in \mu^\sharp_t(\mathcal{V}[\mathtt{t}], R^\sharp)$. As a consequence $n \in R^\sharp_{ret}(\mathtt{y})$.

Realizing that the union of related triples and pairs are related we obtain the desired result. □

After realizing that the bottom elements are related by the above relation, it follows by fixed point induction that their least fixed points (and hence the analyses) are related.

Corollary 6.1. $\mathrm{lfp}\, F^\sharp_{\mathtt{p}}\ \mathrm{R}^{\mathrm{ANF}}_{\mathrm{CPS}}\ \mathrm{lfp}\, T^\sharp_{\mathcal{C}[\mathtt{p}]}$

6.5. CPS-ANF equivalence

Again we formally define a relation now relating CPS analysis pairs to ANF analysis triples.

Definition 6.3. $\langle Q^\sharp,\, R^\sharp\rangle\ \mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\ \langle C,\, F^\sharp,\, E^\sharp\rangle$ iff

$$\begin{aligned}
\mathtt{e} \in Q^\sharp &\implies \mathcal{U}[\mathtt{e}] \in C \ \wedge \\
[\mathtt{fn\ x} \Rightarrow \mathtt{e}] \in R^\sharp(\mathtt{k_s}) &\implies [\mathtt{x}, \mathcal{U}[\mathtt{e}]] \in F^\sharp([\mathtt{s}]_\equiv) \ \wedge \\
\mathtt{stop} \in R^\sharp(\mathtt{k_s}) &\implies \mathtt{stop} \in F^\sharp([\mathtt{s}]_\equiv) \ \wedge \\
[\mathtt{fn\ x}, \mathtt{k_s} \Rightarrow \mathtt{e}] \in R^\sharp(\mathtt{y}) &\implies [\mathtt{fn\ x} \Rightarrow \mathcal{U}[\mathtt{e}]] \in E^\sharp(\mathtt{y}) \ \wedge \\
n \in R^\sharp(\mathtt{y}) &\implies n \in E^\sharp(\mathtt{y})
\end{aligned}$$

We again need a helper lemma relating the helper functions.

Lemma 6.3.

$$\begin{aligned}
[\mathtt{fn\ x}, \mathtt{k_s} \Rightarrow \mathtt{e}] \in \mu^\sharp_t(\mathtt{t}, R^\sharp) \ \wedge\ \langle Q^\sharp,\, R^\sharp\rangle\ \mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\ \langle C,\, F^\sharp,\, E^\sharp\rangle \\
\implies [\mathtt{fn\ x} \Rightarrow \mathcal{U}[\mathtt{e}]] \in \mu^\sharp(\mathcal{P}[\mathtt{t}], E^\sharp)
\end{aligned}$$

and

$$n \in \mu^\sharp_t(\mathtt{t}, R^\sharp) \ \wedge\ \langle Q^\sharp,\, R^\sharp\rangle\ \mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\ \langle C,\, F^\sharp,\, E^\sharp\rangle \implies n \in \mu^\sharp(\mathcal{P}[\mathtt{t}], E^\sharp)$$

Proof. The proof for each part follows by a simple case analysis on $\mathtt{t}$. □

This relation is also preserved by the transfer functions.

Theorem 6.3.

$$\langle Q^\sharp,\, R^\sharp\rangle \;\mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\; \langle C,\, F^\sharp,\, E^\sharp\rangle \\ \Longrightarrow\; T^\sharp_{\mathcal{C}[\mathtt{p}]}(\langle Q^\sharp,\, R^\sharp\rangle)\; \mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\; F^\sharp_{\mathtt{p}}(\langle C,\, F^\sharp,\, E^\sharp\rangle)$$

Proof. The proof follows a similar structure to the above proof. □

The bottom elements are related by the relation and it again follows by fixed point induction that their least fixed points (and hence the analyses) are related.

Corollary 6.2. $\mathrm{lfp}\; T^\sharp_{\mathcal{C}[\mathtt{p}]}\; \mathrm{R}^{\mathrm{CPS}}_{\mathrm{ANF}}\; \mathrm{lfp}\; F^\sharp_{\mathtt{p}}$

7. Extracting a constraint-based CFA

The resulting analysis may appear complex at first glance. However, we can express the analysis in the popular constraint formulation, extracted from the obtained definition. The formulation shown below is in terms of program-specific conditional constraints.

Constraints have a (possibly empty) list of preconditions and a conclusion [26, 28]:

$$\{u_1\} \subseteq rhs_1 \;\wedge\; \ldots \;\wedge\; \{u_n\} \subseteq rhs_n \Rightarrow lhs \subseteq rhs$$

The constraints operate on the same three domains as the above analysis. Left-hand sides *lhs* can be of the form $\{u\}$, $F^\sharp([\mathtt{s}]_\equiv)$, or $E^\sharp(\mathtt{x})$, right-hand sides *rhs* can be of the form C, $F^\sharp([\mathtt{s}]_\equiv)$, or $E^\sharp(\mathtt{x})$, and singleton elements u can be of the form $\mathtt{s}$, n, $[\mathtt{fn\ x\ =>\ s}]$, or $[\mathtt{x},\, \mathtt{s}]$. From Fig. 8 we can directly extract the following constraints that must be satisfied by a valid result of the CFA. More precisely, each set expression in the set union defining the control flow analysis equation gives rise to a set of constraints. For example, one of the expressions in the equation reads:

$$\bigcup_{\substack{\{\mathtt{t}\}\subseteq C \\ \{[\mathtt{x},\,\mathtt{s}']\}\subseteq F^\sharp([\mathtt{t}]_\equiv)}}^{\otimes} \langle\{\mathtt{s}'\},\, F^\sharp,\, E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)]\rangle$$

Correspondingly, for each return expression $\mathtt{t}$ and non-tail call $\mathtt{let\ x{=}t_0\ t_1\ in\ s'}$ in $\mathtt{p}$, we generate the constraints:

$$\{\mathtt{t}\} \subseteq C \;\wedge\; \{[\mathtt{x},\, \mathtt{s}']\} \subseteq F^\sharp([\mathtt{t}]_\equiv) \Rightarrow \begin{cases} \{\mathtt{s}'\} \subseteq C \;\wedge \\ \mu_{sym}(\mathtt{t}, E^\sharp) \subseteq E^\sharp(\mathtt{x}) \end{cases}$$

The other parts of the definition of $F^\sharp$ similarly induce a constraint generation scheme, as follows:

- For the program $\mathtt{p}$:

$$\{\mathtt{p}\} \subseteq C \qquad \{[\mathtt{x_r},\, \mathtt{x_r}]\} \subseteq F^\sharp([\mathtt{p}]_\equiv) \qquad \{\mathtt{stop}\} \subseteq F^\sharp([\mathtt{x_r}]_\equiv)$$

- For each let-binding $\mathtt{let\ x{=}t\ in\ s}$ in $\mathtt{p}$:

$$\{\mathtt{let\ x{=}t\ in\ s}\} \subseteq C \Rightarrow \begin{cases} \{\mathtt{s}\} \subseteq C \ \wedge \\ \mu_{sym}(\mathtt{t}, E^\sharp) \subseteq E^\sharp(\mathtt{x}) \end{cases}$$

- For each tail call $\mathtt{t_0\, t_1}$ and function $\mathtt{fn\ x\ {=}{>}\ s'}$ in $\mathtt{p}$:

$$\begin{array}{c} \{\mathtt{t_0\, t_1}\} \subseteq C \ \wedge \\ \{[\mathtt{fn\ x\ {=}{>}\ s'}]\} \subseteq \mu_{sym}(\mathtt{t_0}, E^\sharp) \end{array} \Rightarrow \begin{cases} \{\mathtt{s'}\} \subseteq C \ \wedge \\ F^\sharp([\mathtt{t_0\, t_1}]_\equiv) \subseteq F^\sharp([\mathtt{s'}]_\equiv) \ \wedge \\ \mu_{sym}(\mathtt{t_1}, E^\sharp) \subseteq E^\sharp(\mathtt{x}) \end{cases}$$

- For each non-tail call $\mathtt{let\ x{=}t_0\, t_1\ in\ s}$ and function $\mathtt{fn\ y\ {=}{>}\ s'}$ in $\mathtt{p}$:

$$\begin{array}{l} \{\mathtt{let\ x{=}t_0\, t_1\ in\ s}\} \subseteq C \ \wedge \\ \{[\mathtt{fn\ y\ {=}{>}\ s'}]\} \subseteq \mu_{sym}(\mathtt{t_0}, E^\sharp) \end{array} \Rightarrow \begin{cases} \{\mathtt{s'}\} \subseteq C \ \wedge \\ \{[\mathtt{x},\, \mathtt{s}]\} \subseteq F^\sharp([\mathtt{s'}]_\equiv) \ \wedge \\ \mu_{sym}(\mathtt{t_1}, E^\sharp) \subseteq E^\sharp(\mathtt{y}) \end{cases}$$

where we partially evaluate the helper function μ_{sym}, i.e., interpret the helper function symbolically at constraint-generation time, to generate a lookup for variables, and a singleton for constants and lambda expressions. The definition of the symbolic helper function otherwise coincides with the abstract helper function $\mu^\sharp$:

$$\begin{aligned} \mu_{sym}(n, E^\sharp) &= \{n\} \\ \mu_{sym}(\mathtt{x}, E^\sharp) &= E^\sharp(\mathtt{x}) \\ \mu_{sym}(\mathtt{fn\ x\ {=}{>}\ s}, E^\sharp) &= \{[\mathtt{fn\ x\ {=}{>}\ s}]\} \end{aligned}$$

We may generate constraints $\{[\mathtt{fn\ x\ {=}{>}\ s}]\} \subseteq \{[\mathtt{fn\ y\ {=}{>}\ s'}]\}$ of a form not covered by the above grammar. We therefore first pre-process the constraints in linear time,

- removing vacuously true inclusions $\{[\mathtt{fn\ x\ {=}{>}\ s}]\} \subseteq \{[\mathtt{fn\ x\ {=}{>}\ s}]\}$ from each constraint, and

- removing constraints with vacuously false preconditions $\{[\mathtt{fn\ x\ {=}{>}\ s}]\} \subseteq \{w^\sharp\}$, where $[\mathtt{fn\ x\ {=}{>}\ s}] \neq w^\sharp$.

The resulting constraint system is formally equivalent to the control flow analysis in the sense that all solutions yield correct control flow information and that the best (smallest) solution of the constraints is as precise as the information computed by the analysis. More formally:

Theorem 7.1. A solution to the CFA constraints of program $\mathtt{p}$ is a safe approximation of the least fixpoint of the analysis function $F^\sharp$ induced by $\mathtt{p}$. Furthermore, the least solution to the CFA constraints is equal to the least fixpoint of $F^\sharp$.

Proof. The first part of the theorem is proved by showing that a solution to the CFA constraints $\langle C,\, F,\, E\rangle$ is a post-fixpoint of $F^\sharp$, *i.e.*, that it satisfies $F^\sharp(\langle C,\, F,\, E\rangle) \subseteq_\otimes \langle C,\, F,\, E\rangle$ and then appeal to the Knaster-Tarski fixpoint theorem that the least fixpoint of a monotone operator $F^\sharp$ is the greatest lower bound of the set of post-fixpoints of $F^\sharp$. This reduces to showing that for each of the expressions defining $F^\sharp$ in Fig. 8 we have that its value is already included in the solution $\langle C,\, F,\, E\rangle$. For example, for the expression

$$\bigcup_{\substack{\{\mathtt{t}\}\subseteq C \\ \{[\mathtt{x},\,\mathtt{s}']\}\subseteq F^\sharp([\mathtt{t}]_\equiv)}}^{\otimes} \langle\{\mathtt{s}'\},\, F^\sharp,\, E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)]\rangle$$

we must have, for all $\mathtt{t}$ satisfying $\{\mathtt{t}\} \subseteq C$ and $\mathtt{s}'$ satisfying $\{[\mathtt{x},\, \mathtt{s}']\} \subseteq F^\sharp([\mathtt{t}]_\equiv)$, that

$$\{\mathtt{s}'\} \subseteq C \quad \text{and} \quad E^\sharp \,\dot\cup\, [\mathtt{x} \mapsto \mu^\sharp(\mathtt{t}, E^\sharp)] \,\dot\subseteq\, E^\sharp.$$

The latter inequality reduces to $\mu_{sym}(\mathtt{t}, E^\sharp) \subseteq E^\sharp(\mathtt{x})$. and we obtain exactly the constraints for return expressions. The other cases follow by similar reasoning.

For the equality of the least solution and the least fixpoint, it then suffices to prove that the fixpoint is a solution to the CFA constraints. The argumentation is again based on unfolding the definition of $F^\sharp$ and using reasoning similar to above. □

Implemented naively, a single constraint may take $O(n)$ space alone. However by using a pointer or the implicit label of each sub-expression instead of the sub-expression itself, a single constraint takes only constant space. By linearly determining a representative for each sub-expression, by generating $O(n^2)$ constraints, linear post-processing, and iteratively solving them using a well-known algorithm [26, 28, 21], we can compute the analysis in worst-case $O(n^3)$ time.

The extracted constraints bear similarities to existing constraint-based analyses in the literature. Consider, e.g., calls $\mathtt{t_0}\,\mathtt{t_1}$, which usually gives rise to two conditional constraints [29, 21]: (1) $\{[\mathtt{fn\ x\ =>\ s'}]\} \subseteq \widehat{C}(\mathtt{t_0}) \Rightarrow \widehat{C}(\mathtt{t_1}) \subseteq \widehat{E}(\mathtt{x})$ and (2) $\{[\mathtt{fn\ x\ =>\ s'}]\} \subseteq \widehat{C}(\mathtt{t_0}) \Rightarrow \widehat{C}(\mathtt{s'}) \subseteq \widehat{C}(\mathtt{t_0}\,\mathtt{t_1})$. The first constraint resembles our third constraint for tail calls. The second "return constraint" differs in that it has a inside-out (or caller-restore) nature, i.e., propagation of return-flow from the function body is handled at the call-site. The extracted reachability constraints are similar to Gasser et al. [28] (modulo an isomorphic encoding $\wp(C) \simeq C \to \wp(\{\mathbf{on}\})$ of powersets).

8. Related work

We separate the discussion of related analyses in two: direct-style analyses and analyses based on CPS.

Direct-style CFA has a long research history. Jones [2] initially developed methods for approximating the control flow of lambda terms. Since then Sestoft [35] conceived the related *closure analysis*. Palsberg [29] simplified the analysis and formulated an equivalent constraint-based analysis. At the same time Heintze [36] developed a related set-based analysis formulated in terms of set constraints. For a detailed account of related work, we refer to a recent survey of the area [7]. It is worth emphasizing that all of the above analyses focus on calls, in that they approximate the source lambdas being called at each call-site. As such they do not directly determine return flow for programs in direct style. Continuation-passing style CFA was pioneered by Shivers [4] who formulated control-flow analysis for Scheme. Since then a number of analyses have been formulated for CPS [3, 37, 38]. In CPS all calls are tail calls, and even returns are encoded as calls to the current continuation. By determining "call flow" and hence the receiver functions of such continuation calls, a CPS-based CFA thereby determines return flow without additional effort.

A long-standing question in flow analysis is to characterize the impact of CPS transformation on the precision of program analysis [23, 39, 40]. The study of this question originated in binding-time analysis, for which the transformation is known to have a positive effect [41, 40]. The following example is due to Damian and Danvy [40]. Consider a let binding `let x=s in s'` in which `s` is *dynamic* (unknown) and `s'` is *static* (known at compile time). Since evaluating `s` may have an effect, e.g, non-termination, the result of the entire expression has to be qualified as dynamic. Now consider its CPS counterpart: `fn k => e (fn x => e' k)` in which `e` and `e'` represent the CPS transformation of `s` and `s'`, respectively. In CPS the result of evaluating `e'` (which is sent to `k`) may now be qualified as static independent of `e`.

As to the impact of CPS transformation on CFA we separate the previous work on the subject in two:

1. results relating an analysis *specialized* to the source language to an analysis *specialized* to the target language (CPS), and

2. results relating the analysis of a program to the *same analysis* of the CPS transformed program.

Sabry and Felleisen [23] designed and compared specialized analyses and hence falls into the first category as does the present paper. Damian and Danvy [40] related the analysis of a program and its CPS counterpart for a standard flow-logic CFA (as well as for two binding-time analyses), and Palsberg and Wand [39] related the analysis of a program and its CPS counterpart for a standard conditional constraint CFA. Hence the latter two fall into the second category.

We paraphrase the relevant theorems of Sabry and Felleisen [23], of Damian and Danvy [40], of Palsberg and Wand [39], and of the present paper in order to underline the difference between the contributions (C refers to non-trivial, 0-CFA-like analyses defined in the cited papers, p ranges over direct-style programs, *cps* denotes CPS transformation, and $\sim$ denotes analysis equivalence).

Our formulations should not be read as a formal system, but only as a means for elucidating the difference between the contributions.

Sabry and Felleisen [23]:

$$\text{exists analyses } C_1,\, C_2 : \text{ exists } p,\, C_1(p) \nsim C_2(cps(p))$$

Damian and Danvy [40], Palsberg and Wand [39]:

$$\text{exists analysis } C : \text{ for all } p,\, C(p) \sim C(cps(p))$$

Present paper, Theorem 6.1:

$$\text{exists analyses } C_1,\, C_2 : \text{ for all } p,\, C_1(p) \sim C_2(cps(p))$$

Our work relates to all of the above contributions. The disciplined derivation of specialized CPS and direct-style analyses results in comparable analyses, contrary to Sabry and Felleisen [23]. Furthermore our equivalence proof extends the results of Damian and Danvy [40] and Palsberg and Wand [39] in that we relate both call flow, *return flow*, and *reachability*, contrary to their relating only the call flow of standard CFAs. In addition, the systematic abstract interpretation-based approach suggests a strategy for obtaining similar equivalence results for other CFAs derived in this fashion.

Formulating CFA in the traditional abstract interpretation framework was stated as an open problem by [6]. It has been a recurring theme in the work of the present authors. In an earlier paper Spoto and Jensen [42] investigated class analysis of object-oriented programs as a Galois connection-based abstraction of a trace semantics. In a recent article [10], the authors systematically derived a CPS-based CFA from the collecting semantics of a stack-less machine. While investigating how to derive a corresponding direct-style analysis we discovered the mismatch between the computed return information.

As tail calls are identified syntactically, the additional information could also have been obtained by a subsequent analysis after a traditional direct-style CFA. However we view the need for such a subsequent analysis as a strong indication of a mismatch between the two analysis formulations. Debray and Proebsting [43] have investigated such a *"return analysis"* for a first-order language with tail-call optimization. The present paper builds a semantics-based CFA that determines such information, and for a higher-order language.

The systematic design of constraint-based analyses is a goal shared with the *flow logic* framework of Nielson and Nielson [44]. In flow logic an analysis specification can be systematically transformed into a constraint-based analysis. The present paper instead extracts a constraint-based analysis from an analysis developed in the original abstract interpretation framework.

9. Conclusions

We have presented a control-flow analysis determining interprocedural control-flow of both calls and returns for a direct-style language. Existing CFAs have

focused on analysing which functions are called at a given call site. In contrast, the systematic derivation of our CFA has lead to an analysis that provides extra information about where a function returns to at no additional cost. In the presence of tail-call optimization, such information enables the creation of more precise call graphs.

The analysis was developed systematically using Galois connection-based abstract interpretation of a standard operational semantics for that language: the C_aEK abstract machine of Flanagan et al. In addition to being more principled, such a formulation of the analysis is pedagogically pleasing since monomorphism of the analysis is made explicit through two Galois connections: one literally merges all bindings to the same variable and one merges all calling contexts of the same function.

The analysis has been shown to provide a result equivalent to what can be obtained by first CPS transforming the program and then running a control-flow analysis derived from a CPS-based operational semantics. This extends previous results obtained by Damian and Danvy, and Palsberg and Wand. The close correspondence between the way that the analyses operate (as illustrated by the analysis trace in Table 1) leads us to conjecture that such equivalence results can be obtained for other CFAs derived using abstract interpretation.

The functional, derived by abstract interpretation, that defines the analysis may appear complex at first glance. As a final result, we have shown how to extract from the analysis an equivalent constraint-based formulation expressed in terms of the more familiar conditional constraints. Nevertheless, we stress that the derived functional can be used directly to implement the analysis. We have developed a prototype implementation of the resulting analysis in OCaml.[2]

The analysis has been developed for a minimalistic functional language in order to be able to focus on the abstraction of the control structure induced by function calls and returns. An obvious extension is to enrich the language with numerical operators and study how our Galois connections interact with abstractions such as the interval or polyhedral abstraction of numerical entities.

The calculations involved in the derivation of a CFA are lengthy and would benefit from some form of machine support. *Certified abstract interpretation* [45, 30] has so far focused on proving the soundness of the analysis inside a proof assistant by using the concretization (γ) component of the Galois connection to prove the correctness of an already defined analysis. Further work should investigate whether proof assistants such as Coq are suitable for conducting the kind of reasoning developed in this paper in a machine-checkable way.

Acknowledgement:. The authors thank Matthew Fluet, Amr Sabry, Matthias Felleisen, Mitchell Wand, Daniel Damian, Olivier Danvy, Casey Klein, and Robby Findler for comments on earlier versions of this paper. Part of this work was done with the support of the Carlsberg Foundation and an INRIA post-doc grant.

[2]available at `http://www.cs.au.dk/~jmi/ANF-CFA/`

References

[1] N. D. Jones, Flow Analysis of Lambda Expressions, Technical report PB-128, Aarhus University, Aarhus, Denmark, 1981.

[2] N. D. Jones, Flow analysis of lambda expressions (preliminary version), in: S. Even, O. Kariv (Eds.), Automata, Languages and Programming, 8th Colloquium, Acre (Akko), volume 115 of *Lecture Notes in Computer Science*, Springer-Verlag, Israel, 1981, pp. 114–128.

[3] A. E. Ayers, Abstract Analysis and Optimization of Scheme, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1993.

[4] O. Shivers, Control-flow analysis in Scheme, in: M. D. Schwartz (Ed.), Proceedings of the ACM SIGPLAN 1988 Conference on Programming Languages Design and Implementation, Atlanta, Georgia, pp. 164–174.

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

[6] F. Nielson, H. R. Nielson, Infinitary control flow analysis: a collecting semantics for closure analysis, in: [46], pp. 332–345.

[7] J. Midtgaard, Control-flow analysis of functional programs, ACM Computing Surveys (2012). In press.

[8] P. Cousot, The calculational design of a generic abstract interpreter, in: M. Broy, R. Steinbrüggen (Eds.), Calculational System Design, NATO ASI Series F. IOS Press, Amsterdam, 1999.

[9] C. Flanagan, A. Sabry, B. F. Duba, M. Felleisen, The essence of compiling with continuations, in: D. W. Wall (Ed.), Proceedings of the ACM SIGPLAN 1993 Conference on Programming Languages Design and Implementation, Albuquerque, New Mexico, pp. 237–247.

[10] J. Midtgaard, T. Jensen, A calculational approach to control-flow analysis by abstract interpretation, in: M. Alpuente, G. Vidal (Eds.), Static Analysis, 15th International Symposium, SAS 2008, volume 5079 of *Lecture Notes in Computer Science*, Springer-Verlag, Valencia, Spain, 2008, pp. 347–362.

[11] J. Midtgaard, T. P. Jensen, Control-flow analysis of function calls and returns by abstract interpretation, in: G. Hutton, A. P. Tolmach (Eds.), Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming (ICFP'09), Edinburgh, Scotland, pp. 287–298.

[12] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: B. K. Rosen (Ed.), Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, pp. 269–282.

[13] P. Cousot, Semantic foundations of program analysis, in: S. S. Muchnick, N. D. Jones (Eds.), Program Flow Analysis: Theory and Applications, Prentice-Hall, 1981, pp. 303–342.

[14] P. Cousot, R. Cousot, Abstract interpretation and application to logic programs, Journal of Logic Programming 13 (1992) 103–179.

[15] P. Cousot, 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 (Ed.), Proceedings of the Fifth IEEE International Conference on Computer Languages, Toulouse, France, pp. 95–112.

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

[17] P. Cousot, R. Cousot, Abstract interpretation of algebraic polynomial systems, in: M. Johnson (Ed.), Proceedings of the Sixth International Conference on Algebraic Methodology and Software Technology, AMAST '97, volume 1349 of *Lecture Notes in Computer Science*, Springer-Verlag, Sydney, Australia, 1997, pp. 138–154.

[18] P. Cousot, R. Cousot, Abstract interpretation frameworks, Journal of Logic and Computation 2 (1992) 511–547.

[19] R. Giacobazzi, F. Ranzato, F. Scozzari, Making abstract interpretations complete, Journal of the ACM 47 (2000) 361–416.

[20] J. C. Reynolds, Definitional interpreters for higher-order programming languages, Higher-Order and Symbolic Computation 11 (1998) 363–397. Reprinted from the proceedings of the 25th ACM National Conference (1972).

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

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

[23] A. Sabry, M. Felleisen, Is continuation-passing useful for data flow analysis?, in: V. Sarkar (Ed.), Proceedings of the ACM SIGPLAN 1994 Conference on Programming Languages Design and Implementation, Orlando, Florida, pp. 1–12.

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

[25] A. E. Ayers, Efficient closure analysis with reachability, in: M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, A. Rauzy (Eds.), Actes WSA'92 Workshop on Static Analysis, Bigre, Atelier Irisa, IRISA, Campus de Beaulieu, Bordeaux, France, 1992, pp. 126–134.

[26] J. Palsberg, M. I. Schwartzbach, Safety analysis versus type inference, Information and Computation 118 (1995) 128–141.

[27] S. K. Biswas, A demand-driven set-based analysis, in: [46], pp. 372–385.

[28] K. L. S. Gasser, F. Nielson, H. R. Nielson, Systematic realisation of control flow analyses for CML, in: M. Tofte (Ed.), Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming, Amsterdam, The Netherlands, pp. 38–51.

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

[30] D. Cachera, T. Jensen, D. Pichardie, V. Rusu, Extracting a data flow analyser in constructive logic, Theoretical Computer Science 342 (2005) 56–78.

[31] O. Danvy, Three Steps for the CPS Transformation, Technical Report CIS-92-2, Kansas State University, Manhattan, Kansas, 1991.

[32] A. Deutsch, On determining lifetime and aliasing of dynamically allocated data in higher-order functional specifications, in: P. Hudak (Ed.), Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, pp. 157–168.

[33] P. Cousot, R. Cousot, Static determination of dynamic properties of programs, in: Proceedings of the Second International Symposium on Programming, Dunod, Paris, France, 1976, pp. 106–130.

[34] G. Kildall, A unified approach to global program optimization, in: J. D. Ullman (Ed.), Proceedings of the First Annual ACM Symposium on Principles of Programming Languages, Boston, Massachusetts, pp. 194–206.

[35] P. Sestoft, Replacing function parameters by global variables, in: J. E. Stoy (Ed.), Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture, London, England, pp. 39–53.

[36] N. Heintze, Set-based program analysis of ML programs, in: C. L. Talcott (Ed.), Proceedings of the 1994 ACM Conference on Lisp and Functional Programming, LISP Pointers, Vol. VII, No. 3, Orlando, Florida, pp. 306–317.

[37] J. M. Ashley, R. K. Dybvig, A practical and flexible flow analysis for higher-order languages, ACM Transactions on Programming Languages and Systems 20 (1998) 845–868.

[38] M. Might, O. Shivers, Environmental analysis via ΔCFA, in: S. Peyton Jones (Ed.), Proceedings of the 33rd Annual ACM Symposium on Principles of Programming Languages, Charleston, South Carolina, pp. 127–140.

[39] J. Palsberg, M. Wand, CPS transformation of flow information, Journal of Functional Programming 13 (2003) 905–923.

[40] D. Damian, O. Danvy, Syntactic accidents in program analysis: On the impact of the CPS transformation, Journal of Functional Programming 13 (2003) 867–904. A preliminary version was presented at the 2000 ACM SIGPLAN International Conference on Functional Programming.

[41] C. Consel, O. Danvy, For a better support of static data flow, in: J. Hughes (Ed.), Proceedings of the Fifth ACM Conference on Functional Programming Languages and Computer Architecture, volume 523 of *Lecture Notes in Computer Science*, Springer-Verlag, Cambridge, Massachusetts, 1991, pp. 496–519.

[42] F. Spoto, T. P. Jensen, Class analyses as abstract interpretations of trace semantics, ACM Transactions on Programming Languages and Systems 25 (2003) 578–630.

[43] S. K. Debray, T. A. Proebsting, Interprocedural control flow analysis of first-order programs with tail-call optimization, ACM Transactions on Programming Languages and Systems 19 (1997) 568–585.

[44] H. R. Nielson, F. Nielson, Flow logic: a multi-paradigmatic approach to static analysis, in: T. Æ. Mogensen, D. A. Schmidt, I. H. Sudborough (Eds.), The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones, volume 2566 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002, pp. 223–244.

[45] D. Pichardie, Interprétation abstraite en logique intuitioniste: extraction d'analyseurs Java certifiés, Ph.D. thesis, Université de Rennes 1, 2005.

[46] N. D. Jones (Ed.), Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages, Paris, France, 1997.