CPS Transformation of Flow Information

Jens Palsberg — Purdue University*
Mitchell Wand — Northeastern University†

December 18, 2001

Abstract

We consider the question of how a continuation-passing-style (CPS) transformation changes the flow analysis of a program. We present an algorithm that takes the least solution to the flow constraints of a program and constructs in linear time the least solution to the flow constraints for the CPS-transformed program.

Previous studies of this question used CPS transformations that had the effect of duplicating code, or of introducing flow sensitivity into the analysis. Our algorithm has the property that for a program point in the original program and the corresponding program point in the CPS-transformed program, the flow information is the same. By carefully avoiding both duplicated code and flow-sensitive analysis, we find that the most accurate analysis of the CPS-transformed program is neither better nor worse than the most accurate analysis of the original. Thus a compiler that needed flow information after CPS transformation could use the flow information from the original program to annotate some program points, and it could use our algorithm to find the rest of the flow information quickly, rather than having to analyze the CPS-transformed program.

1 Introduction

For simply-typed λ-calculus, typability is preserved across CPS transformation. For a CPS transformation $[\![\cdot]\!]$ of call-by-value terms, Meyer and Wand [8] have shown that if $A \vdash e : t$, then $A^* \vdash [\![e]\!] : (t^* \to o) \to o$, where o is a type of answers, where t^* is defined inductively:

$$\begin{aligned} \alpha^* &= \alpha \\ (s \to t)^* &= s^* \to (t^* \to o) \to o, \end{aligned}$$

*Purdue University, Dept of Computer Science, W Lafayette, IN 47907, USA, palsberg@cs.purdue.edu.

†College of Computer Science, Northeastern University, 360 Huntington Avenue, 161CN, Boston, MA 02115, USA, wand@ccs.neu.edu.

and where $A^*(x) = t^*$ if $A(x) = t$. The function $(\cdot)^*$ defines CPS transformation of type information. In fact, for a related system, [19] showed that the converse implication holds as well.

A series of papers [13, 6, 14, 12] have suggested that flow analyses are analogous to type systems. It is therefore natural to ask the question:

Is flow information preserved by a CPS transformation?

We show that for an untyped λ-calculus with constants and conditionals, a standard notion of 0-CFA flow analysis, and a carefully-formulated definition of the CPS translation, flow information is preserved and reflected across CPS transformation. More precisely, we show that if φ is the least (most accurate) flow analysis of E, then φ^* is the least flow analysis of $\mathsf{cps}(E)$, where $(\cdot)^*$ is a linear-time computable transformation of flow information.

Our algorithm has the property that for a program point in the original program and the corresponding program point in the CPS-transformed program, the flow information is the same. Thus, the algorithm does not change the flow information; it merely extends it to cover the new program points.

Aside from its role in answering a theoretical question, our algorithm might be useful in a compiler that needed flow information after CPS transformation. Rather than analyzing the CPS-transformed program, it can instead CPS-transform flow information for the source program. Depending on the amount of flow information actually generated, doing this transformation (in time linear in the size of the annotation) could be faster than reanalyzing the CPS-transformed program (in time possibly cubic in the size of the transformed program). On the other hand, for certain typed programs, Heintze and McAllester [7] have shown that 0-CFA flow information can be computed in $O(n^2)$ time. For such programs, a reanalysis may be more attractive. Experiments are needed to get a firmer grip on these issues.

In previous work, Palsberg [11] has shown that 0-CFA flow information is still valid after beta-reduction. Note, though, that leastness of flow information need not be preserved by beta-reduction. This parallels that principality of types need not be preserved by beta-reduction.

There is a range of previous work on the relationship between CPS transformation and flow analysis. In some previous work, where the studied frameworks are somewhat different, flow analysis is *not* preserved across CPS transformation; see section 5 for more detail. It is necessary to be careful in the formulation of both the analysis and the CPS transformation to avoid the effects of code duplication or flow-sensitivity. In other work, independent of ours, again where the studied framework is somewhat different, Damian and Danvy [2] has shown a result that is similar to our result in Section 3. They consider a different CPS transformation and they study terms in which all intermediate results are given a name with a let-expression.

In the following section we give an informal account of our approach to CPS transformation of flow information, and in section 3 we present our algorithm and main result. In section 4 we show a preservation theorem for flow types, and finally in section 5 we discuss related work.

2 Summary of Our Approach

The principal difficulty in defining a CPS transformation of flow information is that the CPS-transformed program contains program points that have no counterparts in the source program. Our main observation is that this task becomes manageable when we use a variant of Plotkin's call-by-value CPS transformation that was introduced by Danvy and Filinski [4]. Here is part of the definition of their CPS transformation:

$$
\begin{array}{rcl}
[\![x^l]\!] & = & \lambda^{P_l}k.k^{K_l} @^{A_l} x^l \\
[\![\lambda^l x.e]\!] & = & \lambda^{P_l}k.k^{K_l} @^{A_l} (\lambda^l x.\lambda^{Q_l}m.[\![e]\!] @^{B_{L(e)}} (\lambda^{R_{L(e)}}v.m^{M_l} @^{C_l} v^{V_{L(e)}})) \\
[\![e_1 @^l e_2]\!] & = & \lambda^{P_l}k.[\![e_1]\!] @^{B_{L(e_1)}} (\lambda^{R_{L(e_1)}}v_1.[\![e_2]\!] @^{B_{L(e_2)}} (\lambda^{R_{L(e_2)}}v_2. \\
& & (v_1^{V_{L(e_1)}} @^{G_l} v_2^{V_{L(e_2)}}) @^{D_l} (\lambda^{S_l}v.k^{K_l} @^{A_l} v^l)))
\end{array}
$$

Our scheme for labeling expressions is different from the one used by Danvy and Filinski. We label all occurrences of expressions in the source and target programs; the function L maps a λ-term to its topmost label. The labels in the CPS-transformed program are chosen such that the label of a λ-abstraction in the source program is also the label of the corresponding λ-abstraction in the CPS-transformed program. Many of the labels in the target program are obtained by applying a label transformer to a label in the source program. For example, A_l denotes a label computed from the label l. It is essential for our approach that we can compute A_l from l, and that we can compute l from A_l. We use the notation $L(e)$ to denote the topmost label of e.

We will now give an informal explanation of how to compute flow information for some of the new program points. Consider first:

$$[\![\lambda^l x.e]\!] \quad = \quad \ldots (\lambda^l x.\lambda^{Q_l}m.\ldots (m^{M_l} @^{C_l} v^{V_{L(e)}}))\ldots$$

Question: Where can $(\lambda^l x.\lambda^{Q_l}m.\ldots (m^{M_l} @^{C_l} v^{V_{L(e)}}))$ be applied?

To answer that, suppose we have a set σ of labels of application points that tells us where $(\lambda^l x.e)$ in the source program can be applied. Suppose $a \in \sigma$, and consider the CPS transformation of the application point labeled a:

$$[\![e_1 @^a e_2]\!] \quad = \quad \ldots (v_1^{V_{L(e_1)}} @^{G_a} v_2^{V_{L(e_2)}}) @^{D_a} (\lambda^{S_a}v.\ldots)\ldots$$

We have that v_1 will hold the value of evaluating $[\![e_1]\!]$, so $(\lambda^l x.\lambda^{Q_l}m.\ldots)$ can be applied at the application point labeled G_a. So, in general, $(\lambda^l x.\lambda^{Q_l}m.\ldots)$ can be applied at application points labeled with labels from the set $G_\sigma = \{\ G_l \ \mid\ l \in \sigma\ \}$. Below, we will use notation like G_σ for other label transformers than G.

> **Question:** Where can $(\lambda^{Q_l} m.\ldots(m^{M_l} @^{C_l} v^{V_{L(e)}}))$ be applied?

From above we have that $(\lambda^{Q_l} m.\ldots)$ can be the result of evaluating the expression $(v_1^{V_{L(e_1)}} @^{G_a} v_2^{V_{L(e_2)}})$, so $(\lambda^{Q_l} m.\ldots)$ can be applied at application points labeled with labels from the set D_σ.

Consider again $[\![\lambda^l x.e]\!]$.

> **Question:** What are the labels of the λ-abstractions that can be applied at $(m^{M_l} @^{C_l} v^{V_{L(e)}})$?

Consider the program point D_a in $[\![e_1 @^a e_2]\!]$. The argument there is the λ-abstraction $(\lambda^{S_a} v.\ldots)$, so the set of labels of the λ-abstractions that can be applied at $(m^{M_l} @^{C_l} v^{V_{L(e)}})$ is S_σ.

Consider next

$$[\![e_1 @^a e_2]\!] \quad = \quad \ldots(v_1^{V_{L(e_1)}} @^{G_a} v_2^{V_{L(e_2)}}) @^{D_a} (\lambda^{S_a} v.\ldots)\ldots$$

Suppose we have a set π of labels of λ-abstractions that can be applied at $e_1 @^a e_2$ in the source program. Due to our approach to labeling, the set of labels of λ-abstractions that can be applied at $(v_1^{V_{L(e_1)}} @^{G_a} v_2^{V_{L(e_2)}})$ is also π.

> **Question:** What is the set of functions that can be applied at the application labeled D_a?

From

$$[\![\lambda^l x.e]\!] \quad = \quad \ldots\lambda^l x.\lambda^{Q_l} m.\ldots(m^{M_l} @^{C_l} v^{V_{L(e)}})\ldots$$

we have that the set is Q_π. Consider again $[\![e_1 @^a e_2]\!]$.

> **Question:** Where can $(\lambda^{S_a} v.\ldots)$ be applied?

As analyzed above, it is applied at points $(m^{M_l} @^{C_l} v^{V_{L(e)}})$ and we can express the set of labels as C_π.

Our algorithm for CPS transformation of flow information expands the above observations to give a flow analysis for a whole CPS-transformed program.

3 Main Result

We will work with an initial Σ-algebra *Lab* of labels where Σ is:

$$\begin{array}{l} \ldots, l, a, \ldots \ : \ Lab \quad \text{(an infinite collection of constant label symbols)} \\ A, B, C, D, F, G, H^1, H^2, J, K, M, \\ \ \ N^1, N^2, P, Q, R, S, T, U, V, W, X, Y, Z \ : \ Lab \to Lab. \end{array}$$

We apologize that there is no mnemonic significance to the names of the unary operations. We will write the application of, say, A to l as A_l. Notice that since *Lab* is an initial Σ-algebra, the unary operations have ranges that are mutually disjoint and disjoint from the set of constant label symbols. Furthermore, initiality also implies that the operations are injective, so that we can, for example, recover l from A_l.

Our example language is defined by the grammar:

$$e \ ::= \ x^l \ \mid \ q^l \ \mid \ \lambda^l x.e \ \mid \ e_1 \,@^l\, e_2 \ \mid \ e_0 \to^l e_1, e_2.$$

We use q to range over a set of first-order constants that includes true, false. The construct $e_0 \to^l e_1, e_2$ is a conditional expression that branches depending on whether e_0 evaluates to true or false. A *program* is a closed expression. We use E to range over programs. The labels are used solely to identify occurrences of terms; they do not influence the evaluation of a program.

We will use the function L which maps a λ-term to its topmost label:

$$\begin{array}{rcl} L(x^l) & = & l \\ L(\lambda^l x.e) & = & l \\ L(e_1 \,@^l\, e_2) & = & l \\ L(e_0 \to^l e_1, e_2) & = & l \\ L(q^l) & = & l. \end{array}$$

We will study the following 0-CFA-style flow analysis. Given a program E, we let $Flow(E)$ denote the powerset of the set of labels of λ-abstractions occurring in E. Moreover, we let $FlowDom(E)$ denote the union of the set of labels of occurrences of subterms of E, and the set

$$\{ \ U_l \ \mid \ l \text{ is the label of a } \lambda\text{-abstraction in } E \ \}.$$

If l is the label of a λ-abstraction in E, then we use U_l as an implicit label of the variable bound by that λ-abstraction. An alternative would be to use the *name* of the variable as a label; we prefer to keep the name and the label separate.

A *flow analysis* of E is a total mapping

$$\varphi : FlowDom(E) \to Flow(E)$$

such that

- for each x^l occurring in E and bound by a λ-abstraction labeled l', we have $\varphi(U_{l'}) = \varphi(l)$;
- for each $\lambda^l x.e$ occurring in E, we have $l \in \varphi(l)$;
- for each $e_1 \mathbin{@^{l'}} e_2$ and each $\lambda^l x.e$ occurring in E, we have that if $l \in \varphi(L(e_1))$ then

$$\begin{aligned}\varphi(L(e_2)) &\subseteq \varphi(U_l)\\ \varphi(L(e)) &\subseteq \varphi(l');\end{aligned}$$

- for each $e_0 \rightarrow^l e_1, e_2$ occurring in E, we have

$$\begin{aligned}\varphi(L(e_1)) &\subseteq \varphi(l)\\ \varphi(L(e_2)) &\subseteq \varphi(l).\end{aligned}$$

Notice that there are no constraints for constants. The domain $\mathit{FlowDom}(E) \rightarrow \mathit{Flow}(E)$ is ordered by the pointwise ordering of functions induced by set inclusion. We denote this ordering by $\subseteq$. For a given program E, there is a $\subseteq$-least flow analysis which can be computed in $O(n^3)$ time where n is the size of the program [15]. The set of flow analyses of E is denoted by $\mathit{FlowAnalysis}(E)$.

For example, for the program $\lambda^1 x.\mathrm{true}^2$, we have

$$\begin{aligned}\mathit{FlowDom}(E) &= \{\ 1, 2\ \}\\ \mathit{Flow}(E) &= \{\ \emptyset, \{1\}\ \}\end{aligned}$$

and two flow analyses φ, φ' where:

$$\begin{aligned}\varphi(1) &= \{1\}\\ \varphi(2) &= \emptyset\\ \varphi'(1) &= \{1\}\\ \varphi'(2) &= \{1\}.\end{aligned}$$

and $\varphi \subseteq \varphi'$. Notice that the constraint on λ-abstractions forces both φ and φ' to map the label 1 to the set $\{1\}$.

Notice that the constraint for a variable occurrence is an equality rather the inclusion $\varphi(U_{l'}) \subseteq \varphi(l)$. Both choices lead to the same $\subseteq$-least flow analysis for a given program, see Appendix C. We have chosen the equality constraint because it enables a simple proof of Theorem 2, see below.

The function cps transforms a whole program E to a CPS target term:

$$\begin{aligned}\mathsf{cps}(E) &= \lambda^{X_{L(E)}} k.[\![E]\!] \mathbin{@^{B_{L(E)}}} (\lambda^{R_{L(E)}} v.k^{Y_{L(E)}} \mathbin{@^{Z_{L(E)}}} v^{V_{L(E)}})\\ [\![x^l]\!] &= \lambda^{P_l} k.k^{K_l} \mathbin{@^{A_l}} x^l\\ [\![\lambda^l x.e]\!] &= \lambda^{P_l} k.k^{K_l} \mathbin{@^{A_l}} (\lambda^l x.\lambda^{Q_l} m.[\![e]\!] \mathbin{@^{B_{L(e)}}} (\lambda^{R_{L(e)}} v.m^{M_l} \mathbin{@^{C_l}} v^{V_{L(e)}}))\end{aligned}$$

$$
\begin{aligned}
[\![e_1 \,@^l\, e_2]\!] &= \lambda^{P_l} k.[\![e_1]\!] \,@^{B_{L(e_1)}}\, (\lambda^{R_{L(e_1)}} v_1.[\![e_2]\!] \,@^{B_{L(e_2)}}\, (\lambda^{R_{L(e_2)}} v_2. \\
&\qquad (v_1^{V_{L(e_1)}} \,@^{G_l}\, v_2^{V_{L(e_2)}}) \,@^{D_l}\, (\lambda^{S_l} v.k^{K_l} \,@^{A_l}\, v^l))) \\
[\![e_0 \to^l e_1, e_2]\!] &= \lambda^{P_l} k.(\lambda^{J_l} m.[\![e_0]\!] \,@^{B_{L(e_0)}}\, (\lambda^{R_{L(e_0)}} w.w^{V_{L(e_0)}} \to^{F_l} \\
&\qquad\qquad [\![e_1]\!] \,@^{B_{L(e_1)}}\, (\lambda^{R_{L(e_1)}} v_1.m^{H_l^1} \,@^{N_l^1}\, v_1^{V_{L(e_1)}}), \\
&\qquad\qquad [\![e_2]\!] \,@^{B_{L(e_2)}}\, (\lambda^{R_{L(e_2)}} v_2.m^{H_l^2} \,@^{N_l^2}\, v_2^{V_{L(e_2)}})) \\
&\qquad) \,@^{W_l}\, (\lambda^{T_l} v.k^{K_l} \,@^{A_l}\, v^l) \\
[\![q^l]\!] &= \lambda^{P_l} k.k^{K_l} \,@^{A_l}\, q^l.
\end{aligned}
$$

Notice that the CPS translation does not duplicate contexts across conditional expressions, as is sometimes done by CPS transformations used in compilers [1, 5].

All the CPS terms that can result from the CPS transformation can be generated from the following grammar:

$$
\begin{aligned}
\text{Simple} &::= x^l \mid q^l \mid \lambda^l x.\text{Tail} \\
\text{Tail} &::= \text{Simple} \mid \text{Tail} \,@^l\, \text{Simple} \mid \text{Simple} \to^l \text{Tail}, \text{Tail}.
\end{aligned}
$$

The grammar can also generate terms that are not the result of the CPS transformation. While tighter grammars can be constructed, we hope that our grammar may help build intuition about the structure of CPS terms.

The labeling of occurrences of subterms in $\mathsf{cps}(E)$ follows three guidelines. For every occurrence in E of a subterm e with label l, we have:

- $[\![e]\!] = \lambda^{P_l} k.e'$, where $L(e')$ is either $A_{l'}$, $B_{l'}$, or $W_{l'}$, and each occurrence of k is labeled K_l;
- $[\![e]\!]$ occurs in $\mathsf{cps}(E)$ as part of a subterm $[\![e]\!] \,@^{B_{L(e)}}\, (\lambda^{R_{L(e)}} v.\ldots)$, where the body of $\lambda^{R_{L(e)}}\ldots$ is labeled with one of $Z_{l'}$, $C_{l'}$, $B_{l'}$, $D_{l'}$, $F_{l'}$, $N_{l'}^1$, $N_{l'}^2$, and the single occurrence of v is labeled $V_{L(e)}$; and
- there is a subterm $k^{K_l} \,@^{A_l}\, e'$, where $L(e') = l$.

The result of using the guidelines is that:

- $[\![e]\!]$ is a λ-abstraction with label $P_{L(e)}$ which will be applied at an application point labeled $B_{L(e)}$ and
- $[\![e]\!]$ will take an argument (a continuation) with label $R_{L(e)}$ which will be applied at an application point labeled $A_{L(e)}$.

In the setting of simply-typed λ-calculus, these consequences can be stated in a particularly succinct way, see Theorem 4 below. The use of the guidelines also enables succinct proofs, particularly of Lemma 6; see below.

Note that:

$$
\begin{aligned}
\mathit{FlowDom}(E) &\subseteq \mathit{FlowDom}(\mathsf{cps}(E)) \\
\mathit{Flow}(E) &\subseteq \mathit{Flow}(\mathsf{cps}(E)).
\end{aligned}
$$

If all occurrences of subterms of E are labeled distinctly, then the same is true of $\mathsf{cps}(E)$. We will henceforth assume that the occurrences of subterms of E are labeled distinctly and only with constant label symbols.

The function $(\cdot)^*$ transforms flow information for a program E to flow information for the program $\mathsf{cps}(E)$:

$$(\cdot)^* : (FlowDom(E) \to Flow(E)) \to (FlowDom(\mathsf{cps}(E)) \to Flow(\mathsf{cps}(E))).$$

We will present the definition of φ^* in table form.

- For the whole program E, where $L(E) = l$:

a	$\varphi^*(a)$
X_l	$\{X_l\}$
Y_l	$\emptyset$
Z_l	$\emptyset$
U_{X_l}	$\emptyset$

- For all labels l of occurrences of subterms of E:

a	$\varphi^*(a)$
l	$\varphi(l)$
P_l	$\{P_l\}$
K_l	$\{R_l\}$
R_l	$\{R_l\}$
V_l	$\varphi(l)$
A_l	$\emptyset$
B_l	$\emptyset$
U_{P_l}	$\{R_l\}$
U_{R_l}	$\varphi(l)$

- For $\lambda^l x.e$ occurring in E:

a	$\varphi^*(a)$
C_l	$\emptyset$
Q_l	$\{Q_l\}$
M_l	$\{\ S_{l'} \mid e_1 \,@^{l'}\, e_2$ occurs in E and $l \in \varphi(L(e_1))\ \}$
U_{Q_l}	$\{\ S_{l'} \mid e_1 \,@^{l'}\, e_2$ occurs in E and $l \in \varphi(L(e_1))\ \}$
U_l	$\varphi(U_l)$

- For $e_1 \,@^l\, e_2$ occurring in E:

a	$\varphi^*(a)$
D_l	$\emptyset$
S_l	$\{S_l\}$
G_l	$\{\ Q_{l'} \mid l' \in \varphi(L(e_1))\ \}$
U_{S_l}	$\varphi(l)$

- For $e_0 \to^l e_1, e_2$ occurring in E:

a	$\varphi^*(a)$
J_l	$\{J_l\}$
U_{J_l}	$\{T_l\}$
F_l	$\emptyset$
H_l^1	$\{T_l\}$
H_l^2	$\{T_l\}$
N_l^1	$\emptyset$
N_l^2	$\emptyset$
W_l	$\emptyset$
T_l	$\{T_l\}$
U_{T_l}	$\varphi(l)$

Informal justification of the cases of l, U_l, M_l, U_{Q_l}, G_l, V_l in the definition of φ^* were given, explicitly or implicitly, in Section 2. We will now provide some intuition for the other cases. First notice that if $L(E) = l$, then

$$\mathsf{cps}(E) \quad = \quad \lambda^{X_l} k. [\![E]\!] @^{B_l} (\lambda^{R_l} v. k^{Y_l} @^{Z_l} v^{V_l}).$$

We are interested in the flow analysis of $\mathsf{cps}(E)$ in an empty context, so $(\lambda^{X_l}\ldots)$ is never applied, and therefore

$$\varphi^*(Y_l) = \varphi^*(Z_l) = \varphi^*(U_{X_l}) = \emptyset.$$

The initial continuation is $(\lambda^{R_l} v. k^{Y_l} @^{Z_l} v^{V_l})$, and since the flow information for the result of applying it is the empty set, that is, $\varphi^*(Z_l) = \emptyset$, we have that 1) the flow information for the result of applying *any* continuation is the empty set, so $\varphi^*(A_l) = \varphi^*(W_l) = \emptyset$; and 2) the flow information for the body of any continuation is the empty set, so for all $a \in \{C_l, B_l, D_l, F_l, N_l^1, N_l^2\}$, we have $\varphi^*(a) = \emptyset$.

For the λ-abstractions occurring in $\mathsf{cps}(E)$, with labels $X_l, P_l, R_l, Q_l, S_l, J_l, T_l$, we have that for any label a in that list, $\varphi^*(a) = \{a\}$.

Suppose $L(e) = l$. We have that $[\![e]\!]$ occurs in $\mathsf{cps}(E)$ as part of a subterm

$$[\![e]\!] @^{B_l} (\lambda^{R_l} v.\ldots),$$

and that

$$[\![e]\!] = \lambda^{P_l} k.\ldots,$$

so the unique argument supplied to the λ-abstraction with label P_l is another λ-abstraction with label R_l, hence $\varphi^*(U_{P_l}) = \{R_l\}$. Notice that $\mathsf{cps}(E)$ contains the redex

$$(\lambda^{J_l} m.\ldots) @^{W_l} (\lambda^{T_l}\ldots),$$

so $\varphi^*(U_{J_l}) = \{T_l\}$. Finally, the cases of $K_l, U_{R_l}, M_l, U_{S_l}, U_{T_l}, H_l^1, H_l^2$ follow easily from cases we have already explained.

We need one more definition before we can state our results. If $\psi \in FlowDom(\mathsf{cps}(E)) \to Flow(\mathsf{cps}(E))$, then we let ψ_E denote the restriction of ψ to $FlowDom(E)$.

Both $(\cdot)^*$ and $(\cdot)_E$ can be computed in linear time. While this is immediate in the case of $(\cdot)_E$, let us consider how it can be done in the case of $(\cdot)^*$. We assume that the labels in E are presented as numbers from an interval $1..n$, and that we are given the flow information $\varphi \in \mathit{FlowAnalysis}(E)$ as an array over $1..n$ of linked lists of labels. We also require each entry in the array to contain information about the form of the syntax tree node the label stems from, and the labels of the immediate descendants in the syntax tree. We will represent φ^* as a two-dimensional array over $1..n$ and the 32 kinds of derived labels that we use in the definition of φ^*. (The two-dimensional array can easily be flattened to a one-dimensional array, if so desired.) We fill the two-dimensional array during a single traversal of the representation of φ. The main problem in computing φ^* is to compute $\varphi^*(M_l)$ (which is equal to $\varphi^*(U_{Q_l})$), for an occurrence of $\lambda^l x.e$ in E. This can be done by, for each label of an occurrence of an application $e_1 @^{l'} e_2$, finding the entry $\varphi(L(e_1))$, and for each element of the list for $\varphi(L(e_1))$, extending the list for M_l with $S_{l'}$. This requires an amount of work which for each application $e_1 @^{l'} e_2$ is linear in the size of $\varphi(L(e_1))$, so the total amount of work is linear in the size of φ.

We now present our main results.

Theorem 1. *If $\varphi \in \mathit{FlowAnalysis}(E)$, then $\varphi^* \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$. Moreover, if $\psi \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$, then $\psi_E \in \mathit{FlowAnalysis}(E)$.*

Proof. See Appendix A. □

Based on Theorem 1, we will from now on consider the two mappings $(\cdot)^*$ and $(\cdot)_E$ to have the functionalities:

$$\begin{aligned} (\cdot)^* &: \mathit{FlowAnalysis}(E) \to \mathit{FlowAnalysis}(\mathsf{cps}(E)) \\ (\cdot)_E &: \mathit{FlowAnalysis}(\mathsf{cps}(E)) \to \mathit{FlowAnalysis}(E), \end{aligned}$$

and we will ignore that both were originally defined on larger domains.

The following theorem states two basic relationships between $(\cdot)^*$ and $(\cdot)_E$. The first part of the theorem says that $(\cdot)^*$ followed by $(\cdot)_E$ is the identity on $\mathit{FlowAnalysis}(E)$. The second part of the theorem says that if we (1) analyze $\mathsf{cps}(E)$, (2) restrict the result to the source term E using $(\cdot)_E$, and then (3) build an analysis of $\mathsf{cps}(E)$ using $(\cdot)^*$, then we get a result that is at least as good as the initial analysis of $\mathsf{cps}(E)$. The reason is that $(\cdot)^*$ chooses the best solution for the labels in $\mathit{FlowDom}(\mathsf{cps}(E)) \setminus \mathit{FlowDom}(E)$.

Theorem 2. *If $\varphi \in \mathit{FlowAnalysis}(E)$, then $(\varphi^*)_E = \varphi$. Moreover, if $\psi \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$, then $(\psi_E)^* \subseteq \psi$.*

Proof. See Appendix B. □

Both $(\cdot)^*$ and $(\cdot)_E$ are monotone functions, and since they satisfy Theorem 2, it is an immediate consequence that they form a Galois connection, that is, for all $\varphi \in \mathit{FlowAnalysis}(E)$, and for all $\psi \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$, we have $\varphi^* \subseteq \psi$ if and only if $\varphi \subseteq \psi_E$.

Theorem 3. (Main Result) *If φ is the least flow analysis of a program E, and ψ is the least flow analysis of* $\mathsf{cps}(E)$*, then* $\varphi^* = \psi$ *and* $\varphi = \psi_E$.

Proof. From φ being least we have $\varphi \subseteq \psi_E$. From ψ being least we have $\psi \subseteq \varphi^*$. Since $(\cdot)^*$ and $(\cdot)_E$ form a Galois connection, $\varphi \subseteq \psi_E$ implies $\varphi^* \subseteq \psi$. Hence $\varphi^* = \psi$. From that and Theorem 2 we have $\psi_E = (\varphi^*)_E = \varphi$. $\square$

4 Flow Types

We have shown that 0-CFA-style flow information can be maintained by CPS transformation. This parallels the classical result that typability with simple types can be maintained by CPS transformation. We will now show that typability with *flow types* can be maintained by CPS transformation. Flow types have been studied by Tang and Jouvelot [17], Heintze [6], Wells, Dimock, Muller, and Turbak [20, 18], and others. The idea is that if an expression has the flow type

$$s \xrightarrow[\sigma]{\pi} t$$

then π is a set of labels of λ-abstractions to which the expression can evaluate, and σ is a set of labels of application points where those λ-abstractions can be applied.

We will consider flow types, in the style of [20], generated by the grammar:

$$t \ ::= \ \alpha \ \mid \ t \xrightarrow[\sigma]{\pi} t,$$

where α ranges over a set of base types that includes $\mathsf{boolean}$, and where π, σ range over finite subsets of *Lab*. Among the types is a distinguished type o of *answers*. A type environment is a partial function with finite domain which maps λ-variables to types. We use the notation $\mathcal{A}[x : \tau]$ to denote an environment which maps x to τ, and maps y, where $y \neq x$, to $\mathcal{A}(y)$. A type judgment has the form $\mathcal{A} \vdash e : t$, and it means that in the type environment $\mathcal{A}$, the expression e has type t. Formally, this holds when it is derivable by a finite derivation-tree using the rules below, taken from [20].

$$\mathcal{A}[x : t] \vdash x : t \tag{1}$$

$$\frac{\mathcal{A}[x : s] \vdash e : t}{\mathcal{A} \vdash \lambda^l x.e : s \xrightarrow[\sigma]{\pi} t} \quad (l \in \pi) \tag{2}$$

$$\frac{\mathcal{A} \vdash e_1 : s \xrightarrow[\sigma]{\pi} t \quad \mathcal{A} \vdash e_2 : s}{\mathcal{A} \vdash e_1 @^l e_2 : t} \quad (l \in \sigma) \tag{3}$$

$$\frac{\mathcal{A} \vdash e_0 : \mathsf{boolean} \quad \mathcal{A} \vdash e_1 : t \quad \mathcal{A} \vdash e_2 : t}{\mathcal{A} \vdash e_0 \to^l e_1, e_2 : t} \tag{4}$$

$$\mathcal{A} \vdash q^l : \mathsf{boolean} \tag{5}$$

If $\sigma \subseteq Lab$, then we write $A_\sigma = \{ A_l \mid l \in \sigma \}$, and similarly for other unary operations on Lab. We can now define a CPS transformation of flow types:

$$\begin{array}{rcl} \alpha^* & = & \alpha \\ (s \xrightarrow[\sigma]{\pi} t)^* & = & s^* \xrightarrow[G_\sigma]{\pi} (t^* \xrightarrow[C_\pi]{S_\sigma} o) \xrightarrow[D_\sigma]{Q_\pi} o \; . \end{array}$$

Define also $\mathcal{A}^*(x) = t^*$ if $\mathcal{A}(x) = t$.

Theorem 4. *If* $\mathcal{A} \vdash e : t$, *then* $\mathcal{A}^* \vdash [\![e]\!] : (t^* \xrightarrow[\{A_{L(e)}\}]{\{R_{L(e)}\}} o) \xrightarrow[\{B_{L(e)}\}]{\{P_{L(e)}\}} o.$

Proof. By induction on the structure of the derivation of $A \vdash e : t$. □

Notice that Theorem 4 corresponds to the first half of Theorem 1. We do not have counterparts to the rest of our results for the untyped case.

5 Related Work

Nielson [10] showed for an imperative language that an analysis based on a continuation semantics can be more precise than an analysis based on a direct semantics. Similarly, Muylaert-Filho and Burn [9] showed for a call-by-name language that CPS transformation can improve an analysis.

Sabry and Felleisen [16] concluded that the gain in precision in these examples is solely due to the duplication of the analysis of continuations. For example, at a call site and at a conditional the continuation may be duplicated for each possible path, thereby enabling separate analysis for each copy. Thus a flow-insensitive analysis of the transformed program corresponds to a flow-sensitive analysis of the original. One contribution of the current paper is an attempt to separate the consequences of the CPS transformation from those of this duplication.

Sabry and Felleisen [16] gave an example in which a CPS transformation *decreased* the precision of an analysis. They studied a CPS transformation for a call-by-value language, together with a flow analysis. Their example program is:

$$(\mathsf{let}\ (a_1\ (f\ 1))\ (\mathsf{let}\ (a_2\ (f\ 2))\ a_2))$$

where f is bound to $\lambda x.x$. After their CPS transformation, the program becomes:

$$(f\ 1\ (\lambda a_1.(f\ 2\ (\lambda a_2.(k\ a_2)))))$$

where f is bound to $\lambda x.\lambda k_1.(k_1\ x)$, and k is bound to a continuation. Before CPS transformation the analysis finds that a_1 is constant $(= 1)$, while after CPS transformation the analysis fails to find that information.

However, the analysis of Sabry and Felleisen is an *operational* abstract interpretation of a program:

- For the source program, the analysis first reaches the call site $(f\ 1)$, and at this point x gets bound to 1, and thus a_1 gets bound to 1. When the analysis later reaches the call site $(f\ 2)$, the new value for x becomes a merge of the old value 1 and the new value 2. The value of a_1 is unchanged.

- For the CPS program, the analysis first performs the call

  $$(f\ 1\ (\lambda a_1.(f\ 2\ (\lambda a_2.(k\ a_2)))))$$

  where x gets bound to 1 and k_1 gets bound to $(\lambda a_1.(f\ 2\ (\lambda a_2.(k\ a_2))))$. A little later, the analysis performs the call $(f\ 2\ (\lambda a_2.(k\ a_2)))$, the new value for x becomes a merge of 1 and 2, and the new value for k_1 becomes a set consisting of both $(\lambda a_1.(f\ 2\ (\lambda a_2.(k\ a_2))))$ and $(\lambda a_2.(k\ a_2))$. Finally, when the analysis in the end performs the call $(k_1\ x)$, the analysis applies each of the two continuations to the value of x and merges the results. During that final call, both a_1 and a_2 get bound to the merge of 1 and 2.

Sabry and Felleisen state that "the loss of information is due to the confusion of distinct procedure returns" [16]. For example, the two calls $(f\ 1)$ and $(f\ 2)$ become confused in the CPS program.

The operational nature of this process reintroduces flow-sensitivity into the analysis. We instead use a monovariant, constraint-based flow analysis, and we get the same result for corresponding program points in the two programs. If the analysis is monovariant, then there is exactly one flow variable for each occurrence of an expression in the program. There is no notion of "getting to $(f\ 1)$ before getting to $(f\ 2)$." In the source program above, there is one flow variable for x and one flow variable for a_1, and in the least solution of the constraints, both will be assigned a merge of 1 and 2. Thus, "confusion of distinct procedure returns" happens both when we use a monovariant flow analysis on the source program and when we use it on the CPS program.

Our result supports the conclusion of Sabry and Felleisen that the improvement in analysis in the transformed program is due to the duplication of program points.

6 Concluding Remarks

It seems to be straightforward to extend our result to a language with more features. Our experience is a good sign of that: we first proved our result for the language *without* conditional expressions, while codifying and using the three guidelines for the labeling of occurrences of subexpressions, and we then used the guidelines to extend the result to conditional expressions. We found that the statement and proof of Lemma 6 needed no change at all, and the proofs of the other lemmas could be extended easily to cover the new cases.

Our results depend on our particular formulation of the CPS transformation. Other formulations of the CPS transformation may be derived from ours by applying administrative reductions to the CPS terms. Least solutions to the 0-CFA constraints are preserved by administrative reductions [3].

Future work may include extending our results to polyvariant flow analysis, and investigations of whether flow information is preserved by call-by-name CPS transformations.

Acknowledgments Thanks to Fritz Henglein and Didier Rémy for helpful discussions. Thanks to Bob Muller and Joe Wells for the LaTeX macro for formatting $\xrightarrow[\sigma]{\pi}$. Palsberg was supported by a National Science Foundation Faculty Early Career Development Award, CCR–9734265. Wand was supported in part by the National Science Foundation under grants CCR-9629801 and CCR-9804115.

Appendix A: Proof of Theorem 1

Theorem 1 follows immediately from Lemmas 5 and 7. The proof of Lemma 7 uses Lemma 6. Lemma 6 is also used in Appendix B.

Lemma 5. *If $\varphi \in \mathit{FlowAnalysis}(E)$, then $\varphi^* \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$.*

Proof. We proceed by case analysis on the expressions in $\mathsf{cps}(E)$. Consider first an expression in $\mathsf{cps}(E)$ of the form x^l, bound by a λ-abstraction labeled l'. There are two cases. If $\ell \in \mathit{FlowDom}(E)$, then $\varphi(U_{l'}) = \varphi(l)$, and from the definition of φ^* we have $\varphi^*(U_{l'}) = \varphi(U_{l'})$ and $\varphi^*(l) = \varphi(l)$, so we conclude $\varphi^*(U_{l'}) = \varphi^*(l)$. If $l \notin \mathit{FlowDom}(E)$, then a straightforward case analysis of l shows that $\varphi^*(U_{l'}) = \varphi^*(l)$.

Consider next the eight forms of λ-abstractions in $\mathsf{cps}(E)$. For $\lambda^a x.e$, where $a \in \{X_l, P_l, R_l, Q_l, S_l, J_l, T_l\}$, it is immediate from the definition of φ^* that $a \in \{a\} = \varphi^*(a)$. The remaining case is

$$[\![\lambda^l x.e]\!] = \ldots(\lambda^l x.\ldots)\ldots.$$

By assumption we have $l \in \varphi(l)$, and from the definition of φ^* we have $\varphi^*(l) = \varphi(l)$, so we conclude $l \in \varphi^*(l)$.

Consider next the eight forms of applications in $\mathsf{cps}(E)$:

$$
\begin{aligned}
& & & k^{Y_{L(E)}} @^{Z_{L(E)}} v^{V_{L(E)}} && (6)\\
& & & (\lambda^{P_l} k.e') @^{B_l} (\lambda^{R_l}\ldots) && (7)\\
& & & k^{K_l} @^{A_l} e' && (8)\\
[\![e_1 @^{l'} e_2]\!] &=& & \ldots {v_1}^{V_{L(e_1)}} @^{G_{l'}} {v_2}^{V_{L(e_2)}} \ldots && (9)\\
[\![e_1 @^{l'} e_2]\!] &=& & \ldots ({v_1}^{V_{L(e_1)}} @^{G_{l'}} v_2^{V_{L(e_2)}}) @^{D_{l'}} (\lambda^{S_{l'}}\ldots)\ldots && (10)\\
[\![\lambda^l x.e']\!] &=& & \ldots m^{M_l} @^{C_l} v^{V_{L(e')}} \ldots && (11)\\
[\![e_0 \to^l e_1, e_2]\!] &=& & \ldots (\lambda^{J_l} m.e') @^{W_l} (\lambda^{T_l}\ldots.) && (12)\\
[\![e_0 \to^l e_1, e_2]\!] &=& & \ldots m^{H_l^i} @^{N_l^i} v_i^{V_{L(e_i)}} \ldots, \quad i \in \{1,2\}. && (13)
\end{aligned}
$$

We consider each of them in turn:

- (6). We have $\varphi^*(Y_{L(E)}) = \emptyset$, so φ^* has the desired property.
- (7). We have $\varphi^*(P_l) = \{P_l\}$. A case analysis of the possibilities for the body e' shows that $L(e')$ is of the form A_a, B_a, or W_a, hence $\varphi^*(L(e')) = \emptyset$. We conclude $\varphi^*(R_l) = \{R_l\} = \varphi^*(U_{P_l})$ and $\varphi^*(L(e')) = \emptyset = \varphi^*(B_l)$.
- (8). We have $\varphi^*(K_l) = \{R_l\}$. A case analysis of the possibilities for the body, say e'', of the λ-abstraction with label R_l shows that $L(e'')$ is of the form Z_a, B_a, C_a, D_a, F_a, N_a^1, or N_a^2, hence $\psi^*(L(e'')) = \emptyset$. Notice that the argument e' has the property $L(e') = l$. We conclude $\varphi^*(l) = \varphi(l) = \varphi^*(U_{R_l})$ and $\varphi^*(L(e'')) = \emptyset = \varphi^*(A_l)$.

- (9). We have $\varphi^*(V_{L(e_1)}) = \varphi(L(e_1))$. Suppose $l \in \varphi^*(V_{L(e_1)})$ and suppose that in $\mathsf{cps}(E)$ we have $(\lambda^l x.\lambda^{Q_l} \ldots)$ and that in E we have $(\lambda^l x.e)$. From $\varphi \in \mathit{FlowAnalysis}(E)$ and $l \in \varphi(L(e_1))$ we have $\varphi(L(e_2)) \subseteq \varphi(U_l)$. We conclude $\varphi^*(V_{L(e_2)}) = \varphi(L(e_2)) \subseteq \varphi(U_l) = \varphi^*(U_l)$ and $\varphi^*(Q_l) = \{Q_l\} \subseteq \{\ Q_a \mid a \in \varphi(L(e_1))\ \} = \varphi^*(G_{l'})$.

- (10). We have $\varphi^*(G_{l'}) = \{\ Q_l \mid l \in \varphi(L(e_1))\ \}$. Suppose $Q_l \in \varphi^*(G_{l'})$, hence $l \in \varphi(L(e_1))$. In $\mathsf{cps}(E)$ we have $(\lambda^{Q_l} m.e)$ and we have $\varphi^*(L(e)) = \emptyset$. We conclude $\varphi^*(S_{l'}) = \{S_{l'}\} \subseteq \{\ S_{l'} \mid e_1 \,@^{l'}\, e_2$ occurs in E and $l \in \varphi(L(e_1))\ \} = \varphi^*(U_{Q_l})$ and $\varphi^*(L(e)) = \emptyset = \varphi^*(D_{l'})$.

- (11). We have $\varphi^*(M_l) = \{\ S_{l'} \mid e_1 \,@^{l'}\, e_2$ occurs in E and $l \in \varphi(L(e_1))\ \}$. Suppose $S_{l'} \in \varphi^*(M_l)$. In $\mathsf{cps}(E)$ we have $(\lambda^{S_{l'}} v.k^{K_{l'}} \,@^{A_{l'}}\, v^{l'})$. From $\varphi \in \mathit{FlowAnalysis}(E)$ and $l \in \varphi(L(e_1))$ we have $\varphi(L(e')) \subseteq \varphi(l')$. We conclude $\varphi^*(V_{L(e')}) = \varphi(L(e')) \subseteq \varphi(l') = \varphi^*(U_{S_{l'}})$ and $\varphi^*(A_{l'}) = \emptyset = \varphi^*(C_l)$.

- (12). We have $\varphi^*(J_l) = \{J_l\}$. Moreover, $L(e') = B_{L(e_0)}$, hence $\varphi^*(L(e')) = \emptyset$. We conclude $\varphi^*(T_l) = \{T_l\} = \varphi^*(U_{J_l})$ and $\varphi^*(L(e')) = \emptyset = \varphi^*(W_l)$.

- (13). Suppose $i \in \{1, 2\}$. We have $\varphi^*(H_l^i) = \{T_l\}$. The body, say e'', of the λ-abstraction with label T_l satisfies $L(e'') = A_l$, hence $\varphi^*(L(e'')) = \emptyset$. From $\varphi \in \mathit{FlowAnalysis}(E)$ we have $\varphi(L(e_i)) \subseteq \varphi(l)$. We conclude $\varphi^*(V_{L(e_i)}) = \varphi(L(e_i)) \subseteq \varphi(l) = \varphi^*(U_{T_l})$.

Consider finally conditionals in $\mathsf{cps}(E)$:

$$[\![e_0 \to^l e_1, e_2]\!] \quad = \quad \ldots \to^{F_l} (\ldots @^{B_{L(e_1)}} \ldots), (\ldots @^{B_{L(e_2)}} \ldots) \ldots$$

We have $\varphi^*(B_{L(e_i)}) = \emptyset = \varphi^*(F_l)$, where $i \in \{1, 2\}$. □

Lemma 6. *If $\psi \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$, and l is a label of an occurrence of a subterm of E, then $\psi(l) \subseteq \psi(V_l)$.*

Proof. For each l which labels an occurrence of a subterm of E, we have in $\mathsf{cps}(E)$ the expressions:

$$[\![e]\!] \,@^{B_l}\, (\lambda^{R_l} \ldots) \tag{14}$$

$$k^{K_l} \,@^{A_l}\, e' \tag{15}$$

where $L(e) = L(e') = l$. From $L([\![e]\!]) = P_{L(e)} = P_l$ and (14) we have $R_l \in \psi(R_l) \subseteq \psi(U_{P_l})$. We have that k^{K_l} is bound by a λ-abstraction with label P_l, so $\psi(U_{P_l}) = \psi(K_l)$. From $R_l \in \psi(U_{P_l}) = \psi(K_l)$ and (15) we have $\psi(l) \subseteq \psi(U_{R_l})$. The variable occurrence labeled V_l is bound by a λ-abstraction labeled R_l, so $\psi(U_{R_l}) = \psi(V_l)$. We conclude $\psi(l) \subseteq \psi(U_{R_l}) = \psi(V_l)$. □

Lemma 7. *If $\psi \in FlowAnalysis(\mathsf{cps}(E))$, then $\psi_E \in FlowAnalysis(E)$.*

Proof. We proceed by case analysis on the expressions in E. Consider first an expression in E of the form x^l, bound by a λ-abstraction labeled l'. We have

$$[\![x^l]\!] = \ldots x^l \ldots$$

where x^l is bound in $\mathsf{cps}(E)$ by a λ-abstraction labeled l', so we have $\psi(U_{l'}) = \psi(l)$, hence $\psi_E(U_{l'}) = \psi_E(l)$.

Consider next an expression in E of the form $\lambda^l x.e$. We have

$$[\![\lambda^l x.e]\!] = \ldots (\lambda^l x.\ldots) \ldots$$

so we have $l \in \psi(l)$, hence $l \in \psi_E(l)$.

Consider next an expression in E of the form $e_1 \mathbin{@^{l'}} e_2$. Suppose we have $\lambda^l x.e$ in E such that $l \in \psi_E(L(e_1))$. We need to show:

$$\begin{aligned} \psi_E(L(e_2)) &\subseteq \psi_E(U_l) \\ \psi_E(L(e)) &\subseteq \psi_E(l'). \end{aligned}$$

We have in $\mathsf{cps}(E)$ the expressions:

$$[\![e_1 \mathbin{@^{l'}} e_2]\!] = \ldots {v_1}^{V_{L(e_1)}} \mathbin{@^{G_{l'}}} {v_2}^{V_{L(e_2)}} \ldots \tag{16}$$

$$[\![e_1 \mathbin{@^{l'}} e_2]\!] = \ldots ({v_1}^{V_{L(e_1)}} \mathbin{@^{G_{l'}}} v_2^{V_{L(e_2)}}) \mathbin{@^{D_{l'}}} (\lambda^{S_{l'}} v.\ldots v^{l'} \ldots) \ldots \tag{17}$$

$$[\![\lambda^l x.e]\!] = \ldots m^{M_l} \mathbin{@^{C_l}} v^{V_{L(e)}} \ldots. \tag{18}$$

From Lemma 6 we have $\psi(L(e_1)) \subseteq \psi(V_{L(e_1)})$. It follows that $l \in \psi_E(L(e_1)) = \psi(L(e_1)) \subseteq \psi(V_{L(e_1)})$, so from (16) and the observation that the body of the λ-abstraction labeled l is labeled Q_l, we have $\psi(V_{L(e_2)}) \subseteq \psi(U_l)$ and $\psi(Q_l) \subseteq \psi(G_{l'})$. From Lemma 6 we have $\psi(L(e_2)) \subseteq \psi(V_{L(e_2)})$, so $\psi_E(L(e_2)) = \psi(L(e_2)) \subseteq \psi(V_{L(e_2)}) \subseteq \psi(U_l) = \psi_E(U_l)$. From $Q_l \in \psi(Q_l) \subseteq \psi(G_{l'})$ and (17) we have $\psi(S_{l'}) \subseteq \psi(U_{Q_l})$. The variable occurrence labeled M_l is bound by a λ-abstraction labeled Q_l, so $\psi(U_{Q_l}) = \psi(M_l)$. It follows that $S_{l'} \in \psi(S_{l'}) \subseteq \psi(U_{Q_l}) = \psi(M_l)$, so from (18) we have $\psi(V_{L(e)}) \subseteq \psi(U_{S_{l'}})$. From Lemma 6 we have $\psi(L(e)) \subseteq \psi(V_{L(e)})$. The variable occurrence labeled l' is bound by a λ-abstraction labeled $S_{l'}$, so $\psi(U_{S_{l'}}) = \psi(l')$. We conclude $\psi_E(L(e)) = \psi(L(e)) \subseteq \psi(V_{L(e)}) \subseteq \psi(U_{S_{l'}}) = \psi(l') = \psi_E(l')$.

Consider finally an expression in E of the form $e_0 \to^l e_1, e_2$. For $i \in \{1, 2\}$, we have

$$\begin{array}{rcll} \psi_E(L(e_i)) & = & \psi(L(e_i)) & L(e_i) \in FlowDom(E) \\ & \subseteq & \psi(V_{L(e_i)}) & \text{Lemma 6} \\ & \subseteq & \psi(U_{T_l}) & m^{H_l^i} \mathbin{@^{N_l^i}} v_i^{V_{L(e_i)}},\ T_l \in \psi(H_l^i) \\ & = & \psi(l) & v^l \text{ is bound by } \lambda^{T_l}\ldots. \\ & = & \psi_E(l) & l \in FlowDom(E). \end{array}$$

□

Appendix B: Proof of Theorem 2

It is straightforward to show that if $\varphi \in \mathit{FlowAnalysis}(E)$, then $(\varphi^*)_E = \varphi$. We will now prove that if $\psi \in \mathit{FlowAnalysis}(\mathsf{cps}(E))$, then $(\psi_E)^* \subseteq \psi$.

Proof. Suppose $a \in \mathit{FlowDom}(\mathsf{cps}(E))$ and $l \in \mathit{FlowDom}(E)$. We proceed by case analysis on a.

- If $a \in \{X_l, P_l, R_l, Q_l, S_l, J_l, T_l\}$, then $(\psi_E)^*(a) = \{a\}$. Since a is the label of a λ-abstraction we have $a \in \psi(a)$. We conclude $(\psi_E)^*(a) \subseteq \psi(a)$.

- If $a \in \{Y_l, Z_l, U_{X_l}, A_l, B_l, C_l, D_l, F_l, N_l^1, N_l^2, W_l\}$, then $(\psi_E)^*(a) = \emptyset \subseteq \psi(a)$.

- If $a \in \{l, U_l\}$, then $(\psi_E)^*(a) = \psi(a)$.

- If $a \equiv V_l$, then from Lemma 6 we have $(\psi_E)^*(a) = \psi(l) \subseteq \psi(a)$.

- If $a \equiv G_l$ where $e_1 \mathbin{@^l} e_2$ occurs in E, then $(\psi_E)^*(a) = \{\ Q_{l'} \ \mid\ l' \in \psi_E(L(e_1))\ \}$. Suppose $Q_{l'} \in (\psi_E)^*(a)$. From Lemma 6 we have $l' \in \psi_E(L(e_1)) = \psi(L(e_1)) \subseteq \psi(V_{L(e_1)})$. In $\mathsf{cps}(E)$ we have the expression

  $$v_1^{V_{L(e_1)}} \mathbin{@^{G_l}} v_2^{V_{L(e_2)}}$$

  and since the body of the λ-abstraction in $\mathsf{cps}(E)$ labeled l' is labeled $Q_{l'}$, we have $\psi(Q_{l'}) \subseteq \psi(G_l)$. We have $Q_{l'} \in \psi(Q_{l'})$, so we conclude $Q_{l'} \in \psi(G_l)$.

- If $a \equiv U_{Q_l}$, where $\lambda^l x.e$ occurs E, then

  $$(\psi_E)^*(a) = \{\ S_{l'} \ \mid\ e_1 \mathbin{@^{l'}} e_2 \text{ occurs in } E \text{ and } l \in \psi_E(L(e_1))\ \}.$$

  Suppose $S_{l'} \in (\psi_E)^*(a)$, such that $e_1 \mathbin{@^{l'}} e_2$ occurs in E. In $\mathsf{cps}(E)$ we have

  $$[\![e_1 \mathbin{@^{l'}} e_2]\!] = \ldots (v_1^{V_{L(e_1)}} \mathbin{@^{G_{l'}}} v_2^{V_{L(e_2)}}) \mathbin{@^{D_{l'}}} (\lambda^{S_{l'}} \ldots) \ldots.$$

  and we have established already that $Q_l \in \psi(G_{l'})$. This gives $S_{l'} \in \psi(S_{l'}) \subseteq \psi(U_{Q_l})$.

- If $a \equiv U_{P_l}$, then $(\psi_E)^*(a) = \{R_l\}$ and we have in $\mathsf{cps}(E)$

  $$[\![e]\!] \mathbin{@^{B_l}} (\lambda^{R_l} \ldots)$$

  where $L([\![e]\!]) = P_l$. It follows that $P_l \in \psi(L([\![e]\!]))$, so $R_l \in \psi(R_l) \subseteq \psi(U_{P_l})$.

- If $a \equiv U_{J_l}$, then $(\psi_E)^*(a) = \{T_l\}$ and we have in $\mathsf{cps}(E)$:

$$(\lambda^{J^l} \ldots) @^{W_l} (\lambda^{T_l} \ldots).$$

  It follows that $T_l \in \psi(T_l) \subseteq \psi(U_{J_l})$.

- If $a \in \{K_l, U_{R_l}, M_l, U_{S_l}, U_{T_l}, H_l^1, H_l^2\}$, then there is a label $b \in \mathit{FlowDom}(\mathsf{cps}(E))$, namely $U_{P_l}, V_l, U_{Q_l}, l, l, U_{J_l}, U_{J_l}$, respectively, such that $(\psi_E)^*(a) = (\psi_E)^*(b)$ and such that we have already established that $(\psi_E)^*(b) \subseteq \psi(b)$. Moreover, $\psi(a) = \psi(b)$. To see that in the case of $a \equiv K_l$ and $b \equiv U_{P_l}$, notice that K_l is the label of a variable bound by a λ-abstraction labeled P_l, so $\psi(U_{P_l}) = \psi(K_l)$. Similar remarks apply to the other six cases. We conclude $(\psi_E)^*(a) = (\psi_E)^*(b) \subseteq \psi(b) = \psi(a)$.

□

In the last item of the above proof, notice that if the constraint for a variable occurrence were an inclusion rather than an equality, then our proof technique would not work. For example, if $a \equiv U_{R_l}$ and $b \equiv V_l$, then we would have $\psi(U_{R_l}) \subseteq \psi(V_l)$ rather than $\psi(U_{R_l}) = \psi(V_l)$, so we would only derive $\psi(b) \supseteq \psi(a)$.

Appendix C: Equalities versus Inclusions

A *full-blown flow analysis* of E is a total mapping

$$\varphi : \mathit{FlowDom}(E) \to \mathit{Flow}(E)$$

which is defined in the same way as a flow analysis of E, except that for each x^l occurring in E and bound by a λ-abstraction labeled l', we have $\varphi(U_{l'}) \subseteq \varphi(l)$ (rather than $\varphi(U_{l'}) = \varphi(l)$).

Theorem 8. *For a program E, let φ' be the $\subseteq$-least full-blown flow analysis, and let φ be the $\subseteq$-least flow analysis. We have $\varphi' = \varphi$.*

Proof. First notice that every flow analysis is also a full-blown flow analysis. Hence, $\varphi' \subseteq \varphi$. Second, define

$$\varphi''(l) = \begin{cases} \varphi'(U_{l'}) & \text{for } x^l \text{ occurring in } E \text{ and} \\ & \text{bound by a } \lambda\text{-abstraction labeled } l' \\ \varphi'(l) & \text{otherwise.} \end{cases}$$

Since φ' is a flow analysis of E, we have that for any x^l occurring in E and bound by a λ-abstraction labeled l', φ' satisfies $\varphi'(U_{l'}) \subseteq \varphi'(l)$, so $\varphi'' \subseteq \varphi'$.

To see that φ'' is a flow analysis of E, notice φ'' satisfies the constraints for variable occurrences. Notice also that in the remaining constraints, if x^l

occurs in E, then $\varphi''(l)$ occurs in constraints of the form $\varphi''(l) \subseteq \varphi''(l')$ for some l', or $\varphi(l)$ may be used in a conditional statement of the form

$$\text{if } a \in \varphi''(l), \text{ then } \varphi''(l') \subseteq \varphi''(l''),$$

for some l', l''. Thus, from the definition of φ'' and φ' being a flow analysis of E, we have that φ'' is a flow analysis of E.

Since φ is the $\subseteq$-least flow analysis of E, we have that $\varphi \subseteq \varphi''$.

Putting it all together, we have that

$$\varphi'' \subseteq \varphi' \subseteq \varphi \subseteq \varphi'',$$

so $\varphi' = \varphi$. □

References

[1] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[2] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. In *Proceedings of ICFP'00, ACM SIGPLAN International Conference on Functional Programming*, pages 209–220, 2000.

[3] Daniel Damian and Olivier Danvy. CPS transformation of flow information, part II: Administrative reductions. Technical Report RS–01–40, BRICS, University of Aarhus, 2001.

[4] Olivier Danvy and Andrzej Filinski. Representing control, a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, 1992.

[5] Olivier Danvy, Karoline Malmkjær, and Jens Palsberg. Eta-expansion does the trick. *ACM Transactions on Programming Languages and Systems*, 18(6):730–751, November 1996.

[6] Nevin Heintze. Control-flow analysis and type systems. In *Proceedings of SAS'95, International Static Analysis Symposium*, pages 189–206. Springer-Verlag (*LNCS* 983), Glasgow, Scotland, September 1995.

[7] Nevin Heintze and David McAllester. Linear-time subtransitive control flow analysis. In *Proceedings of ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation*, pages 261–272, 1997.

[8] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi. In *Proceedings of Logics of Programs*, pages 219–224. Springer-Verlag (*LNCS* 193), 1985.

[9] J. Muylaert-Filho and G. Burn. Continuation passing transformation and abstract interpretation. In *Proceedings of First Imperial College, Department of Computing, Workshop on Theory and Formal Methods*, 1993.

[10] Flemming Nielson. A denotational framework for data flow analysis. *Acta Informatica*, 18:265–287, 1982.

[11] Jens Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 17(1):47–62, January 1995. Preliminary version in Proceedings of CAAP'94, Colloquium on Trees in Algebra and Programming, Springer-Verlag (*LNCS* 787), pages 276–290, Edinburgh, Scotland, April 1994.

[12] Jens Palsberg. Equality-based flow analysis versus recursive types. *ACM Transactions on Programming Languages and Systems*, 20(6):1251–1264, 1998.

[13] Jens Palsberg and Patrick M. O'Keefe. A type system equivalent to flow analysis. *ACM Transactions on Programming Languages and Systems*, 17(4):576–599, July 1995. Preliminary version in Proceedings of POPL'95, 22nd Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pages 367–378, San Francisco, California, January 1995.

[14] Jens Palsberg and Christina Pavlopoulou. From polyvariant flow information to intersection and union types. *Journal of Functional Programming*, 11(3):263–317, May 2001. Preliminary version in Proceedings of POPL'98, 25th Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pages 197–208, San Diego, California, January 1998.

[15] Jens Palsberg and Michael I. Schwartzbach. *Object-Oriented Type Systems*. John Wiley & Sons, 1994.

[16] Amr Sabry and Matthias Felleisen. Is continuation-passing useful for data flow analysis? In *Proceedings of SIGPLAN'94 Conference on Programming Language Design and Implementation*, pages 1–12, 1994.

[17] Yan Mei Tang and Pierre Jouvelot. Separate abstract interpretation for control-flow analysis. In *Proceedings of TACS'94, Theoretical Aspects of Computing Software*, pages 224–243. Springer-Verlag (*LNCS* 789), 1994.

[18] Franklyn Turbak, Allyn Dimock, Robert Muller, and J. B. Wells. Compiling with polymorphic and polyvariant flow types. In *ACM SIGPLAN Workshop on Types in Compilation*, June 1997. http://www.cs.bc.edu/~muller/postscript/tic97.ps.Z.

[19] Mitchell Wand. Embedding type structure in semantics. In *Proceedings of POPL'85, 12nd Annual Symposium on Principles of Programming Languages*, pages 1–6, 1985.

[20] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A typed intermediate language for flow-directed compilation. In *Proceedings of TAPSOFT'97, Theory and Practice of Software Development.* Springer-Verlag (*LNCS* 1214), 1997.