Syntactic Accidents in Program Analysis *

Daniel Damian and Olivier Danvy
BRICS†
Department of Computer Science, University of Aarhus
Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark
{damian,danvy}@brics.dk

ABSTRACT

We show that a non-duplicating CPS transformation has zero effect on control-flow analysis (i.e., it makes it yield strictly comparable results) and that it has a positive effect on binding-time analysis (i.e., it makes it yield more precise results). Our proof technique amounts to constructing the continuation-passing style (CPS) counterpart of flow information and of binding times.

Our results comfort common intuitions about control-flow analysis, and prove a folklore theorem about binding-time analysis, namely that CPS has no adverse effects, on the contrary, even without duplicating contexts or continuations.

The present study is symptomatic of an unsettling property of program analyses: their quality is unpredictably susceptible to syntactic accidents in source programs. More robust program analyses require a better understanding of the effects of syntactic change.

1. INTRODUCTION AND MOTIVATION

Program analyses are susceptible to syntactic accidents in source programs in that innocent-looking, meaning-preserving transformations may substantially alter the precision of an analysis.

For a simple example, binding-time analysis (BTA) is susceptible to associativity: given two static expressions s_1 and s_2 and one dynamic expression d, it makes a difference whether one analyzes $(s_1 + s_2) + d$ or $s_1 + (s_2 + d)$. In the former case, the inner addition is classified as static and the outside addition is classified as dynamic. In the latter case, both occurrences are classified as dynamic.

*Draft, 00-03-16. Please do not distribute. Comments are welcome.

†Basic Research in Computer Science (www.brics.dk), Centre of the Danish National Research Foundation.

With the exception of BTA, little is known about the impact of programming style on program analyses. And indeed little can be known at all, since typically such analyses are buried, e.g., in a compiler where nobody can check out, let alone control, the quality of their result.

BTA is an exception because its output critically determines the amount of specialization carried out by an offline partial evaluator [5, 18]. For this reason, partial-evaluation users are encouraged to write programs in a very specific style [17]. Furthermore, a genuine panoply of tricks has been developed to milk out extra precision from binding-time analyses—the so-called "binding-time-improvements" [18, Chapter 12]. We note that binding-time-improvements are not restricted to users of offline partial-evaluators—they are also a routine task in the formal specification of programming languages for semantics-directed compiling [22, Section 8.2].

Since one of the most effective binding-time improvements is the transformation into continuation-passing style (CPS) [3, 32], it has therefore been wondered whether CPS is a boon for program analysis in general. Nielson's early work on data flow analysis [21] suggests so, since it shows that a continuation semantics yields the best possible precision even for a non-distributive analysis, whereas a direct semantics yields the second-best precision. Since given a direct semantics, a source CPS transformation gives the effect of a continuation semantics, the CPS transformation is therefore A Good Thing. In the early 90s, further work by Muylaert-Filho and Burn [20] was providing further indication of the value of the CPS transformation for abstract interpretation when Sabry and Felleisen entered the scene.

In their stunning article "Is continuation-passing useful for data-flow analysis?" [31], Sabry and Felleisen showed that for constant propagation, analyzing direct-style programs or their CPS counterpart yield incomparable results. They showed that CPS might improve results from duplicating continuations, and also that CPS might worsen the results by confusing return points. These results are essentially confirmed by Palsberg and Wand's CPS-transformation of flow information [30]. At any rate, except for continuation-based partial evaluation [13], there seems to have been no further work about the effect of CPS on the precision of program analysis in general.

The situation is thus that the CPS transformation is known to have an impredictable effect on data-flow analysis and is believed to have a positive effect on binding-time analysis. However, we do not know for sure whether this positive effect is truly positive, i.e., whether it does not worsen binding times elsewhere in the source program. One may also wonder whether there exist analyses on which CPS has zero effect.

In this article, we answer to these two questions by studying the effect of the CPS transformation on two constraint-based program analyses—0-CFA and BTA. Using a uniform proof technique, we formally show that:

(1) CPS has zero effect on 0-CFA, i.e., it yields precisely similar results; and that:

(2) CPS does not make BTA yield less precise results, and for the class of examples for which continuation-based partial evaluation was developed, it makes BTA yield results that are strictly more precise.

This increased precision also concerns analyses that have been noticed to be structurally similar to BTA, such as security analysis, program slicing, and call tracking [1]. These analyses also appear to be similar in use: we are told that in practice, security-analysis users tend to find these analyses too conservative, without quite knowing what to do to obtain more precise results. (Here, "more precise results" means that more parts of the source program should be classified as low.)

Let us introduce our central example: the let rule. If the computation of a header is dynamic (resp. high), the whole let expression is classified to be dynamic (resp. high), even though its body could be classified as static (resp. low). The co-domain of the following lambda-abstraction is thus classified as dynamic if e is dynamic:

$$\lambda x.\text{let } v = e \text{ in } b$$

The CPS counterpart of this lambda-abstraction reads as follows:

$$\lambda x.\lambda k.e' \, (\lambda v.b' \, k)$$

where e' and b' are the CPS counterparts of e and b, respectively. Now assume that b naturally yields a static result but is coerced to be dynamic because of the let rule. In the CPS term, e' also yields a dynamic result, i.e., intuitively, v is classified to be dynamic.[1] Intuitively, b' also yields a static result and passes it to its continuation k. Therefore, in direct style, b yields a dynamic result whereas in CPS, it yields a static result.

Two observations need to be made at this point:

(1) The paragraph above is the standard motivation for improving binding times by CPS transformation [3] (see Section 5.2 for further detail). What it leaves unsaid, and actually what has always been left unsaid, however, is whether this local binding-time improvement corresponds to a global improvement as well, or whether it may make things worse elsewhere in the source program. (In Section 4, we prove that this local improvement actually is a global improvement as well.)

(2) In their core calculus of dependency [1], Abadi et al. make a point that any function classified as $d \to s$ (resp. $h \to l$, etc.) is necessarily a constant function. However, as argued above, given a direct-style function classified to be $d \to d$ because of the let rule, its CPS counterpart may very well be classified as $d \to (s \to o) \to o$ and *not* be a constant CPS function (i.e., a function applying its continuation to a constant).

[1]This intuition is formalized in the rest of this article.

$$\begin{array}{rcrcl}
p & \in & Root & ::= & e^\ell \\
e & \in & Exp & ::= & t \mid s \mid \textbf{let } v = s^\ell \textbf{ in } e^{\ell_1} \\
s & \in & Step & ::= & t_0^{\ell_0}\ t_1^{\ell_1} \mid op(t^\ell) \mid \textbf{if0 } t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} \\
t & \in & Triv & ::= & n \mid x \mid v \mid \lambda^{\ell'} x.e^\ell \\
x & \in & Ide & & \text{(identifiers)} \\
v & \in & VIde & & \text{(value identifiers)} \\
n & \in & Int & & \text{(integers)} \\
op & \in & & & \text{an unspecified set of base-type operators}
\end{array}$$

Figure 1: The direct-style language

Together, these two observations tell us that the let rule entails a real slack in BTA, security analysis, etc.—a slack that CPS makes it possible to exploit non-trivially by providing a local improvement which is also a global improvement.

Before moving on to the rest of this article, let us briefly get back to Sabry and Felleisen's observation that any improvement in precision provided by CPS is solely due to continuation duplication [31]. True as this observation may be for data-flow analysis, we have just shown that it does not necessarily hold for other analyses such as BTA.

The rest of this article is organized as follows: Section 2 presents the language of discourse and its CPS transformation; Section 3 addresses control-flow analysis in constraint form; Section 4 addresses binding-time analysis in constraint form; Section 5 reviews related work; and Section 6 concludes.

2. THE LANGUAGE

We consider a direct-style λ-language (Figure 1) and its CPS counterpart (Figure 2). These two languages are connected by CPS transformation (Figure 3). Terms in these languages are labelled for the purpose of program analysis. The direct-style language is a variant of A-normal forms [9], an intermediate language inspired by Moggi's computational monads [12], and where intermediate computations are sequentialized and their result is named.

The CPS-transformation of (unlabeled) direct-style terms is defined as in Figure 3. For the moment, the labels are not relevant. In the following sections we will show how to treat the labels as well.

We want to analyze direct-style and CPS terms with the

$$
\begin{array}{rclcl}
p & \in & \mathit{CRoot} & ::= & \lambda^{\ell'} k.e^{\ell} \\
e & \in & \mathit{CExp} & ::= & c^{\ell_1}\ t^{\ell_2} \mid t_0^{\ell_0}\ t_1^{\ell_1}\ \lambda^{\ell'} v.e^{\ell} \mid \\
 & & & & \tilde{op}\ t_1^{\ell_1}\ \lambda^{\ell'} v.e^{\ell} \mid \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} \mid \\
 & & & & \mathbf{let}\ k = \lambda^{\ell'} v.e^{\ell_2}\ \mathbf{in}\ \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} \\
t & \in & \mathit{CTriv} & ::= & n \mid x \mid \lambda^{\ell'} x.\lambda^{\ell_1} k.e^{\ell} \\
c & \in & \mathit{Cont} & ::= & k \mid \lambda^{\ell'} v.e^{\ell} \\
x & \in & \mathit{Ide} & & \text{(identifiers)} \\
v & \in & \mathit{VIde} & & \text{(value identifiers)} \\
k & \in & \mathit{CIde} & & \text{(continuation identifiers)} \\
\tilde{op} & \in & \multicolumn{3}{l}{\text{an unspecified set of base-type CPS operators}}
\end{array}
$$

Figure 2: The CPS language

$$
\begin{array}{rcl}
\mathcal{C}[\![e]\!] & = & \lambda k.\mathcal{C}[\![e]\!]k \\
 & & \text{ where } k \text{ is fresh} \\
\\
\mathcal{C}[\![t]\!]k & = & k\ \mathcal{C}[\![t]\!] \\
\mathcal{C}[\![t_0\ t_1]\!]k & = & \mathcal{C}[\![t_0]\!]\ \mathcal{C}[\![t_1]\!]\ \lambda v.k\ v \\
 & & \text{ where } v \text{ is fresh} \\
\mathcal{C}[\![op(t)]\!]k & = & \tilde{op}\ \mathcal{C}[\![t]\!]\ \lambda v.k\ v \\
 & & \text{ where } v \text{ is fresh} \\
\mathcal{C}[\![\mathbf{if0}\ t\ e_0\ e_1]\!]k & = & \mathbf{if0}\ \mathcal{C}[\![t]\!]\ \mathcal{C}[\![e_0]\!]k\ \mathcal{C}[\![e_1]\!]k \\
\mathcal{C}[\![\mathbf{let}\ v = t_0\ t_1\ \mathbf{in}\ e]\!]k & = & \mathcal{C}[\![t_0]\!]\ \mathcal{C}[\![t_1]\!]\ \lambda v.\mathcal{C}[\![e]\!]k \\
\mathcal{C}[\![\mathbf{let}\ v = op(t)\ \mathbf{in}\ e]\!]k & = & \tilde{op}\ \mathcal{C}[\![t]\!]\ \lambda v.\mathcal{C}[\![e]\!]k \\
\\
\mathcal{C}\left[\!\!\left[\begin{array}{l}\mathbf{let}\ v = \mathbf{if0}\ t\ e_0\ e_1 \\ \mathbf{in}\ e\end{array}\right]\!\!\right]k & = & \begin{array}{l}\mathbf{let}\ k' = \lambda v.\mathcal{C}[\![e]\!] \\ \mathbf{in}\ \mathbf{if0}\ \mathcal{C}[\![t]\!]\ \mathcal{C}[\![e_0]\!]k'\ \mathcal{C}[\![e_1]\!]k'\end{array} \\
 & & \text{ where } k' \text{ is fresh}
\end{array}
$$

$$
\begin{array}{rclrcl}
\mathcal{C}[\![n]\!] & = & n & \mathcal{C}[\![v]\!] & = & v \\
\mathcal{C}[\![x]\!] & = & x & \mathcal{C}[\![\lambda x.e]\!] & = & \lambda x.\lambda k.\mathcal{C}[\![e]\!]k \\
 & & & & & \text{ where } k \text{ is fresh}
\end{array}
$$

Figure 3: The CPS transformation

same program analysis. To this end, we embed CPS terms into direct-style terms, as specified in Figure 4.

We consider all terms to be labeled with labels ℓ from a countable domain. We distinguish *Lab* as the set of labels on program points and *Lam* as the set of labels of lambda-abstractions:

$$(\lambda^{\ell'} x.e^{\ell_1})^{\ell_0} : \ell_0 \in \mathit{Lab}, \ell' \in \mathit{Lam}$$

Lam and the set of syntactic lambda-abstractions are in bijective correspondence. We can refer to a syntactic lambda-abstraction using its label or vice versa.

3. CONTROL-FLOW ANALYSIS

We consider a control flow analysis over direct-style programs. We choose a constraint-based flow analysis, for simplicity [10, 16, 23, 24, 27]. Constraints have been shown to be equivalent to other methods such as set-based analysis [14] and type inference [28]. For simplicity, we adopt the same definition and notation as in Nielson, Nielson and Hankin's recent textbook on program analysis [25].

A control flow analysis computes two mappings: a cache $\widehat{C}$, mapping expressions to sets of lambda-abstractions that

$$
\begin{array}{rcl}
\Phi[\![\lambda k.e]\!] & = & \lambda k.\Phi[\![e]\!] \\
\\
\Phi[\![c\ t]\!] & = & \Phi[\![c]\!]\ \Phi[\![t]\!] \\
\Phi[\![t_0\ t_1\ \lambda v.e]\!] & = & \mathbf{let}\ v_0 = \Phi[\![t_0]\!]\Phi[\![t_1]\!] \\
 & & \mathbf{in}\ v_0\ \lambda v.\Phi[\![e]\!] \\
\Phi[\![\tilde{op}\ t\ \lambda v.e]\!] & = & \mathbf{let}\ v = op(\Phi[\![t]\!])\ \mathbf{in}\ \Phi[\![e]\!] \\
\Phi\left[\!\!\left[\begin{array}{l}\mathbf{let}\ k = \lambda v.e \\ \mathbf{in}\ \mathbf{if0}\ t\ e_0\ e_1\end{array}\right]\!\!\right] & = & \begin{array}{l}(\lambda k.\mathbf{if0}\ \Phi[\![t]\!]\ \Phi[\![e_0]\!]\ \Phi[\![e_1]\!]) \\ \quad (\lambda v.\Phi[\![e]\!])\end{array} \\
\Phi[\![\mathbf{if0}\ t\ e_0\ e_1]\!] & = & \mathbf{if0}\ \Phi[\![t]\!]\ \Phi[\![e_0]\!]\ \Phi[\![e_1]\!]
\end{array}
$$

$$
\begin{array}{rclrcl}
\Phi[\![n]\!] & = & n & & & \\
\Phi[\![x]\!] & = & x & \Phi[\![k]\!] & = & k \\
\Phi[\![v]\!] & = & v & \Phi[\![\lambda x.\lambda k.e]\!] & = & \lambda x.\lambda k.\Phi[\![e]\!]
\end{array}
$$

Figure 4: Embedding CPS into direct style

Lam_p	The set of lambda terms in p
Var_p	The union of *VIde* and *Ide* terms in p
Lab_p	The set of term labels in p

$$
\begin{array}{rcrcll}
 & & \mathit{Val}_p & = & \mathcal{P}(\mathit{Lam}_p) & \text{Abstract values} \\
\widehat{C} & \in & \mathit{Cache}_p & = & \mathit{Lab}_p \rightarrow \mathit{Val}_p & \text{Cache} \\
\widehat{\rho} & \in & \mathit{Env}_p & = & \mathit{Var}_p \rightarrow \mathit{Val}_p & \text{Environment}
\end{array}
$$

$$\models_p : (\mathit{Cache}_p \times \mathit{Env}_p) \times \mathit{Lab}_p \rightarrow \{\mathit{true}, \mathit{false}\}$$

Figure 5: Functionality of 0-CFA predicate

might be the result of evaluating the expression, and an environment $\widehat{\rho}$, mapping program variables to sets of lambda-abstractions these variables may denote. The constraint-based control-flow analysis is specified as a predicate $\models$ on pairs of caches and environments and program points. The informal meaning of the statement $(\widehat{C}, \widehat{\rho}) \models e$ is that $(\widehat{C}, \widehat{\rho})$ is a safe analysis of the expression e.

In this work we consider a syntax-directed version of the analysis [25], which defines a "satisfaction predicate" restricted to the program being analyzed. Given a closed direct-style program p, the functionality of its associated satisfaction predicate $\models_p$ is defined in Figure 5. The predicate is defined recursively on the syntax of the program as in Figure 6.

For a syntax-directed analysis, the predicate $\models$ defined by the recursive equations in Figure 6 is already proven to be defined uniquely and to provide a safe analysis [25]. Furthermore, Nielson, Nielson and Hankin have shown that the set of acceptable solutions of the analysis of a term e

$$A = \{(\widehat{C}, \widehat{\rho}) | (\widehat{C}, \widehat{\rho}) \models_p e\}$$

has a model intersection property (i.e., it is a Moore family). In other words, the greatest lower bound of any subset of A exists and is an element of A.

This property ensures the existence of a least analysis, i.e., a most precise one. (Here the order relation is given by the pointwise ordering of functions induced by set inclusion.) This puts us in an ideal position to compare precisions. Furthermore, it is also known that algorithmically,

$$
\begin{array}{lcl}
(\widehat{C},\widehat{\rho}) \models_p n^\ell & \iff & \mathit{true} \\
(\widehat{C},\widehat{\rho}) \models_p x^\ell & \iff & \widehat{\rho}(x) \sqsubseteq \widehat{C}(\ell) \\
(\widehat{C},\widehat{\rho}) \models_p v^\ell & \iff & \widehat{\rho}(v) \sqsubseteq \widehat{C}(\ell) \\
(\widehat{C},\widehat{\rho}) \models_p (\lambda^{\ell'} x.e)^\ell & \iff & \{\ell'\} \sqsubseteq \widehat{C}(\ell) \wedge (\widehat{C},\widehat{\rho}) \models_p e \\
(\widehat{C},\widehat{\rho}) \models_p (t_0^{\ell_0}\ t_1^{\ell_1})^\ell & \iff & (\widehat{C},\widehat{\rho}) \models_p t_0^{\ell_0} \wedge (\widehat{C},\widehat{\rho}) \models_p t_0^{\ell_1} \wedge \\
 & & \forall(\lambda^{\ell'} x.e^{\ell_2}) \in \widehat{C}(\ell_0).\ \widehat{C}(\ell_1) \sqsubseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_2) \sqsubseteq \widehat{C}(\ell) \\
(\widehat{C},\widehat{\rho}) \models_p (op(t^\ell))^{\ell_0} & \iff & (\widehat{C},\widehat{\rho}) \models_p t^\ell \\
(\widehat{C},\widehat{\rho}) \models_p (\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \iff & (\widehat{C},\widehat{\rho}) \models_p t^\ell \wedge (\widehat{C},\widehat{\rho}) \models_p e_0^{\ell_0} \wedge (\widehat{C},\widehat{\rho}) \models_p e_1^{\ell_1} \wedge \widehat{C}(\ell_0) \sqsubseteq \widehat{C}(\ell_2) \wedge \widehat{C}(\ell_1) \sqsubseteq \widehat{C}(\ell_2) \\
(\widehat{C},\widehat{\rho}) \models_p (\mathbf{let}\ v = s^\ell\ \mathbf{in}\ e^{\ell_1})^{\ell_2} & \iff & (\widehat{C},\widehat{\rho}) \models_p s^\ell \wedge (\widehat{C},\widehat{\rho}) \models_p e^{\ell_1} \wedge \widehat{C}(\ell) \sqsubseteq \widehat{\rho}(v) \wedge \widehat{C}(\ell_1) \sqsubseteq \widehat{C}(\ell_2)
\end{array}
$$

Figure 6: Syntax-directed 0-CFA

the computed solution is the least one.

The rest of this section is organized as follows. First, we show how to CPS-transform the environment/cache flow information (Sections 3.1 and 3.2). We consider that given a program p and a solution for its associated analysis $(\widehat{C},\widehat{\rho}) \models_p p$, we can construct a solution $(\widehat{C}',\widehat{\rho}') \models_{p'} p'$ for the analysis associated to the CPS counterpart p' of p. The transformation is such that "no information is lost" (Section 3.3). Furthermore, we present a converse transformation that also suffers from no loss of information (Section 3.4). Section 3.5 concludes.

Using the model intersection property and the monotonicity of our transformations, we show that the least solution of the analysis for p corresponds to the least solution of the same analysis on p' and vice versa. This leads us to conclude that the CPS transformation has no influence on the flow information obtained by 0-CFA.

3.1 Labeled CPS transformation and embedding

The syntactic transformation is obtained from composing the CPS transformation of Figure 3 with the embedding of Figure 4.

We design the transformation to maintain most of the labels and variable names. In particular, we have:

$$
\begin{array}{rcl}
Var_p & \sqsubseteq & Var_{p'} \\
Lam_p & \sqsubseteq & Lam_{p'} \\
Val_p & \sqsubseteq & Val_{p'}
\end{array}
$$

The following sections will use only the syntactic transformation. We delay until then the details about which labels and variables are preserved, and what are the newly created variables and labels.

3.2 Transformation of flow

The syntactic transformation yields new variables and new labels for the new program points. Given a solution of the analysis for the program p, let us construct a solution for the analysis of the program p' obtained by CPS transforming p and embedding the result into direct style.

The preservation of precision of the analysis is enforced in our construction by maintaining the property that, for a given variable $x \in Var_p$, $\widehat{\rho}'(x) = \widehat{\rho}(x)$. That is, a variable evaluates to the same set of lambda-abstractions in CPS as in direct style. In fact, in our construction, for any trivial term t^ℓ we have that $\widehat{C}'(\ell) = \widehat{C}(\ell)$.

The environment transformation uses a function $Kl(\cdot)$ which extracts the labels of partially applied CPS lambda-abstractions. Formally, it is defined as follows: considering A to be the set of CPS lambda-abstractions $\{\lambda^{\ell_i} x_i.\lambda^{\ell_{k_i}} k_i.e_i | 1 \leq i \leq n\}$, for some n, then

$$Kl(A) = \{\ell_{k_i} | 1 \leq i \leq n\}.$$

The CPS transformation of the cache/environment is almost syntax-directed as well. The only thing that is necessary to inductively construct the CPS solution $(\widehat{C}',\widehat{\rho}')$ is the information related to the bindings of continuation variables. Such information can be obtained from the initial flow information. Looking at the places where functions are applied, we can *syntactically* detect what are the continuations that are being passed at the application points. This is a byproduct of introducing a syntactic continuation for each application point.

We parameterize the transformation of the cache/environment pair by a function $\chi : CIde_{p'} \rightarrow Val_{p'}$ mapping continuation identifiers to sets of lambda-abstractions. The idea is that the environment transformation is done in two steps: first, the function χ is computed (as detailed just below), and next, the transformation is performed following Figure 7.

The function χ is defined in fact using the syntactic transformation of the program p as follows (we omit labels where they are not relevant):

- If k is the top-level continuation, introduced at the transformation of the root term:

  $$[\![e]\!]^{Root} = \lambda k.[\![e]\!]^{Exp} k$$

  then $\chi(k) = \emptyset$. That is, we consider that no top-level continuation is provided.

- If k is introduced by the transformation of a named

$$
\begin{array}{rcll}
[\![n^\ell]\!]^{Triv} & = & n^\ell & \widehat{C}'(\ell)=\emptyset \\
[\![x^\ell]\!]^{Triv} & = & x^\ell & \widehat{C}'(\ell)=\widehat{C}(\ell) \\
[\![v^\ell]\!]^{Triv} & = & v^\ell & \widehat{C}'(\ell)=\widehat{C}(\ell) \\
[\![(\lambda^{\ell'} x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^{\ell'} x.(\lambda^{\ell''} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell & \widehat{C}'(\ell)=\widehat{C}(\ell) \quad \widehat{C}'(\ell_2)=Kl(\widehat{C}(\ell)) \\
 & & & \widehat{\rho}'(x)=\widehat{\rho}(x) \quad \widehat{\rho}'(k)=\chi(k) \\
[\![e^\ell]\!]^{Root} & = & (\lambda^{\ell_0} k.[\![e^\ell]\!]^{Exp} k)^{\ell_1} & \widehat{C}'(\ell_1)=\{\ell_0\} \quad \widehat{\rho}'(k)=\emptyset \\
[\![t^\ell]\!]^{Exp} k & = & (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0} & \widehat{C}'(\ell_1)=\widehat{\rho}'(k) \quad \widehat{C}'(\ell_0)=\emptyset \\
[\![(t_0^{\ell_0}\ t_1^{\ell_1})^\ell]\!]^{Exp} k & = & (\mathbf{let}\ v_0=([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{\rho}'(v)=\widehat{C}(\ell) \quad \widehat{C}'(\ell_5)=\{\ell'\} \\
 & & \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'} v.(k\ v)^{\ell_4})^{\ell_5})^{\ell_6})^{\ell_7} & \widehat{C}'(\ell_4)=\widehat{C}'(\ell_6)=\widehat{C}'(\ell_7)=\emptyset \\
 & & & \widehat{\rho}'(v_0)=\widehat{C}'(\ell_3)=\widehat{C}'(\ell_2)=Kl(\widehat{C}(\ell_0)) \\
[\![(op(t_0^{\ell_0}))^\ell]\!]^{Exp} k & = & (\mathbf{let}\ v=(op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell & \widehat{C}'(\ell)=\widehat{C}(\ell) \quad \widehat{\rho}'(v)=\widehat{C}'(\ell) \\
 & & \mathbf{in}\ (k\ v)^{\ell_1})^{\ell_2} & \widehat{C}'(\ell_1)=\widehat{C}'(\ell_2)=\emptyset \\
[\![(\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}]\!]^{Exp} k & = & (\mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp} k'\ [\![e_1^{\ell_1}]\!]^{Exp} k')^{\ell_3} & \widehat{C}'(\ell_3)=\emptyset \\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(t_0^{\ell_0}\ t_1^{\ell_1})^\ell \\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}(\mathbf{let}\ v_0=([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} \\ \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'} v.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_5})^{\ell_6})^{\ell_7}\end{array} & \begin{array}{l}\widehat{\rho}'(v)=\widehat{C}(\ell) \quad \widehat{C}'(\ell_5)=\{\ell'\} \\ \widehat{C}'(\ell_6)=\widehat{C}'(\ell_7)=\emptyset \\ \widehat{\rho}'(v_0)=\widehat{C}'(\ell_3)=\widehat{C}'(\ell_2)=Kl(\widehat{C}(\ell_0))\end{array} \\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(op(t_0^{\ell_0}))^\ell \\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}(\mathbf{let}\ v=(op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell \\ \mathbf{in}\ [\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4}\end{array} & \begin{array}{l}\widehat{\rho}'(v)=\widehat{\rho}(v) \quad \widehat{C}'(\ell)=\widehat{C}(\ell) \\ \widehat{C}'(\ell_4)=\emptyset\end{array} \\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \\ \mathbf{in}\ e^{\ell_3})^{\ell_4}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}((\lambda^{\ell'} k'.(\mathbf{if0}\ [\![t^\ell]\!]^{Triv} \\ \qquad [\![e_0^{\ell_0}]\!]^{Exp} k' \\ \qquad [\![e_1^{\ell_1}]\!]^{Exp} k')^{\ell_5})^{\ell_6} \\ \quad (\lambda^{\ell''} v.[\![e^{\ell_3}]\!]^{Exp} k)^{\ell_7})^{\ell_8}\end{array} & \begin{array}{l}\widehat{\rho}'(v)=\widehat{C}(\ell_2) \\ \widehat{C}'(\ell_6)=\{\ell'\} \quad \widehat{C}'(\ell_7)=\{\ell''\} \\ \widehat{C}'(\ell_5)=\widehat{C}'(\ell_8)=\emptyset\end{array}
\end{array}
$$

Figure 7: Flow transformation from direct style to CPS

if0 :

$$
\left[\!\!\left[\begin{array}{l}\mathbf{let}\ v= \\ \quad \mathbf{if0}\ t\ e_0\ e_1 \\ \mathbf{in}\ e\end{array}\right]\!\!\right]^{Exp} k' = \begin{array}{l}(\lambda k.\mathbf{if0}\ [\![t]\!]^{Triv} \\ \qquad [\![e_0]\!]^{Exp} k \\ \qquad [\![e_1]\!]^{Exp} k) \\ (\lambda^{\ell''} v.[\![e]\!]^{Exp} k')\end{array}
$$

then $\chi(k)=\{\ell''\}$.

- If k is the continuation of a function, i.e., is introduced by lambda-abstraction:

$$[\![\lambda^{\ell'} x.e]\!]^{Triv} = \lambda x.\lambda k.[\![e]\!]^{Exp} k$$

then the value of $\chi(k)$ is defined as follows:

  – for each transformation:

$$[\![t_0^{\ell_0}\ t_1]\!]^{Exp} k = \begin{array}{l}\mathbf{let}\ v_0=[\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \mathbf{in}\ (v_0\ \lambda^{\ell''} v.k\ v)\end{array}$$

  such that $\ell' \in \widehat{C}(\ell_0)$, then $\ell'' \in \chi(k)$.

  – for each transformation:

$$\left[\!\!\left[\begin{array}{l}\mathbf{let}\ v=t_0^{\ell_0}\ t_1 \\ \mathbf{in}\ e\end{array}\right]\!\!\right]^{Exp} k = \begin{array}{l}\mathbf{let}\ v_0=[\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \mathbf{in}\ v_0\ \lambda^{\ell''} v.[\![e]\!]^{Exp} k\end{array}$$

  such that $\ell' \in \widehat{C}(\ell_0)$, then $\ell'' \in \chi(k)$.

Intuitively, the mapping χ will gather all return syntactic continuations for each function, using the direct-style flow information.

One can thus clearly see that the function χ can be constructed only from $\widehat{C}$, $\widehat{\rho}$ and p. Then, we can show that we can define $\widehat{C}'$ and $\widehat{\rho}'$ by induction on the transformation of p into p', following Figure 7. Thus, $\widehat{C}'$ and $\widehat{\rho}'$ are well defined.

In fact, we can consider this construction of flow information as a mapping:

$$\mathcal{F}_p : (Cache_p \times Env_p) \rightarrow (Cache_{p'} \times Env_{p'})$$

and we can relatively easily show that $\mathcal{F}_p$ is injective, monotone and distributive [6].

3.3 Correctness of transformation

Take a program p and its associated satisfaction predicate $\models_p$ constructed as in Section 3.2. Given $(\widehat{C},\widehat{\rho}) \models_p p$ a solution for the analysis of p, we consider the CPS counterpart of p as p' and $\models_{p'}$ its associated predicate. Let $(\widehat{C}',\widehat{\rho}')$ be constructed as in Section 3.2. The following theorem establishes the correctness of our transformation.

Theorem 1. $(\widehat{C}',\widehat{\rho}') \models_{p'} p'$

Definition 1. Given a labeled expression $e^\ell \in Exp$, we define the set $\mathcal{R}(e)$ of return labels of e by induction on the syntax as follows:

- $\mathcal{R}(t^\ell) = \{l\}$
- $\mathcal{R}((t_0^{\ell_1}\ t_1^{\ell_1})^\ell) = \{l\}$
- $\mathcal{R}((op(t^{\ell_1}))^\ell) = \{l\}$
- $\mathcal{R}((\mathbf{if0}\ t^{\ell_0}\ e_1^{\ell_1}\ e_2^{\ell_2})^\ell) = \mathcal{R}(e_1^{\ell_1}) \cup \mathcal{R}(e_1^{\ell_1})$
- $\mathcal{R}((\mathbf{let}\ v = s^{\ell_1}\ \mathbf{in}\ e^{\ell_2})^\ell) = \mathcal{R}(e^{\ell_2})$

LEMMA 1. *Given $(\widehat{C}, \widehat{\rho}) \vDash_p p$ and $e^{\ell'}$ a subexpression of e we have:*

$$\forall \ell \in \mathcal{R}(e).\widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell')$$

PROOF. By induction, using the definition of $\mathcal{R}(e)$ and the constraints generated by the hypothesis. □

LEMMA 2. *For all subterms t^ℓ in p, we have $\widehat{C}'(\ell) = \widehat{C}(\ell)$. For all variables x, v in p, we have $\widehat{\rho}'(x) = \widehat{\rho}(x)$ and $\widehat{\rho}'(v) = \widehat{\rho}(v)$.*

PROOF. Immediate. □

PROOF OF THEOREM 1. The proof proceeds by induction on the transformation of p into p'. We only present two sample cases.

In order to prove $(\widehat{C}', \widehat{\rho}') \vDash_{p'} (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0}$, we need to show that $(\widehat{C}', \widehat{\rho}') \vDash_{p'} k^{\ell_1}$, which follows by construction. We also need to show that $(\widehat{C}', \widehat{\rho}') \vDash_{p'} [\![t^\ell]\!]^{Triv}$, which follows from the induction hypothesis.

Taking an arbitrary $\lambda^{\ell_k} v.e^{\ell'}$ in $\widehat{C}'(\ell_1) = \widehat{\rho}'(k)$, we need to show that:

- $\widehat{C}'(\ell') \sqsubseteq \widehat{C}'(\ell_0)$: by definition amounts to $\emptyset \sqsubseteq \emptyset$
- $\widehat{C}'(\ell) \sqsubseteq \widehat{\rho}'(v)$: We can do a case-analysis on the continuation k that comes from the transformation. If k comes from the top-level, then $\widehat{\rho}'(k) = \emptyset$. If k comes from the transformation of an **if0** then, using notations from Figure 7, $\widehat{\rho}'(k) = \{\ell''\}$. Then, $\widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell_3)$, by Lemma 1.

  Otherwise, k comes from the transformation of a lambda-abstraction from p:

  $$[\![\lambda^{\ell_3} x.e^{\ell_4}]\!]^{Triv} = \lambda^{\ell_3} x.\lambda^{\ell''} k.[\![e^{\ell_4}]\!]^{Exp} k$$

  such that $\ell \in \mathcal{R}(e^{\ell_4})$.

  The fact that $\ell_k \in \widehat{\rho}'(k) = \chi(k)$ leads again to two cases. In one case, we must have a transformation step as:

  $$\begin{aligned}[\![(t_0^{\ell_7}\ t_1)^{\ell_8}]\!]^{Exp} k = \mathbf{let}\ v_0 = [\![t_0^{\ell_7}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \mathbf{in}\ v_0\ \lambda^{\ell_k} v.e^{\ell'}\end{aligned}$$

  such that $\ell_3 \in \widehat{C}(\ell_7)$, which means that the lambda-abstraction $\lambda^{\ell_3} x.e^{\ell_4}$ is called at the application point ℓ_8 in p, which implies:

  $$\widehat{C}(\ell_4) \sqsubseteq \widehat{C}(\ell_8)$$

  By definition we also have that $\widehat{\rho}'(v) = \widehat{C}(\ell_8)$. On the other hand, since $\ell \in \mathcal{R}(\ell_4)$, using Lemma 1, yields:

  $$\widehat{C}'(\ell) = \widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell_4) \sqsubseteq \widehat{C}(\ell_8) = \widehat{\rho}'(v)$$

  The other case follows through a similar reasoning.

What about the transformation of an application $(t_0^{\ell_0}\ t_1^{\ell_1})^\ell$? We need to show that

$$\begin{aligned}(\widehat{C}', \widehat{\rho}') \vDash_p (\mathbf{let}\ v_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} \\ \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'} v.(k\ v)^{\ell_4})^{\ell_5})^{\ell_6})^{\ell_7}\end{aligned}$$

While much of the constraints hold trivially from induction hypothesis or from the hypothesis on $(\widehat{C}, \widehat{\rho})$, it is interesting again to show that $(\widehat{C}', \widehat{\rho}') \vDash_{p'} (k\ v)^{\ell_4}$.

In particular we need to show that, whichever $\lambda^{\ell_i} v_1 \ldots$ the continuation k might evaluate to, $\widehat{\rho}'(v) \sqsubseteq \widehat{\rho}'(v_1)$. We perform again a case distinction on the origin of k. If k comes from the transformation of a lambda-abstraction $\lambda x.e_1^{\ell_e}$ into $\lambda x.\lambda k.[\![e]\!]^{Exp} k$ then obviously $\ell \in \mathcal{R}(e)$. Applying again Lemma 1, $\widehat{C}(\ell) = \widehat{\rho}'(v) \sqsubseteq \widehat{C}(\ell_e)$. Exploring the cases where ℓ_k comes from, we eventually establish that $\widehat{C}(\ell_e) \sqsubseteq \widehat{\rho}'(v_1)$.

The other cases are proven using similar arguments. □

3.4 The inverse flow transformation and its correctness

Up until now we have shown that we can transform direct-style flow information into CPS flow information. We have informally explained that the transformation preserves the precision. Thus analyzing a CPS term yields results at least as precise as analyzing its original direct style version.

We can also show that any result of the analysis on CPS can be matched by a result on the direct-style version.

Using again the structure given by the syntactic transformation, we exhibit a direct-style flow transformation. Given the direct-style program p and its CPS counterpart p', and given $(\widehat{C}', \widehat{\rho}') \vDash_{p'} p'$ a valid result of the analysis on p', we can define a valid solution $(\widehat{C}, \widehat{\rho}) \vDash_p p$.

The only thing we need to care about is "filtering" the continuation functions from the results of the analysis. The definition is given in Figure 8, where we use $A =_p A'$ as a shortcut for $A = A' \cap \mathcal{P}(Lam_p)$

In fact, it can be shown that Figure 8 defines a function

$$\mathcal{F}_p^b : (Cache_{p'} \times Env_{p'}) \rightarrow (Cache_p \times Env_p)$$

We can relatively easily show that $\mathcal{F}_p^b$ is monotone as well.

We only need to show that the backward transformation indeed defines a valid solution of the analysis:

$$
\begin{array}{rcll}
[\![n^\ell]\!]^{Triv} & = & n^\ell & \widehat{C}(\ell) = \emptyset \\
[\![x^\ell]\!]^{Triv} & = & x^\ell & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
[\![v^\ell]\!]^{Triv} & = & v^\ell & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
[\![(\lambda^{\ell'} x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^{\ell'} x.(\lambda^{\ell''} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \quad \widehat{\rho}(x) =_p \widehat{\rho}'(x) \\
\\
[\![e^\ell]\!]^{Root} & = & (\lambda^{\ell_0} k.[\![e^\ell]\!]^{Exp} k)^{\ell_1} & \\
[\![t^\ell]\!]^{Exp} k & = & (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0} & \\
[\![(t_0^{\ell_0}\ t_1^{\ell_1})^\ell]\!]^{Exp} k & = & (\mathbf{let}\ v_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{C}(\ell) =_p \widehat{\rho}'(v) \\
 & & \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'} v.(k\ v)^{\ell_4})^{\ell_5})^{\ell_6})^{\ell_7} & \\
[\![(op(t_0^{\ell_0}))^\ell]\!]^{Exp} k & = & (\mathbf{let}\ v = (op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
 & & \mathbf{in}\ (k\ v)^{\ell_1})^{\ell_2} & \\
[\![(\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}]\!]^{Exp} k & = & (\mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp} k\ [\![e_1^{\ell_1}]\!]^{Exp} k)^{\ell_3} & \widehat{C}(\ell_2) = \widehat{C}(\ell_0) \sqcup \widehat{C}(\ell_1) \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v = (t_0^{\ell_0}\ t_1^{\ell_1})^\ell \\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}(\mathbf{let}\ v_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} \\ \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'} v.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_5})^{\ell_6})^{\ell_7}\end{array} & \begin{array}{l}\widehat{\rho}(v) = \widehat{C}(\ell) =_p \widehat{\rho}'(v) \\ \widehat{C}(\ell_3) = \widehat{C}(\ell_2)\end{array} \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v = (op(t_0^{\ell_0}))^\ell \\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}(\mathbf{let}\ v = (op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell \\ \mathbf{in}\ [\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4}\end{array} & \begin{array}{l}\widehat{\rho}(v) =_p \widehat{\rho}'(v) \quad \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\ \widehat{C}(\ell_3) = \widehat{C}(\ell_2)\end{array} \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v = (\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \\ \mathbf{in}\ e^{\ell_3})^{\ell_4}\end{array}\right]\!\!\right]^{Exp} k & = & \begin{array}{l}((\lambda^{\ell'} k'.(\mathbf{if0}\ [\![t^\ell]\!]^{Triv} \\ \qquad\qquad [\![e_0^{\ell_0}]\!]^{Exp} k' \\ \qquad\qquad [\![e_1^{\ell_1}]\!]^{Exp} k')^{\ell_5})^{\ell_6} \\ \quad (\lambda^{\ell''} v.[\![e^{\ell_3}]\!]^{Exp} k)^{\ell_7})^{\ell_8}\end{array} & \begin{array}{l}\widehat{C}(\ell_2) =_p \widehat{\rho}'(v) \\ \widehat{\rho}(v) = \widehat{C}(\ell_2) \\ \widehat{C}(\ell_4) = \widehat{C}(\ell_3)\end{array}
\end{array}
$$

Figure 8: Flow transformation from CPS to direct style

THEOREM 2. $(\widehat{C}, \widehat{\rho}) \models_p p$

Taking the same steps as in the proof of Section 3.2, we establish the following lemma.

LEMMA 3. *Assuming the construction of Figure 8, let e^ℓ be an expression in p. Then*

$$\widehat{C}(\ell) = \bigsqcup_{\ell' \in \mathcal{R}(e)} \widehat{C}(\ell')$$

Similarly, an equivalent of Lemma 2 holds for this transformation as well.

PROOF OF THEOREM 2 (SKETCH). Again, the proof proceeds by induction on the transformation of p into p'. An interesting case is to prove that the constraints for an application $(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell_2}$ are satisfied. More specifically, to show that for an arbitrary $\lambda^{\ell'} x.e^\ell \in \widehat{C}(\ell_0)$ we have

- $\widehat{C}'(\ell_1) \sqsubseteq \widehat{\rho}'(x) \iff \widehat{C}(\ell_1) \sqsubseteq \widehat{\rho}(x)$: follows from $(\widehat{C}', \widehat{\rho}') \models_{p'} ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2}$
- $\widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell_2)$: Using Lemma 3, all we need to show is that for an arbitrary $\ell' \in \mathcal{R}(e^\ell)$, $\widehat{C}(\ell') \sqsubseteq \widehat{\rho}'(v)$, where v is the continuation variable in the CPS transformation of $(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell_2}$. But this is true: the continuation is "applied" at all return points. This can be shown by induction on e^ℓ.

The other cases are similar. □

After defining the two flow transformations, we can almost immediately show that given $(\widehat{C}, \widehat{\rho}) \models_p p$, then $\mathcal{F}_p^b(\mathcal{F}_p(\widehat{C}, \widehat{\rho})) \sqsubseteq (\widehat{C}, \widehat{\rho})$. Although not that immediately, we can also show that given an arbitrary solution $(\widehat{C}', \widehat{\rho}') \models_p p'$, $\mathcal{F}_p(\mathcal{F}_p^b(\widehat{C}', \widehat{\rho}')) \sqsubseteq (\widehat{C}', \widehat{\rho}')$, which is characteristic of a Galois connection. From these two properties it is a simple exercise to prove the following main theorem.

THEOREM 3 (EQUIVALENCE OF FLOW). *Given program p, let $(\widehat{C}, \widehat{\rho}) \models_p p$ be the least solution of the CFA on p. Let $(\widehat{C}', \widehat{\rho}') \models_{p'} p'$ be the least solution of the CFA on p'. Then we have:*

- $\mathcal{F}_p(\widehat{C}, \widehat{\rho}) = (\widehat{C}', \widehat{\rho}')$
- $\mathcal{F}_p^b(\widehat{C}', \widehat{\rho}') = (\widehat{C}, \widehat{\rho})$

3.5 Summary and conclusion

Theorem 3 shows that the least flow information obtained by a constraint-based analysis on a direct-style program can be transformed into the least flow information obtainable by the same analysis on the CPS counterpart of this program and vice versa. From Lemma 2 we thus conjecture that, in this setting, the same amount of flow-based optimizations are possible in direct style as in CPS.

Therefore, for constraint-based control-flow analysis, no loss and no gain of information are incurred by the CPS transformation.

$$\begin{array}{rcrcll} & & Val^b & = & \{\mathbf{S},\mathbf{D}\} & \text{Abstract values} \\ \widehat{C}_b & \in & Cache^b_p & = & Lab_p \rightarrow Val^b & \text{Cache} \\ \widehat{\rho}_b & \in & Env^b_p & = & Var_p \rightarrow Val^b & \text{Environment} \end{array}$$

$$\models^b_p : (Cache^b_p \times Env^b_p) \times Lab^\star \rightarrow \{true, false\}$$

Figure 9: Functionality of BTA predicate

We will use this result in the next section to show that the CPS transformation does not incur losses for a BTA based on flow information. Unlike for 0-CFA, though, we will show that the syntactic CPS transformation may lead to more specialization opportunities.

4. BINDING-TIME ANALYSIS

With a similar development, we now consider a binding-time analysis based on constraints [13, 26, 27, 29]. The definition of the analysis is similar to the development of the 0-CFA in Section 3. The satisfaction predicate is defined on similar domains (Figure 9): only the domain of abstract values changes into the standard lattice $\{\mathbf{S},\mathbf{D}\}$ of static and dynamic annotations.

The analysis is specified in Figure 10. It makes use of the flow information $(\widehat{C},\widehat{\rho})$, which is computed by the control-flow analysis of Section 3. Alternatively, we could consider an analysis computing both flow and binding-time information, but such an analysis is known to be equivalent [27]. Based on our previous result, we thus now know that the flow analysis on direct style and on CPS will be the same.

In contrast to the 0-CFA, a BTA accepts non-closed terms. In this case, following the tradition, we consider the free variables to be dynamic. Their associated flow information is considered to be empty.

Compared the 0-CFA analysis, the BTA constraints are equality constraints. Moreover, we have additional constraints for lambda-abstraction, branching and naming of intermediate results. In each of these cases, classifying a component of the construct as dynamic can force other components to be dynamic as well.

The result of the analysis is used to make standard binding-time annotations [18, 22, 26]. Informally, the annotations are made as follows:

$$\begin{array}{rcll} (\lambda^{\ell'} x.e^{\ell_0})^\ell & \rightarrow & (\underline{\lambda}^{\ell'} x.e^{\ell_0})^\ell & \text{if } \widehat{C}(\ell)=\mathbf{D} \\ (t_0^{\ell_0}\ t_1^{\ell_1})^\ell & \rightarrow & (t_0^{\ell_0}\underline{@}t_1^{\ell_1})^\ell & \text{if } \widehat{C}(\ell_0)=\mathbf{D} \\ (op(t_0^{\ell_0}))^\ell & \rightarrow & (\underline{op}(t_0^{\ell_0}))^\ell & \text{if } \widehat{C}(\ell_0)=\mathbf{D} \\ (\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \rightarrow & (\underline{\mathbf{if0}}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \text{if } \widehat{C}(\ell)=\mathbf{D} \\ (\mathbf{let}\ v=s^\ell & \rightarrow & (\underline{\mathbf{let}}\ v=s^\ell & \text{if } \widehat{C}(\ell)=\mathbf{D} \\ \mathbf{in}\ e^{\ell_2})^{\ell_3} & & \underline{\mathbf{in}}\ e^{\ell_2})^{\ell_3} & \end{array}$$

4.1 Relating the binding times

As in Section 3, we provide a similar construction to show that the binding times obtained by the analysis on a CPS term are at least as good as the ones obtained by the analysis on the corresponding direct-style term. The transformation of binding times is defined inductively on the translation to CPS (see Figure 11).

This time, the only connection between the binding times on direct style and CPS is to show that given any binding-time solution for direct style, we can exhibit a safe set of binding times for CPS such that all trivial terms have the same binding times. In particular, this implies that no lifting of values is required. It is also trivial to see that the static applications or static tests in the direct-style program remain static as well in the CPS program.

We thus prove the following two lemmas.

LEMMA 4. *For all subterms t^ℓ in p, $\widehat{C}'_b(\ell)=\widehat{C}_b(\ell)$. For all variables x, v in p, $\widehat{\rho}'_b(x)=\widehat{\rho}_b(x)$ and $\widehat{\rho}'_b(v)=\widehat{\rho}_b(v)$.*

PROOF. Immediate. □

THEOREM 4. $(\widehat{C}'_b,\widehat{\rho}'_b)\models^b_p p'$

The equivalent of Lemma 1 for BTA is as follows.

LEMMA 5. *Given $(\widehat{C},\widehat{\rho})\models_p p$ and $e^{\ell'}$ a subexpression of e,*

$$\forall \ell \in \mathcal{R}(e).\widehat{C}(\ell)=\widehat{C}(\ell')$$

PROOF. Inductive, using the definition of $\mathcal{R}(e)$ and the constraints generated by the hypothesis. □

PROOF OF THEOREM 4. Similar to the previous proofs. We proceed by induction on the transformation of p into p'.

To check that $(\widehat{C}'_b,\widehat{\rho}'_b)\models^b_{p'} (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0}$, among others, we need to show that for an arbitrary $\lambda^{\ell_k}v.e^{\ell'}\in\widehat{C}'(\ell_1)=\widehat{\rho}'(k)$ the following holds:

- $\widehat{C}'_b(\ell')=\widehat{C}'_b(\ell_0)$: by construction, equivalent to $\mathbf{D}=\mathbf{D}$
- $\widehat{C}'_b(\ell)=\widehat{\rho}'_b(v)$: Again, we do a case-analysis on the continuation k that comes from the transformation. If k comes from the top-level, then $\widehat{\rho}'_b(k)=\mathbf{D}$ and $\widehat{C}'_b(\ell)=\mathbf{D}$ from Lemma 5 (is a return for the top-level). If k comes from the transformation of an **if0** then $\widehat{\rho}'_b(k)=\ell''$, as in Figure 11. Then, $\widehat{C}_b(\ell_0)=\widehat{C}_b(\ell_2)=\widehat{C}_b(\ell_1)=\widehat{\rho}'_b(v)$, using Lemma 5.

  Otherwise, k comes from the translation of a lambda-abstraction from p:

  $$[\![\lambda^{\ell_3}x.e^{\ell_4}]\!]^{Triv}=\lambda^{\ell_3}x.\lambda^{\ell''}k.[\![e^{\ell_4}]\!]^{Exp}k$$

  such that $\ell\in\mathcal{R}(e^{\ell_4})$.

  The fact that $\ell_k\in\widehat{\rho}'(k)=\chi(k)$ leads again to two cases. In one case, we must have a transformation step as:

  $$[\![(t_0^{\ell_7}\ t_1)^{\ell_8}]\!]^{Exp}k=\mathbf{let}\ v_0=[\![t_0^{\ell_7}]\!]^{Triv}\ [\![t_1]\!]^{Triv}\ \mathbf{in}\ v_0\ \lambda^{\ell_k}v.e^{\ell'}$$

$$
\begin{array}{lcl}
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p n^\ell & \iff & \mathbf{S}\sqsubseteq \widehat{C}_b(\ell)\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p x^\ell & \iff & \widehat{\rho}_b(x)\sqsubseteq\widehat{C}_b(\ell)\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p x^\ell & \iff & \widehat{\rho}_b(x)=\mathbf{D} \text{ if } x \text{ is free in } p\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p v^\ell & \iff & \widehat{\rho}_b(v)\sqsubseteq \widehat{C}_b(\ell)\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p (\lambda^{\ell'}x.e^{\ell_1})^\ell & \iff & (\widehat{\rho}_b(\ell)=\mathbf{D}\Rightarrow \widehat{C}_b(\ell_1)=\widehat{\rho}_b(x)=\mathbf{D})\wedge(\widehat{C}_b,\widehat{\rho}_b)\models^b_p e\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p (t_0^{\ell_0}\ t_1^{\ell_1})^\ell & \iff & (\widehat{C}_b,\widehat{\rho}_b)\models^b_p t_0^{\ell_0}\wedge (\widehat{C}_b,\widehat{\rho}_b)\models^b_p t_0^{\ell_1}\wedge\\
&& \forall(\lambda^{\ell'}x.e^{\ell_2})\in\widehat{C}(\ell_0).\ \widehat{C}_b(\ell_1)=\widehat{\rho}_b(x)\wedge\widehat{C}_b(\ell_2)=\widehat{C}_b(\ell)\wedge\\
&& \widehat{C}_b(\ell_0)=\mathbf{D}\Rightarrow\widehat{C}_b(\ell_1)=\mathbf{D}\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p (op(t^\ell))^{\ell_0} & \iff & (\widehat{C}_b,\widehat{\rho}_b)\models^b_p t^\ell \wedge \widehat{C}_b(\ell)\sqsubseteq\widehat{C}_b(\ell_0)\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p (\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \iff & (\widehat{C}_b,\widehat{\rho}_b)\models^b_p t^\ell\wedge(\widehat{C}_b,\widehat{\rho}_b)\models^b_p e_0^{\ell_0}\wedge(\widehat{C}_b,\widehat{\rho}_b)\models^b_p e_1^{\ell_1}\wedge \widehat{C}_b(\ell_0)=\widehat{C}_b(\ell_1)=\widehat{C}_b(\ell_2)\wedge\\
&& \widehat{C}_b(\ell)=\mathbf{D}\Rightarrow \widehat{C}_b(\ell_0)=\widehat{C}_b(\ell_1)=\mathbf{D}\\
(\widehat{C}_b,\widehat{\rho}_b)\models^b_p (\mathbf{let}\ v=s^\ell\ \mathbf{in}\ e^{\ell_1})^{\ell_2} & \iff & (\widehat{C}_b,\widehat{\rho}_b)\models^b_p s^\ell\wedge (\widehat{C}_b,\widehat{\rho}_b)\models^b_p e^{\ell_1}\wedge\widehat{C}_b(\ell)=\widehat{\rho}_b(v)\wedge\widehat{C}_b(\ell_1)=\widehat{C}_b(\ell_2)\wedge\\
&& \widehat{C}_b(\ell)=\mathbf{D}\Rightarrow\widehat{C}_b(\ell_1)=\mathbf{D}
\end{array}
$$

Figure 10: Syntax-directed BTA

$$
\begin{array}{rcll}
[\![n^\ell]\!]^{Triv} & = & n^\ell & \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\\
[\![x^\ell]\!]^{Triv} & = & x^\ell & \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\\
[\![v^\ell]\!]^{Triv} & = & v^\ell & \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\\
[\![(\lambda^{\ell'}x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^{\ell'}x.(\lambda^{\ell''}k.[\![e^{\ell_0}]\!]^{Exp}k)^{\ell_2})^\ell & \widehat{C}'_b(\ell_2)=\widehat{C}_b(\ell)\quad \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\\
&&& \widehat{\rho}'_b(x)=\widehat{\rho}_b(x)\quad \widehat{\rho}'_b(k)=\widehat{C}_b(\ell)\\
[\![e^\ell]\!]^{Root} & = & (\lambda^{\ell_0}k.[\![e^\ell]\!]^{Exp}k)^{\ell_1} & \widehat{C}'_b(\ell_1)=\mathbf{D}\quad \widehat{\rho}'_b(k)=\mathbf{D}\\
[\![t^\ell]\!]^{Exp}k & = & (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0} & \widehat{C}'_b(\ell_1)=\widehat{\rho}'_b(k)\quad \widehat{C}'_b(\ell_0)=\mathbf{D}\\
[\![(t_0^{\ell_0}\ t_1^{\ell_1})^\ell]\!]^{Exp}k & = & (\mathbf{let}\ v_0=([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{\rho}'_b(v)=\widehat{C}_b(\ell)\quad \widehat{C}'_b(\ell_5)=\widehat{C}_b(\ell_0)\\
&& \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'}v.(k\ v)^{\ell_4})^{\ell_5})^{\ell_6})^{\ell_7} & \widehat{C}'_b(\ell_4)=\widehat{C}'_b(\ell_6)=\widehat{C}'_b(\ell_7)=\mathbf{D}\\
&&& \widehat{\rho}'_b(v_0)=\widehat{C}'_b(\ell_3)=\widehat{C}'_b(\ell_2)=\widehat{C}_b(\ell_0)\\
[\![(op(t_0^{\ell_0}))^\ell]\!]^{Exp}k & = & (\mathbf{let}\ v=(op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell & \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\quad \widehat{\rho}'_b(v)=\widehat{C}'_b(\ell)\\
&& \mathbf{in}\ (k\ v)^{\ell_1})^{\ell_2} & \widehat{C}'_b(\ell_1)=\widehat{C}'_b(\ell_2)=\mathbf{D}\\
[\![(\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}]\!]^{Exp}k & = & (\mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp}k'\ [\![e_1^{\ell_1}]\!]^{Exp}k')^{\ell_3} & \widehat{C}'(\ell_3)=\mathbf{D}\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(t_0^{\ell_0}\ t_1^{\ell_1})^\ell\\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp}k & = & (\mathbf{let}\ v_0=([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} & \widehat{\rho}'_b(v)=\widehat{C}_b(\ell)\quad \widehat{C}'_b(\ell_5)=\widehat{C}_b(\ell_0)\\
&& \mathbf{in}\ (v_0^{\ell_3}\ (\lambda^{\ell'}v.[\![e^{\ell_2}]\!]^{Exp}k)^{\ell_5})^{\ell_6})^{\ell_7} & \widehat{C}'_b(\ell_6)=\widehat{C}'_b(\ell_7)=\mathbf{D}\\
&&& \widehat{\rho}'_b(v_0)=\widehat{C}'_b(\ell_3)=\widehat{C}'_b(\ell_2)=\widehat{C}_b(\ell_0)\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(op(t_0^{\ell_0}))^\ell\\ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp}k & = & (\mathbf{let}\ v=(op([\![t_0^{\ell_0}]\!]^{Triv}))^\ell & \widehat{\rho}'_b(v)=\widehat{\rho}_b(v)\quad \widehat{C}'_b(\ell)=\widehat{C}_b(\ell)\\
&& \mathbf{in}\ [\![e^{\ell_2}]\!]^{Exp}k)^{\ell_4} & \widehat{C}'_b(\ell_4)=\mathbf{D}\\
&& ((\lambda^{\ell'}k'.(\mathbf{if0}\ [\![t^\ell]\!]^{Triv} & \\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ v=(\mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}\\ \mathbf{in}\ e^{\ell_3})^{\ell_4}\end{array}\right]\!\!\right]^{Exp}k & = & \qquad [\![e_0^{\ell_0}]\!]^{Exp}k' & \widehat{\rho}'_b(v)=\widehat{C}_b(\ell_2)\\
&& \qquad [\![e_1^{\ell_1}]\!]^{Exp}k')^{\ell_5})^{\ell_6} & \widehat{C}'_b(\ell_6)=\widehat{C}_b(\ell)\quad \widehat{C}'_b(\ell_7)=\widehat{C}_b(\ell)\\
&& (\lambda^{\ell''}v.[\![e^{\ell_3}]\!]^{Exp}k)^{\ell_7})^{\ell_8} & \widehat{C}'_b(\ell_5)=\widehat{C}'_b(\ell_8)=\mathbf{D}
\end{array}
$$

Figure 11: Transformation of binding times from direct style to CPS

such that $\ell_3 \in \widehat{C}(\ell_7)$, which means that the lambda-abstraction $\lambda^{\ell_3}x.e^{\ell_4}$ is called at the application point ℓ_8 in p, which implies:

$$\widehat{C}(\ell_4) = \widehat{C}(\ell_8) = \widehat{\rho}'(v)$$

On the other hand, since $\ell \in \mathcal{R}(\ell_4)$, using Lemma 1, yields:

$$\widehat{C}'(\ell) = \widehat{C}(\ell) = \widehat{C}(\ell_4) = \widehat{C}(\ell_8) = \widehat{\rho}'(v)$$

But we are already almost repeating ourselves.

The line of reasoning repeats throughout. $\square$

We now show that a reverse transformation is not possible. There are cases when the least analysis on CPS programs produces more static trivial terms than the same analysis on the direct-style term. Here is a standard example [13]:

$$\textbf{let } r = (\lambda y.(\textbf{let } v = \mathit{print}(z) \textbf{ in } 2)\ 1) \textbf{ in } \mathit{inc}(r)$$

Here we consider *print* as a side-effecting operator and *inc* as the successor function. The free variable z is considered to be dynamic. It is not difficult to find the least solution of the BTA on this term. Even if the application of $\lambda y...$ to 1 is classified as static, the result is classified as dynamic, because of the dynamic header of the **let**-expression, and thus r is dynamic. Since the increment computation depends on r, it is dynamic as well.

The CPS version is as follows:

$$\begin{array}{l}\lambda k.\textbf{let } v_0 = (\lambda y.\lambda k_1.\textbf{let } v = \mathit{print}(z) \textbf{ in } k_1\ 2)\ 1 \\ \quad \textbf{in } v_0\ (\lambda r.\textbf{let } r_1 = \mathit{inc}(r) \textbf{ in } k\ r_1)\end{array}$$

In this case, however, the continuation k_1 is static, and the application k_1 2 is static as well. Thus, r is static, and further computation based on r can be performed at specialization time.

We should note, however, that other examples of improvements are possible where a dynamic test disables further computation based on the result. The standard example is as follows:

$$\begin{array}{l}\textbf{let } v = \textbf{if0 } z\ 1\ 2 \\ \textbf{in } \mathit{inc}(v)\end{array}$$

It is true that this improvement leads to code duplication, but this code duplication happens at specialization time, not at BTA time. Therefore, in opposition to Sabry and Felleisen's observation, the improvement is not due to duplication of the analysis on the two branches. The cost of such improvements is thus not to be compared with the undecidability produced by the duplication of the data-flow analysis.

4.2 Continuation-based partial evaluation

From the two examples above we can see that the binding-time improvements come from two special constraints in the specification of the BTA: the body of a **let** has to be dynamic if the header is dynamic, and both branches of an **if0** have to be dynamic if the test is dynamic.

These constraints can be lifted if one uses a *continuation-based program specializer* [2, 13, 19]. Given such a specializer, we can show that lifting the two special constraints nulls the impact of the CPS on the result of BTA.

More precisely, using the same proof technique for a modified binding-time analysis, we can formally show that the CPS transformation has no effect, i.e., no local increase and also no loss of precision elsewhere in the program: the best binding times in direct style are the best binding times in CPS as well.

4.3 Summary and conclusion

Theorem 4 and Lemma 4 show that we can translate any binding-time solution of a direct-style program into a solution of its CPS counterpart in such a way that the binding times of the trivial terms are preserved. Therefore, the same amount of specialization of the input program can be achieved after CPS transformation.

Through two examples, we have shown that there are cases where the BTA on CPS terms gives more precise results than on the corresponding direct-style terms. The binding-time improvements can be observed both for **let**-constructs with dynamic headers and for conditional expressions with dynamic tests.

We have also shown that if we modify the binding-time rules for **let**-constructs and for conditional expressions (which requires a continuation-based specializer to process the resulting binding-time information [13]), then, just like 0-CFA, no loss or gain of information is incurred by the CPS transformation.

5. RELATED WORK

5.1 Program analysis in general

With the exception of Nielson's early work on data-flow analysis [21],[2] we are not aware of any study of the effect of program transformation on the precision of program analysis in general.[3] For one data-flow analysis, constant propagation, Sabry and Felleisen have shown that CPS transformation yields incomparable results [31].

Recently, Palsberg and Wand have performed a similar study for 0-CFA [30], supporting Sabry and Felleisen's conclusion that the extra precision enabled by CPS transformation is due to the duplication of program points. Palsberg and Wand's CPS transformation of flow types was developed independently, but is otherwise comparable to ours in Section 3. Palsberg and Wand implicitly share our concern about syntactic accidents, in that they mention that least solutions may or may not be preserved by administrative reductions.

[2]Actually, Nielson's work is only indirectly connected since it addresses a continuation semantics instead of a direct semantics of a CPS-transformed program.

[3]The functional-programming technique of "strictification", for example, does not fit the bill.

5.2 Binding-time analysis and the CPS transformation

Binding-time improvements have always been customary for users of binding-time analysis [18, 22]. One of them amounts to considering source programs in CPS [4, 7], which suggests that source programs could be systematically CPS-transformed instead [3].

Essentially, the CPS transformation relocates potentially static contexts inside definitely dynamic contexts (let expressions and case expressions), thereby providing a binding-time improvement. To this end, the CPS transformation itself is continuation-based [8], which paved the way to continuation-based partial evaluation [2, 19].

The full effect of continuation-based partial evaluation has been characterized as online let flattening in Moggi's computational meta-language [13]. This characterization justifies why offline let flattening is also, partially, a binding-time improvement [15]. In any case, offline let flattening is now known to be part of the CPS transformation [12].

What has not been shown, however, and what we do show here, is whether such 'improvements' do not worsen binding times elsewhere in a source program.

6. CONCLUSION AND ISSUES

Observing that program analyses are susceptible to syntactic accidents, we have studied the interaction between a non-duplicating CPS transformation and two program analyses: control-flow analysis (0-CFA) and binding-time analysis (BTA). Through a systematic construction of the CPS counterpart of flow information, similar to Palsberg and Wand's [30] but developed independently, we have found that 0-CFA is insensitive to continuation-passing, and that the CPS transformation does improve binding times. Using the same technique, we have also found that with a modified let rule, BTA is insensitive to continuation-passing.

These results suggest two further avenues of study:

- In BTA, the beneficial effect of the CPS transformation can be accounted for by tuning the let rule (as well as the case rule, if one is willing to duplicate static contexts at specialization time). The price of this change, however, is that the corresponding program specializer has to be made continuation-based. We conjecture that the situation is similar, e.g., for security analysis, which has a similar let rule. Just like BTA, a security analysis thus ought to yield more precise results over CPS-transformed programs. We therefore also conjecture that the beneficial effect of the CPS transformation can be accounted for by tuning the let rule, if one is willing to develop a corresponding continuation-based processor of security information.

- More generally, as a step towards more robust program analyses that are less susceptible to syntactic accidents, we need to understand better the program-analysis perspective over syntactic landscapes. Two key questions arise, that may be general to program analysis or specific to individual program analyses: which program transformations affect precision? and among those that do, which ones affect precision *monotonically*? Answering these questions would enable one to develop more robust program analyses, either with some kind of subject reduction property or with some kind of intermediate language for program analysis.

7. REFERENCES

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In Alex Aiken, editor, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 147–160, San Antonio, Texas, January 1999. ACM Press.

[2] Anders Bondorf. Improving binding times without explicit cps-conversion. In William Clinger, editor, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. V, No. 1, pages 1–10, San Francisco, California, June 1992. ACM Press.

[3] Charles Consel and Olivier Danvy. For a better support of static data flow. In John Hughes, editor, *Proceedings of the Fifth ACM Conference on Functional Programming and Computer Architecture*, number 523 in Lecture Notes in Computer Science, pages 496–519, Cambridge, Massachusetts, August 1991. Springer-Verlag.

[4] Charles Consel and Olivier Danvy. Static and dynamic semantics processing. In Robert (Corky) Cartwright, editor, *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 14–24, Orlando, Florida, January 1991. ACM Press.

[5] Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In Graham [11], pages 493–501.

[6] Daniel Damian. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, 2001. Forthcoming.

[7] Olivier Danvy. Semantics-directed compilation of non-linear patterns. *Information Processing Letters*, 37:315–322, March 1991.

[8] Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

[9] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In David W. Wall, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Languages Design and Implementation*, SIGPLAN Notices, Vol. 28, No 6, pages 237–247, Albuquerque, New Mexico, June 1993. ACM Press.

[10] Kirsten L. Solberg Gasser, Flemming Nielson, and Hanne Riis Nielson. Systematic realisation of control flow analyses for CML. In Mads Tofte, editor, *Proceedings of the 1997 ACM SIGPLAN International*

Conference on Functional Programming, pages 38–51, Amsterdam, The Netherlands, June 1997. ACM Press.

[11] Susan L. Graham, editor. *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, Charleston, South Carolina, January 1993. ACM Press.

[12] John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 458–471, Portland, Oregon, January 1994. ACM Press.

[13] John Hatcliff and Olivier Danvy. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science*, pages 507–541, 1997. Extended version available as the technical report BRICS RS-96-34.

[14] Nevin Heintze. Set-based program analysis of ML programs. In Talcott [33], pages 306–317.

[15] Carsten K. Holst and Carsten K. Gomard. Partial evaluation is fuller laziness. In Paul Hudak and Neil D. Jones, editors, *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, SIGPLAN Notices, Vol. 26, No 9, pages 223–233, New Haven, Connecticut, June 1991. ACM Press.

[16] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In Graham [11], pages 393–407.

[17] Neil D. Jones. What *not* to do when writing an interpreter for specialisation. In Olivier Danvy, Robert Glück, and Peter Thiemann, editors, *Partial Evaluation*, number 1110 in Lecture Notes in Computer Science, pages 216–237, Dagstuhl, Germany, February 1996. Springer-Verlag.

[18] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, 1993. Available online at `http://www.dina.kvl.dk/~sestoft/pebook/pebook.html`.

[19] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Talcott [33].

[20] J.A. Muylaert-Filho and G. L. Burn. Continuation passing transformation and abstract interpretation. Technical Report Doc 93/35, Department of Computing, Imperial College, London, England, 1993.

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

[22] Flemming Nielson and Hanne Riis Nielson. *Two-Level Functional Languages*, volume 34 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

[23] Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In Neil D. Jones, editor, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 332–345, Paris, France, January 1997. ACM Press.

[24] Flemming Nielson and Hanne Riis Nielson. Interprocedural control flow analysis. In S. Doaitse Swierstra, editor, *Proceedings of the Eighth European Symposium on Programming*, number 1576 in Lecture Notes in Computer Science, pages 20–39, Amsterdam, The Netherlands, March 1999. Springer-Verlag.

[25] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.

[26] Jens Palsberg. Correctness of binding-time analysis. *Journal of Functional Programming*, 3(3):347–363, 1993.

[27] Jens Palsberg. Comparing flow-based binding-time analyses. In Peter Mosses, Mogens Nielsen, and Michael Schwartzbach, editors, *Proceedings of TAPSOFT '95*, number 915 in Lecture Notes in Computer Science, pages 561–574, Aarhus, Denmark, May 1995. Springer-Verlag.

[28] Jens Palsberg and Patrick O'Keefe. A type system equivalent to flow analysis. *ACM Transactions on Programming Languages and Systems*, 17(4):576–599, July 1995.

[29] Jens Palsberg and Michael I. Schwartzbach. Binding-time analysis: Abstract interpretation versus type inference. In *Proceedings of the Fifth IEEE International Conference on Computer Languages*, pages 289–298. IEEE Computer Society Press, 1994.

[30] Jens Palsberg and Mitchell Wand. CPS transformation of flow information. Unpublished manuscript, available at `http://www.cs.purdue.edu/homes/palsberg/publications.html`, March 2000.

[31] Amr Sabry and Matthias Felleisen. Is continuation-passing useful for data flow analysis? In Vivek Sarkar, editor, *Proceedings of the ACM SIGPLAN'94 Conference on Programming Languages Design and Implementation*, SIGPLAN Notices, Vol. 29, No 6, pages 1–12, Orlando, Florida, June 1994. ACM Press.

[32] Guy L. Steele Jr. Rabbit: A compiler for Scheme. Technical Report AI-TR-474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978.

[33] Carolyn L. Talcott, editor. *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. VII, No. 3, Orlando, Florida, June 1994. ACM Press.