Syntactic Accidents in Program Analysis: on the Impact of the CPS Transformation *

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 no effect on control-flow analysis and that it has a positive effect on binding-time analysis: a monovariant control-flow analysis yields strictly comparable results on a direct-style program and on its CPS counterpart; and a monovariant binding-time analysis yields more precise results on a CPS program than on its direct-style counterpart. Our proof technique amounts to constructing the continuation-passing style (CPS) counterpart of flow information and of binding times.

Our results confirm a folklore theorem about binding-time analysis, namely that CPS has a positive impact on binding times. What may be more surprising is that this benefit holds even if contexts or continuations are not duplicated.

The present study is symptomatic of an unsettling property of program analyses: their quality is unpredictably vulnerable to syntactic accidents in source programs, i.e., to the way these programs are written. More reliable program analyses require a better understanding of the effects of syntactic change.

1. INTRODUCTION AND MOTIVATION

Program analyses are vulnerable 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 vulnerable to associativity: given two static expressions s_1 and s_2 and one dynamic expression d, it makes a difference whether the source program is expressed as $(s_1 + s_2) + d$ or as $s_1 + (s_2 + d)$. In the former case, the inner addition is classified as static and the outer one is classified as dynamic. In the latter case, both additions are classified as dynamic.

With the exception of BTA, little is known about the impact of programming style on program analyses. BTA is an exception because its output critically determines the amount of specialization carried out by an offline partial evaluator [5, 16]. For this reason, partial-evaluation users are encouraged to write programs in a very specific style [15]. Furthermore, a panoply of tricks has been developed to milk out extra precision from binding-time analyses—the so-called "binding-time improvements" [16, Chapter 12]. We note that binding-time improvements are not restricted to users of offline partial-evaluators—they are also routine in staging transformations [17] and in the formal specification of programming languages for semantics-directed compiling [20, Section 8.2].

Since one of the most effective binding-time improvements is the transformation of the source program into continuation-passing style (CPS) [3, 29], it has been wondered whether CPS may help program analysis in general. Nielson's early work on data-flow analysis [19] 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. The CPS transformation is therefore A Good Thing, since for a direct semantics, it gives the effect of a continuation semantics. In the early 90s, Muylaert-Filho and Burn's work [8] 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?" [28], Sabry and Felleisen showed that for constant propagation, analyzing direct-style programs or their CPS counterparts yields incomparable results. They showed that CPS might make things better by duplicating continuations, and also that CPS might make things worse by confusing return points. These results are essentially confirmed by Palsberg and Wand's CPS-transformation of flow information [27]. At any rate, except for continuation-based partial evaluation [11], there seems to have been no

*Draft, 00-07-18. Please do not distribute.
Comments are welcome.
Extended version to appear as the technical report BRICS-RS-00-15.

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

further work about the effect of CPS on the precision of program analysis in general.

The situation is therefore that the CPS transformation is known to have an unpredictable effect on data-flow analysis and is also believed to have a positive effect on binding-time analysis. However, we do not know for sure whether this positive effect is truly positive, or whether it makes binding times worse elsewhere in the source program. One may also wonder whether there exist analyses on which CPS has no effect.

In this article, we answer these two questions by studying the effect of a non-duplicating CPS transformation on two off-the-shelf constraint-based program analyses—control-flow analysis (0-CFA) and BTA. Using a uniform proof technique, we formally show that:

(1) CPS has no impact on the 0-CFA, i.e., it yields equivalent results.

(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.

(3) CPS has no impact on an enhanced BTA which takes into account continuation-based partial evaluation

This increased precision entailed by CPS 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 display a similar symptom: for example, we are told that in practice, users tend to find security 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 security.)

Our central example is the let rule. A let-expression is classified to be dynamic (resp. with a high security) if the computation of its header is dynamic (resp. with a high security), even though its body could be classified as static (resp. with a low security). The body of the following λ-abstraction is thus classified as dynamic if e is dynamic, regardless of b:

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

The CPS counterpart of this λ-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]This intuition is formalized in the rest of this article.

(1) The paragraph above is the standard motivation for improving binding times by CPS transformation [3] (see Section 5.2 for further detail). However, what this paragraph leaves unsaid, and what actually has always been left unsaid, 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).

Together, these two observations tell us that the let rule is overly conservative in BTA, security analysis, etc. CPS makes it possible to exploit the untapped precision of this rule 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 [28]. 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.

Let us also point out that the CPS transformation leads to binding-time improvements for conditionals. Indeed, the case rule makes conditional branches dynamic if the test is dynamic. This approximation can be circumvented with a CPS transformation. The improvement, however, is not produced by the duplication of the analysis, but merely by the context relocation induced when transforming to CPS.

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 the direct-style λ-language of untyped terms given by the grammar in Figure 1. The language patterns the call-by-value encoding of PCF into Moggi's computational meta-language after let-flattening [10]. Terms in *Triv* represent values, while those in *Step* and *Exp* represent computations.

In a program, terms occurrences are identified by unique labels ℓ taken from a countable set *Lab*. In addition, λ-abstractions are identified by unique labels ℓ' from another set *Lam*, so that, for example, in $(\lambda^{\ell'} x.e^{\ell_1})^{\ell_0}$, ℓ_0 and ℓ_1 belong to *Lab* and ℓ' belongs to *Lam*.

We freely refer to terms using their labels and vice versa.

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

Figure 1: The direct-style language

$$
[\![e]\!]^{Pgm} = \lambda k.[\![e]\!]^{Exp} k \quad \text{where } k \text{ is fresh}
$$

$$
[\![n]\!]^{Triv} = n \qquad [\![\lambda x.e]\!]^{Triv} = \lambda x.\lambda k.[\![e]\!]^{Exp} k
$$
$$
[\![x]\!]^{Triv} = x \qquad \text{where } k \text{ is fresh}
$$

$$
\begin{array}{rcl}
[\![t]\!]^{Exp} k & = & k\ [\![t]\!]^{Triv} \\
[\![t_0\ t_1]\!]^{Exp} k & = & \textbf{let } x_1 = [\![t_0]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\
 & & \textbf{in } x_1\ \lambda x.k\ x \\
 & & \text{where } x_1 \text{ and } x \text{ are fresh} \\
[\![op(t)]\!]^{Exp} k & = & \textbf{let } x_1 = op([\![t]\!]^{Triv}) \textbf{ in } k\ x_1 \\
 & & \text{where } x_1 \text{ is fresh} \\
[\![\textbf{if0 } t\ e_0\ e_1]\!]^{Exp} k & = & \textbf{if0 } [\![t]\!]^{Triv}\ [\![e_0]\!]^{Exp} k\ [\![e_1]\!]^{Exp} k \\
[\![\textbf{let } x = t \textbf{ in } e]\!]^{Exp} k & = & (\lambda x.[\![e]\!]^{Exp} k)\ [\![t]\!]^{Triv} \\
\left[\!\!\left[\begin{array}{l}\textbf{let } x = t_0\ t_1 \\ \textbf{in } e\end{array}\right]\!\!\right]^{Exp} k & = & \textbf{let } x_1 = [\![t_0]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\
 & & \textbf{in } x_1\ \lambda x.[\![e]\!]^{Exp} k \\
 & & \text{where } x_1 \text{ is fresh} \\
\left[\!\!\left[\begin{array}{l}\textbf{let } x = op(t) \\ \textbf{in } e\end{array}\right]\!\!\right]^{Exp} k & = & \textbf{let } x = op([\![t]\!]^{Triv}) \\
 & & \textbf{in } [\![e]\!]^{Exp} k \\
\left[\!\!\left[\begin{array}{l}\textbf{let } x = \\ \quad \textbf{if0 } t\ e_0\ e_1 \\ \textbf{in } e\end{array}\right]\!\!\right]^{Exp} k & = & \textbf{let } k_1 = \lambda x.[\![e]\!]^{Exp} k \\
 & & \textbf{in if0 } [\![t]\!]^{Triv} [\![e_0]\!]^{Exp} k_1 \\
 & & \qquad\qquad [\![e_1]\!]^{Exp} k_1 \\
 & & \text{where } k_1 \text{ is fresh}
\end{array}
$$

Figure 2: CPS transformation

The CPS transformation of direct-style terms yields terms in a CPS language. For simplicity, we consider the CPS language as a subset of the direct-style language, so for example, applications are transformed into **let**-expressions naming partially applied CPS λ-abstractions. Figure 2 displays the corresponding CPS transformation. (We have omitted the labels, because they only matter in the following sections. Suffices to say that for simplicity we label each CPS trivial term with the same label as its direct-style counterpart.)

3. CONTROL-FLOW ANALYSIS

We consider a monovariant control-flow analysis over direct-style programs. For simplicity, we choose a constraint-based analysis [9, 14, 21, 24]. The constraint-based method is equivalent to other program-analysis methods such as set-based analysis [12] and type inference [25]. For simplicity, we adopt the same definition and notation as in Nielson, Nielson and Hankin's recent textbook on program analysis [22].

$$
\begin{array}{rl}
Lam_p & \text{The set of } \lambda\text{-abstraction labels in } p \\
Var_p & \text{The set of identifiers in } p \\
Lab_p & \text{The set of term labels in } p
\end{array}
$$

$$
\begin{array}{rcll}
Val_p & = & \mathcal{P}(Lam_p) & \text{Flow values} \\
\widehat{C} \in Cache_p & = & Lab_p \to Val_p & \text{Cache} \\
\widehat{\rho} \in Env_p & = & Var_p \to Val_p & \text{Environment}
\end{array}
$$

$$
\vDash_p : (Cache_p \times Env_p) \times Lab_p \to \{true, false\}
$$

Figure 3: 0-CFA predicate for a program p

The flow information computed by the analysis is a pair consisting of a cache $\widehat{C}$ mapping terms to flow values and an environment $\widehat{\rho}$ mapping variables to flow values. Flow values are sets of labels of λ-abstractions that a term can yield and that a variable can denote. The constraint-based control-flow analysis is specified as a predicate $\vDash$ on caches, environments and terms. Given a term e, $(\widehat{C}, \widehat{\rho}) \vDash e$ means that $(\widehat{C}, \widehat{\rho})$ is a result of the analysis of e.

In this work we use the syntax-directed variant of the analysis [22, Chapter 3], which restricts the universal predicate $\vDash$ to a predicate $\vDash_p$ associated to each program being analyzed. Given a closed direct-style program p, the functionality of the associated predicate $\vDash_p$ is defined in Figure 3. The predicate is defined in Figure 4 by induction over the syntax of the program.

Any solution $(\widehat{C}, \widehat{\rho})$ accepted by the predicate $\vDash_p$ (i.e., such that $(\widehat{C}, \widehat{\rho}) \vDash_p p$) is a conservative approximation of the flow information [22, Chapter 3]. Furthermore, the predicate $\vDash_p$ has a model-intersection property. i.e., the set of solutions accepted by $\vDash_p$ is closed under intersection. The model-intersection property ensures the existence of a least solution of the analysis, i.e., a most precise one. (Here the order relation is given by the pointwise ordering of functions induced by set inclusion.) In practice, a work-list based algorithm computes the least solution.

The rest of this section is organized as follows. First, we show how to CPS-transform control-flow information (Section 3.1). Given a direct-style program and an arbitrary solution for its associated analysis, we construct a solution for the analysis associated to the CPS counterpart of the program. We then ensure that the construction provides a valid solution (Section 3.2). We present a converse transformation (Section 3.3), which we also prove to be correct (Section 3.4). The specification of the analysis puts us in an ideal position to compare absolute precisions (Section 3.5). We show that the least solution of the analysis on an arbitrary program is transformed into the least solution of the same analysis on its CPS counterpart of this program 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 CPS transformation of control flow

Given a solution $(\widehat{C}, \widehat{\rho})$ of the analysis of a program p (i.e., a cache-environment pair such that $(\widehat{C}, \widehat{\rho}) \vDash_p p$), we now con-

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

Figure 4: Syntax-directed 0-CFA

struct a solution $(\widehat{C}',\widehat{\rho}')$ of the analysis of $p' = [\![p]\!]^{Pgm}$, the CPS counterpart of p (i.e., the solution satisfies $(\widehat{C}',\widehat{\rho}') \vDash_{p'} p'$). By analogy, we refer to the construction of $(\widehat{C}',\widehat{\rho}')$ out of $(\widehat{C},\widehat{\rho})$ as the CPS transformation of $(\widehat{C},\widehat{\rho})$ into $(\widehat{C}',\widehat{\rho}')$.

As mentioned in Section 2, we have designed the CPS transformation on labeled terms so that it preserves the labels of each trivial term and that each direct-style λ-abstraction is annotated with the same label as its CPS counterpart. As a consequence, the flow values in direct style are included into the flow values in CPS, i.e., $Lam_p \sqsubseteq Lam_{p'}$ and $Val_p \sqsubseteq Val_{p'}$. The CPS transformation preserves all the variables defined in the original direct-style program. Therefore $Var_p \sqsubseteq Var_{p'}$. In essence, we construct a solution for the CPS program such that the flow information assigned to the variables and to the trivial terms preserved by the transformation is at least as precise as in the direct-style solution.

We also assign flow information to the newly introduced terms and variables, in particular to continuation abstractions and continuation identifiers. To this end, we use two auxiliary functions: γ and ξ.

γ extracts the labels of partially applied CPS λ-abstractions. Formally, considering A to be a set of CPS λ-abstractions $\{\lambda^{\ell'_i} x_i.\lambda^{\ell''_i} k_i.e_i | 1 \le i \le n\}$, for some n, then $\gamma(A) = \{\ell''_i | 1 \le i \le n\}$.

ξ assigns flow information to each continuation identifier k introduced by the CPS transformation. This information can be obtained from the direct-style flow information. For instance, we can *syntactically* identify the continuation of the CPS counterpart of any direct-style application. This is the reason why we introduce an explicit λ-abstraction ($\lambda x.k\ x$, in Figure 2) for each application.

We define $\xi(k)$ by case analysis where k is introduced, using the direct-style flow information:

• If k is introduced at the top level of the program:

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

then $\xi(k) = \emptyset$. In other words, we consider that no top-level continuation is provided: we are not interested in functions flowing outside of the program.

• If k is introduced by the transformation of a named **if0**:

$$\left[\!\!\left[\begin{array}{l}\mathbf{let}\ x = \mathbf{if0}\ t\ e_0\ e_1 \\ \mathbf{in}\ e\end{array}\right]\!\!\right]^{Exp} k_1 = \begin{array}{l}\mathbf{let}\ k = \lambda^{\ell'} x.[\![e]\!]^{Exp} k_1 \\ \mathbf{in}\ \ldots [\![e_0]\!]^{Exp} k \ldots\end{array}$$

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

• If k is the continuation of a function, i.e., it is introduced by the transformation of a λ-abstraction:

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

then $\xi(k)$ gathers all continuations that are passed at each program point where $\lambda^{\ell''} x.e$ can be applied. Formally, $\xi(k)$ is defined as the set of all labels ℓ' such that:

- there exists a transformation step

$$[\![t_0^{\ell_0}\ t_1]\!]^{Exp} k_1 = \begin{array}{l}\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \mathbf{in}\ x_0\ \lambda^{\ell'} x.k_1\ x\end{array}$$

  such that $\ell'' \in \widehat{C}(\ell_0)$, or

- there exists a transformation step

$$\left[\!\!\left[\begin{array}{l}\mathbf{let}\ x = t_0^{\ell_0}\ t_1 \\ \mathbf{in}\ e\end{array}\right]\!\!\right]^{Exp} k_1 = \begin{array}{l}\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \mathbf{in}\ x_0\ \lambda^{\ell'} x.[\![e]\!]^{Exp} k_1\end{array}$$

  such that $\ell'' \in \widehat{C}(\ell_0)$.

Using γ and ξ, we define $(\widehat{C}',\widehat{\rho}')$ by a top-down descent on the output of the CPS transformation, following the left part of Figure 5. In the right part, for each CPS-transformation step, we assign flow values into $\widehat{C}'$ and $\widehat{\rho}'$ using previously defined values.

It is straightforward to show that $\widehat{C}'$ and $\widehat{\rho}'$ are well-defined. The construction of flow information is a function

$$\mathcal{C}_p : (Cache_p \times Env_p) \to (Cache_{p'} \times Env_{p'}).$$

It is also easy to show that $\mathcal{C}_p$ is monotone.

3.2 Correctness of the transformation

Let us show that the cache-environment pair constructed by $\mathcal{C}_p$ is indeed a valid solution for the analysis of the CPS counterpart of p.

THEOREM 1. *Let $(\widehat{C},\widehat{\rho}) \vDash_p p$ be a solution for the analysis of a direct-style program p and let $p' = [\![p]\!]^{Pgm}$. Let $(\widehat{C}',\widehat{\rho}') = \mathcal{C}_p(\widehat{C},\widehat{\rho})$. Then $(\widehat{C}',\widehat{\rho}') \vDash_{p'} p'$.*

$$
\begin{array}{rcll}
[\![e^{\ell}]\!]^{Pgm} & = & (\lambda^{\ell_0} k.[\![e^{\ell}]\!]^{Exp} k)^{\ell_1} & \widehat{C}'(\ell_1) = \{\ell_0\} \quad \widehat{\rho}'(k) = \xi(k) \\
\\
[\![n^{\ell}]\!]^{Triv} & = & n^{\ell} & \widehat{C}'(\ell) = \widehat{C}(\ell) \\
[\![x^{\ell}]\!]^{Triv} & = & x^{\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) = \{\ell''\} \\
 & & & \widehat{\rho}'(x) = \widehat{\rho}(x) \quad \widehat{\rho}'(k) = \xi(k) \\
\\
[\![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 & = & (\textbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{C}'(\ell_3) = \widehat{\rho}'(x_0) = \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
 & & \ \textbf{in}\ (x_0^{\ell_3}\ (\lambda^{\ell'} x.(k^{\ell_4}\ x^{\ell_5})^{\ell_6})^{\ell_7})^{\ell_8})^{\ell_9} & \widehat{C}'(\ell_4) = \widehat{\rho}'(k) \quad \widehat{C}'(\ell_7) = \{\ell'\} \\
 & & & \widehat{C}'(\ell_5) = \widehat{\rho}'(x) = \widehat{C}(\ell) \\
 & & & \widehat{C}'(\ell_9) = \widehat{C}'(\ell_8) = \widehat{C}'(\ell_6) = \emptyset \\
\\
[\![(op(t_0^{\ell_0}))^{\ell}]\!]^{Exp} k & = & (\textbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell} & \widehat{C}'(\ell_1) = \widehat{\rho}'(k) \quad \widehat{C}'(\ell_4) = \widehat{C}'(\ell_3) = \emptyset \\
 & & \ \textbf{in}\ (k^{\ell_1}\ x^{\ell_2})^{\ell_3})^{\ell_4} & \widehat{C}'(\ell_2) = \widehat{\rho}'(x) = \widehat{C}'(\ell) = \widehat{C}(\ell) \\
\\
[\![(\textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}]\!]^{Exp} k & = & (\textbf{if0}\ [\![t^{\ell}]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp} k\ [\![e_1^{\ell_1}]\!]^{Exp} k)^{\ell_3} & \widehat{C}'(\ell_3) = \emptyset \\
\\
[\![(\textbf{let}\ x = t_0^{\ell_0}\ \textbf{in}\ e^{\ell_1})^{\ell}]\!]^{Exp} k & = & ((\lambda^{\ell'} x.[\![e^{\ell_1}]\!]^{Exp} k)^{\ell_2}\ [\![t_0^{\ell_0}]\!]^{Triv})^{\ell_3} & \widehat{C}'(\ell_2) = \{\ell'\} \quad \widehat{C}'(\ell_3) = \emptyset \\
 & & & \widehat{\rho}'(x) = \widehat{\rho}(x) \\
\\
\left[\!\!\left[(\textbf{let}\ x = (t_0^{\ell_0}\ t_1^{\ell_1})^{\ell} \right.\right. & & & \widehat{C}'(\ell_5) = \widehat{\rho}'(x_0) = \widehat{C}'(\ell_4) = \gamma(\widehat{C}(\ell_0)) \\
\left.\left.\ \textbf{in}\ e^{\ell_2})^{\ell_3}\right]\!\!\right]^{Exp} k & = & (\textbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} & \widehat{C}'(\ell_6) = \{\ell'\} \quad \widehat{C}'(\ell_8) = \widehat{C}'(\ell_7) = \emptyset \\
 & & \ \textbf{in}\ (x_0^{\ell_5}\ (\lambda^{\ell'} x.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_6})^{\ell_7})^{\ell_8} & \widehat{\rho}'(x) = \widehat{C}(\ell) \\
\\
\left[\!\!\left[(\textbf{let}\ x = (op(t_0^{\ell_0}))^{\ell} \right.\right. & & & \widehat{C}'(\ell) = \widehat{C}(\ell) \quad \widehat{C}'(\ell_4) = \emptyset \\
\left.\left.\ \textbf{in}\ e^{\ell_2})^{\ell_3}\right]\!\!\right]^{Exp} k & = & (\textbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell} & \widehat{\rho}'(x) = \widehat{\rho}(x) \\
 & & \ \textbf{in}\ [\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4} & \\
\\
\left[\!\!\left[(\textbf{let}\ x = (\textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \right.\right. & & & \widehat{C}'(\ell_5) = \{\ell'\} \quad \widehat{C}'(\ell_6) = \widehat{C}'(\ell_7) = \emptyset \\
\left.\left.\ \textbf{in}\ e^{\ell_3})^{\ell_4}\right]\!\!\right]^{Exp} k & = & (\textbf{let}\ k_1 = (\lambda^{\ell'} x.[\![e^{\ell_3}]\!]^{Exp} k)^{\ell_5} & \widehat{\rho}'(k_1) = \xi(k_1) \quad \widehat{\rho}'(x) = \widehat{C}(\ell_2) \\
 & & \ \textbf{in}\ (\textbf{if0}\ [\![t^{\ell}]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp} k_1\ [\![e_1^{\ell_1}]\!]^{Exp} k_1)^{\ell_6})^{\ell_7} &
\end{array}
$$

Figure 5: Flow transformation from direct style to CPS

Under the assumptions of the theorem, we start by observing three immediate properties of the flow transformation:

LEMMA 1. *For all variables x in p, $\widehat{\rho}'(x) \sqsubseteq \widehat{\rho}(x)$; for all trivial terms t^{ℓ} in p, $\widehat{C}'(\ell) = \widehat{C}(\ell)$; and for all expressions e^{ℓ} in p', $\widehat{C}'(\ell) = \emptyset$.*

For an arbitrary expression, we define the notion of return labels to capture the points from which 0-CFA collects flow information, as shown just below in Lemma 2.

Definition 1. Given a labeled expression $e^{\ell} \in Exp$, we define the set $\mathcal{R}[\![e^{\ell}]\!]$ of return labels of e^{ℓ} by structural induction as follows:

$$
\begin{array}{rcl}
\mathcal{R}[\![t^{\ell}]\!] & = & \{\ell\} \\
\mathcal{R}[\![(t_0^{\ell_1}\ t_1^{\ell_1})^{\ell}]\!] & = & \{\ell\} \\
\mathcal{R}[\![(op(t^{\ell_1}))^{\ell}]\!] & = & \{\ell\} \\
\mathcal{R}[\![(\textbf{if0}\ t^{\ell_2}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell}]\!] & = & \mathcal{R}[\![e_0^{\ell_0}]\!] \cup \mathcal{R}[\![e_1^{\ell_1}]\!] \\
\mathcal{R}[\![(\textbf{let}\ x = s^{\ell_1}\ \textbf{in}\ e^{\ell_2})^{\ell}]\!] & = & \mathcal{R}[\![e^{\ell_2}]\!]
\end{array}
$$

LEMMA 2. *Let e^{ℓ} be an arbitrary subexpression of p. Then, for all ℓ_1 in $\mathcal{R}[\![e^{\ell}]\!]$, $\widehat{C}(\ell_1) \sqsubseteq \widehat{C}(\ell)$.*

The definition of return labels also follows the way in which continuations identifiers are propagated by the CPS transformation. The return labels provide a syntactic connection between the points where flow information is collected in direct style and the points where flow information is passed to continuations in CPS.

LEMMA 3. *Let k be a continuation identifier introduced by the transformation of a λ-abstraction from p:*

$$[\![\lambda^{\ell''} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\ell''} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

Then, for each $\lambda^{\ell'} x.e^{\ell_1} \in \widehat{\rho}'(k)$, and for each $\ell \in \mathcal{R}[\![\ell_0]\!]$, $\widehat{C}(\ell) \sqsubseteq \widehat{\rho}'(x)$.

The proof of the lemma explores the two possibilities of $\ell' \in \widehat{\rho}'(k)$ given by the definition of ξ. In both cases, we can show that $\widehat{C}(\ell_0) \sqsubseteq \widehat{\rho}'(x)$. We then use Lemma 2 to show that $\widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell_0)$.

PROOF OF THEOREM 1. The proof proceeds by induction on the transformation of p into p'. We focus on two main induction steps.

We show that $(\widehat{C}', \widehat{\rho}') \vDash_{p'} (k^{\ell_1}\ [\![t^{\ell}]\!]^{Triv})^{\ell_0}$. For an arbitrary continuation $\lambda^{\ell'} x.e^{\ell_2}$ in $\widehat{C}'(\ell_1) = \widehat{\rho}'(k)$, we show that the flow constraints are satisfied.

The first constraint is $\widehat{C}'(\ell) \sqsubseteq \widehat{\rho}'(x)$. By Lemma 1, $\widehat{C}'(\ell) = \widehat{C}(\ell)$. Since k is a continuation identifier that comes from the CPS transformation (and thus $\widehat{\rho}'(k) = \xi(k)$), we can make a case analysis on its introduction. If k is the top-level continuation, the constraint is vacuously satisfied. If k is introduced by the transformation of a named **if0**-test, then, (with labels and variables as in Figure 5), obviously $\widehat{C}(\ell) \sqsubseteq \widehat{C}(\ell_2) = \widehat{\rho}'(x)$. Otherwise, k comes from the transformation of a λ-abstraction $\lambda^{\ell_3} y.e^{\ell_4}$ from p, such that $\ell \in \mathcal{R}[\![e^{\ell_4}]\!]$. We apply Lemma 3.

The second constraint is $\widehat{C}'(\ell_2) \sqsubseteq \widehat{C}'(\ell_0)$. Following Lemma 1, it amounts to $\emptyset \sqsubseteq \emptyset$.

In the case of a transformation of an application $(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}$, we need to show that

$$(\widehat{C}', \widehat{\rho}') \vDash_{p'} \textbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv} \textbf{in}\ x_0\ \lambda^{\ell''} x_1.(k^{\ell_4}\ x_1^{\ell_5})^{\ell_6}$$

Since $\widehat{C}'(\ell_0) = \widehat{C}(\ell_0)$, most of the constraints hold directly from the induction hypothesis, the hypothesis on $(\widehat{C}, \widehat{\rho})$ and the definition of ξ and γ. The more interesting part is to show that $(\widehat{C}', \widehat{\rho}') \vDash_{p'} (k^{\ell_4}\ x^{\ell_5})^{\ell_6}$. In particular, for an arbitrary $\lambda^{\ell'} x.e_1 \in \widehat{\rho}'(k)$, we need to show that $\widehat{\rho}'(x_1) \sqsubseteq \widehat{\rho}'(x)$, which is equivalent to $\widehat{C}(\ell) \sqsubseteq \widehat{\rho}'(x)$. Similarly to the previous case, we make a case analysis on the introduction of k, and use Lemma 3 to prove the result. □

3.3 Reversing the transformation

In the previous section we have shown that direct-style flow information can be transformed into CPS flow information. We can also show that any result of the analysis on a CPS transformed program can be matched by a result of the analysis of its direct-style counterpart. Using again the structure given by the CPS 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 recover a valid solution $(\widehat{C}, \widehat{\rho}) \vDash_p p$.

Recovering a direct-style solution is significantly simpler than constructing the CPS solution. For variables and trivial terms in p, we are only "filtering out" the continuation functions from the results of the analysis on p'. For "return" program points, we assign flow information using the flow information for the variables of the continuation abstractions. We define the direct-style solution inductively following Figure 6. In the right part, for each CPS-transformation step, we assign flow values into $\widehat{C}$ and $\widehat{\rho}$, using $A =_p A'$ as a shorthand for $A = A' \cap \mathcal{P}(Lam_p)$. The left parts of Figures 5 and 6 are identical.

We can show that Figure 6 defines another function

$$\mathcal{D}_p : (Cache_{p'} \times Env_{p'}) \to (Cache_p \times Env_p).$$

It is also relatively easy to show that, like $\mathcal{C}_p$ in Section 3.2, $\mathcal{D}_p$ is monotone as well.

3.4 Correctness of the reverse transformation

Let us show that the reverse transformation indeed obtains a valid solution of the analysis of the original program.

THEOREM 2. *Let p be a direct-style program and p' be its CPS counterpart. Let $(\widehat{C}', \widehat{\rho}') \vDash_{p'} p'$ be a solution of the analysis on p', and $(\widehat{C}, \widehat{\rho}) = \mathcal{D}_p(\widehat{C}', \widehat{\rho}')$. Then $(\widehat{C}, \widehat{\rho}) \vDash_p p$.*

To prove Theorem 2, we use similar intermediate results as in Section 3.2. Working under the assumptions of the theorem, we observe two immediate properties of the reverse transformation:

LEMMA 4. *For all $x \in Var_p$, $\widehat{\rho}(x) = \widehat{\rho}'(x)$; for all trivial terms t^{ℓ} in p, $\widehat{C}(\ell) = \widehat{C}'(\ell)$.*

For an arbitrary expression, the new solution collects all the flow information from the return points of the expression.

LEMMA 5. *Let e^{ℓ} be an expression in p. Then*

$$\widehat{C}(\ell) = \bigsqcup_{\ell_1 \in \mathcal{R}[\![e^{\ell}]\!]} \widehat{C}(\ell_1).$$

As a parallel of Lemma 3, the following lemma connects the flow at the return points of functions with the flow collected for the variables declared by continuations.

LEMMA 6. *Let k be a continuation identifier introduced by the transformation of a λ-abstraction from p:*

$$[\![\lambda^{\ell''} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\ell''} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

Then, for each $\lambda^{\ell'} x.e^{\ell_1} \in \widehat{\rho}'(k)$ and for each $\ell \in \mathcal{R}[\![\ell_0]\!]$, $\widehat{C}(\ell) \sqsubseteq \widehat{\rho}'(x)$.

Unlike Lemma 3, the proof proceeds by case distinction on the term labeled with $\ell \in \mathcal{R}[\![\ell_0]\!]$. In each case we are able to show that the flow in $\widehat{C}(\ell)$ is passed into the variable x.

PROOF OF THEOREM 2. The proof is by induction on the transformation of p into p'. Again, we focus on two cases.

Let us show that the constraints for an application $(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}$ are satisfied. According to Figure 6, the application is transformed as

$$[\![(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}]\!]^{Exp} k = \textbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} \textbf{in}\ x_0\ \lambda x.(k\ x)$$

Take an arbitrary $\lambda y.e^{\ell_3} \in \widehat{C}(\ell_0)$. We have to show that $\widehat{C}'(\ell_1) \sqsubseteq \widehat{\rho}'(y)$. From Lemma 4, this is equivalent to $\widehat{C}(\ell_1) \sqsubseteq$

$$
\begin{array}{rcll}
[\![e^{\ell}]\!]^{Pgm} & = & (\lambda^{\ell_0} k.[\![e^{\ell}]\!]^{Exp} k)^{\ell_1} & \\
[\![n^{\ell}]\!]^{Triv} & = & n^{\ell} & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
[\![x^{\ell}]\!]^{Triv} & = & x^{\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) \\
[\![t^{\ell}]\!]^{Exp} k & = & (k^{\ell_1}\ [\![t^{\ell}]\!]^{Triv})^{\ell_0} & \\
[\![(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}]\!]^{Exp} k & = & (\textbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{C}(\ell) =_p \widehat{\rho}'(x) \\
 & & \textbf{in}\ (x_0^{\ell_3}\ (\lambda^{\ell'} x.(k^{\ell_4}\ x^{\ell_5})^{\ell_6})^{\ell_7})^{\ell_8})^{\ell_9} & \\
[\![(op(t_0^{\ell_0}))^{\ell}]\!]^{Exp} k & = & (\textbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell}\ \textbf{in}\ (k^{\ell_1}\ x^{\ell_2})^{\ell_3})^{\ell_4} & \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
[\![(\textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2}]\!]^{Exp} k & = & (\textbf{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) \\
[\![(\textbf{let}\ x = t_0^{\ell_0}\ \textbf{in}\ e^{\ell_1})^{\ell}]\!]^{Exp} k & = & ((\lambda^{\ell'} x.[\![e^{\ell_1}]\!]^{Exp} k)^{\ell_2}\ [\![t_0^{\ell_0}]\!]^{Triv})^{\ell_3} & \widehat{C}(\ell) = \widehat{C}(\ell_1) \quad \widehat{\rho}(x) = \widehat{\rho}'(x) \\
[\![(\textbf{let}\ x = (t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}\ \textbf{in}\ e^{\ell_2})^{\ell_3}]\!]^{Exp} k & = & (\textbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} & \widehat{\rho}(x) = \widehat{C}(\ell) =_p \widehat{\rho}'(x) \\
 & & \textbf{in}\ (x_0^{\ell_5}\ (\lambda^{\ell'} x.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_6})^{\ell_7})^{\ell_8} & \widehat{C}(\ell_3) = \widehat{C}(\ell_2) \\
[\![(\textbf{let}\ x = (op(t_0^{\ell_0}))^{\ell}\ \textbf{in}\ e^{\ell_2})^{\ell_3}]\!]^{Exp} k & = & (\textbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell}\ \textbf{in}\ [\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4} & \widehat{\rho}(x) =_p \widehat{\rho}'(x) \quad \widehat{C}(\ell) =_p \widehat{C}'(\ell) \\
 & & & \widehat{C}(\ell_3) = \widehat{C}(\ell_2) \\
\left[\!\!\left[\begin{array}{l}(\textbf{let}\ x = (\textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \\ \textbf{in}\ e^{\ell_3})^{\ell_4}\end{array} \right]\!\!\right]^{Exp} k & = & (\textbf{let}\ k_1 = (\lambda^{\ell'} x.[\![e^{\ell_3}]\!]^{Exp} k)^{\ell_5} & \widehat{C}(\ell_2) = \widehat{\rho}(x) =_p \widehat{\rho}'(x) \\
 & & \textbf{in}\ (\textbf{if0}\ [\![t^{\ell}]\!]^{Triv}\ [\![e_0^{\ell_0}]\!]^{Exp} k_1\ [\![e_1^{\ell_1}]\!]^{Exp} k_1)^{\ell_6})^{\ell_7} & \widehat{C}(\ell_4) = \widehat{C}(\ell_3)
\end{array}
$$

Figure 6: Flow transformation from CPS to direct style

$\widehat{\rho}(y)$, and follows from $(\widehat{C}', \widehat{\rho}') \models_{p'} ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2}$. We also have to show that $\widehat{C}(\ell_3) \sqsubseteq \widehat{C}(\ell)$. From Lemma 5, all we need to show is that for an arbitrary $\ell' \in \mathcal{R}[\![e^{\ell_3}]\!]$, $\widehat{C}(\ell') \sqsubseteq \widehat{\rho}'(x)$. This follows from Lemma 6.

In the case of the transformation of a named application:

$$
\left[\!\!\left[\begin{array}{l}\textbf{let}\ x = (t_0^{\ell_3}\ t_1)^{\ell_4} \\ \textbf{in}\ e_2\end{array} \right]\!\!\right]^{Exp} k = \begin{array}{l}\textbf{let}\ x_0 = [\![t_0^{\ell_3}]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\ \textbf{in}\ x_0\ \lambda^{\ell'} x.e^{\ell_2}\end{array}
$$

Consider an arbitrary λ-abstraction $\lambda^{\ell''} y.e_1^{\ell_5}$ from p such that $\ell'' \in \widehat{C}(\ell_3)$. Let the CPS transformation of the λ-abstraction be $\lambda^{\ell''} y.\lambda k_1.e_1$. Then $\ell' \in \widehat{\rho}'(k_1)$. We use Lemma 5 and Lemma 6 as in the previous case to obtain that $\widehat{C}(\ell_5) \sqsubseteq \widehat{C}(\ell_4)$. □

3.5 Equivalence of flow

It is a matter of tedious calculations to prove the following lemma:

LEMMA 7. *Given the direct-style program p and its CPS counterpart p', if $(\widehat{C}, \widehat{\rho}) \models_p p$ then $\mathcal{D}_p(\mathcal{C}_p(\widehat{C}, \widehat{\rho})) \sqsubseteq (\widehat{C}, \widehat{\rho})$ and if $(\widehat{C}', \widehat{\rho}') \models_{p'} p'$ then $\mathcal{C}_p(\mathcal{D}_p(\widehat{C}', \widehat{\rho}')) \sqsubseteq (\widehat{C}', \widehat{\rho}')$.*

From these two properties the following main theorem follows directly.

THEOREM 3 (EQUIVALENCE OF FLOW). *Given the direct-style 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 $\mathcal{C}_p(\widehat{C}, \widehat{\rho}) = (\widehat{C}', \widehat{\rho}')$ and $\mathcal{D}_p(\widehat{C}', \widehat{\rho}') = (\widehat{C}, \widehat{\rho})$.*

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. Lemma 1 and Lemma 4 show that the two solutions are in fact equal on the variables and program points common to the two programs. We conclude that, in this setting, no information is lost or gained by the CPS transformation.

4. BINDING-TIME ANALYSIS

We consider a constraint-based binding-time analysis (BTA) [11, 23, 24, 26]. The analysis determines binding times of program points and program variables. This information is used to annotate the source program for offline partial evaluation [5, 16, 23]. The results of the analysis therefore determine the static computation performed at specialization time.

The constraint-based BTA uses flow information to determine the binding times of the operators and operands of applications. Alternatively, we could have considered an analysis computing both flow and binding-time information at the same time, but this approach is known to be equivalent [24]. We have chosen to separate the flow analysis from the binding-time analysis in order to reuse the results from Section 3.

$$
\begin{array}{rcll}
Val^{bt} & = & \{\mathbf{S}, \mathbf{D}\} & \text{Binding times} \\
\widehat{C}_{bt} \in Cache_p^{bt} & = & Lab_p \to Val^{bt} & \text{Cache} \\
\widehat{\rho}_{bt} \in \quad Env_p^{bt} & = & Var_p \to Val^{bt} & \text{Environment}
\end{array}
$$

$$\vDash_p^{bt} : (Cache_p^{bt} \times Env_p^{bt}) \times Lab^\star \to \{true, false\}$$

Figure 7: BTA predicate for a program p

The formal definition of the analysis is similar to the definition of the 0-CFA of Section 3. The analysis is specified as a satisfaction predicate, defined on essentially the same domains (Figure 7); the difference is that the domain of abstract values is now the standard lattice $\{\mathbf{S}, \mathbf{D}\}$ of static and dynamic annotations. The analysis predicate is defined recursively over the syntax (Figure 8). The definition refers to the flow information $(\widehat{C}, \widehat{\rho})$, which is considered to be the least solution of the control-flow analysis of Section 3.

In contrast to the 0-CFA, our BTA accepts non-closed terms. Following the tradition, we consider the free variables to be dynamic. Their associated flow information is considered to be empty, which is the result of applying the 0-CFA to the program closed by abstraction over the free variables.

Another difference from the 0-CFA of Section 3 is that the constraints generated by the BTA are equality constraints. Moreover, additional constraints are generated for λ-abstraction, branching and naming of intermediate results. The importance of these additional constraints is discussed in Section 4.4. A proof of correctness of a specializer using the annotations obtained by the BTA can be found elsewhere [23].

The rest of the section is organized as follows. First, we define a CPS transformation of binding times (Section 4.1), which we show to be correct and to preserve the quality of the binding times (Section 4.2). Unlike for 0-CFA, however, we show examples where BTA on CPS terms gives more precise results than on the corresponding direct-style terms, thus showing that the CPS transformation may lead to more specialization opportunities (Section 4.3). Finally (Section 4.4) we show that if we relax the constraints of the BTA to take into account continuation-based partial evaluation, then, just like 0-CFA, no loss and no gain of information can be observed after the CPS transformation.

4.1 CPS transformation of binding times

We show that the binding times obtained by analyzing the CPS counterpart of a program are at least as good as the ones obtained by analyzing the original program. With the same technique as in Section 3, we construct a solution of the BTA over the CPS-transformed program from a solution of the BTA over the original program, such that the quality of the binding times is preserved.

Given the program p and $(\widehat{C}_{bt}, \widehat{\rho}_{bt})$ a solution of the BTA over p, we define $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt})$ as a solution of the BTA over p', the CPS counterpart of p. The definition is by induction on the CPS transformation and is given in Figure 9, where the left part is identical to the left part of Figures 5 and 6. In the right part, we assign binding times into $\widehat{C}'$ and $\widehat{\rho}'$. As in Section 3, we use $\mathcal{C}_p^{bt}$ to denote the function induced by the transformation.

4.2 Correctness of the transformation

Let us show that the solution defined in Figure 9 is indeed a valid solution of the BTA. We follow the same technique as in Section 3.2. The correctness of the transformation is established by the following theorem.

THEOREM 4. *Let $(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p p$ be a solution for the BTA of a direct-style program p, and let $p' = [\![p]\!]^{Pgm}$. Let $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt}) = \mathcal{C}_p^{bt}(\widehat{C}_{bt}, \widehat{\rho}_{bt})$. Then $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt}) \vDash_{p'}^{bt} p'$*

Under the assumption of the theorem, we first observe immediate properties of the CPS transformation of binding times, similar to the ones stated in Lemma 1.

LEMMA 8. *For all variables x in p, $\widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(x)$; for all trivial terms t^ℓ in p, $\widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\ell)$; and for all expressions e in p', $\widehat{C}'_{bt}(e) = \mathbf{D}$.*

The binding time of an expression is equal to the binding times of its return points.

LEMMA 9. *Given $(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^{bt} p$, let e^ℓ be an arbitrary subexpression of p. Then, for any ℓ_1 in $\mathcal{R}[\![e^\ell]\!]$, $\widehat{C}_{bt}(\ell_1) = \widehat{C}_{bt}(\ell)$.*

The flow of the continuation abstractions connects the binding times of return points of expressions and continuation variables. The binding time of the value abstracted by a continuation is equal to the binding time of any expression that the continuation can be passed to.

LEMMA 10. *Let k be a continuation identifier introduced by the transformation of a λ-abstraction from p:*

$$[\![\lambda^{\ell''} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\ell''} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

Then, for each $\lambda^{\ell'} x.e^{\ell_1} \in \widehat{\rho}'(k)$ and for each $\ell \in \mathcal{R}[\![\ell_0]\!]$, $\widehat{C}_{bt}(\ell) = \widehat{\rho}'_{bt}(x)$.

PROOF OF THEOREM 4. The proof is an adaptation of the proof of Theorem 3 to equality constraints. In addition, we need to prove the satisfaction of the additional constraints introduced by BTA. We focus on two cases.

We show that $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt}) \vDash_{p'}^{bt} (k^{\ell_1}\ [\![t^\ell]\!]^{Triv})^{\ell_0}$. For this purpose, given an arbitrary $\lambda^{\ell_k} x.e^{\ell_2} \in \widehat{C}'(\ell_1) = \widehat{\rho}'(k) = \widehat{C}(k)$ we must show that two equality contraints are satisfied. Similarly to the proof of Theorem 3, we make a case analysis on the introduction of k, using Lemma 9 and Lemma 10 to prove the satisfaction of the constraints.

We also need to show that $\widehat{C}'_{bt}(\ell_1) = \mathbf{D} \Rightarrow \widehat{C}'_{bt}(\ell) = \mathbf{D}$. Again, we make a case analysis on the introduction of k.

$$
\begin{array}{lcl}
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} n^{\ell} & \Longleftrightarrow & \mathit{true} \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} x^{\ell} & \Longleftrightarrow & \widehat{\rho}_{bt}(x) = \widehat{C}_{bt}(\ell) \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} (\lambda^{\ell'} x.e^{\ell_1})^{\ell} & \Longleftrightarrow & (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} e^{\ell_1} \wedge (\widehat{\rho}_{bt}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{bt}(\ell_1) = \widehat{\rho}_{bt}(x) = \mathbf{D}) \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} (t_0^{\ell_0}\ t_1^{\ell_1})^{\ell} & \Longleftrightarrow & (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} t_0^{\ell_0} \wedge (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} t_1^{\ell_1} \wedge (\widehat{C}_{bt}(\ell_0) = \mathbf{D} \Rightarrow \widehat{C}_{bt}(\ell_1) = \mathbf{D}) \wedge \\
 & & \forall (\lambda^{\ell'} x.e^{\ell_2}) \in \widehat{C}(\ell_0).\widehat{C}_{bt}(\ell_1) = \widehat{\rho}_{bt}(x) \wedge \widehat{C}_{bt}(\ell_2) = \widehat{C}_{bt}(\ell) \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} (op(t^{\ell}))^{\ell_0} & \Longleftrightarrow & (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} t^{\ell} \wedge \widehat{C}_{bt}(\ell) = \widehat{C}_{bt}(\ell_0) \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} (\mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \Longleftrightarrow & (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} t^{\ell} \wedge (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} e_0^{\ell_0} \wedge (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} e_1^{\ell_1} \wedge \\
 & & (\widehat{C}_{bt}(\ell_0) = \widehat{C}_{bt}(\ell_1) = \widehat{C}_{bt}(\ell_2)) \wedge (\widehat{C}_{bt}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{bt}(\ell_0) = \widehat{C}_{bt}(\ell_1) = \mathbf{D}) \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} (\mathbf{let}\ x = s^{\ell}\ \mathbf{in}\ e^{\ell_1})^{\ell_2} & \Longleftrightarrow & (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} s^{\ell} \wedge (\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} e^{\ell_1} \wedge \widehat{\rho}_{bt}(x) = \widehat{C}_{bt}(\ell) \wedge \widehat{C}_{bt}(\ell_1) = \widehat{C}_{bt}(\ell_2) \wedge \\
 & & \widehat{C}_{bt}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{bt}(\ell_1) = \mathbf{D} \\
 & & \\
(\widehat{C}_{bt},\widehat{\rho}_{bt}) \vDash_p^{bt} p & \Longleftrightarrow & (\forall x.x \text{ free in } p \Rightarrow \widehat{\rho}_{bt}(x) = \mathbf{D}) \wedge (p = e^{\ell} \Rightarrow \widehat{C}_{bt}(\ell) = \mathbf{D})
\end{array}
$$

Figure 8: Syntax-directed BTA

The top-level case is trivial. The case where k is introduced by the transformation of a function $(\lambda x.e_1^{\ell_3})^{\ell_4}$ implies that $\widehat{C}_{bt}(\ell_4) = \mathbf{D}$. Thus $\widehat{C}_{bt}(\ell_3) = \mathbf{D}$ and then $\widehat{C}'_{bt}(\ell) = \mathbf{D}$, since $t^{\ell} \in \mathcal{R}[\![e_1^{\ell_3}]\!]$. The same reasoning follows for the case where k comes from the transformation of an **if0** construct.

In the case of a transformation of an application $(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}$, we need to show that

$$
(\widehat{C}'_{bt},\widehat{\rho}'_{bt}) \vDash_{p'}^{bt} \mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in}\ x_0\ \lambda^{\ell''} x_1.(k^{\ell_4}\ x_1^{\ell_5})^{\ell_6}
$$

Again, most of the constraints are trivially satisfied, and it remains to show that $k^{\ell_4}\ x_1^{\ell_5}$ satisfies the constraints. We make a new case analysis where k was introduced, and apply again Lemma 10 to show that for an arbitrary $\lambda x.e_1$ in $\widehat{C}'(k)$, $\widehat{\rho}'_{bt}(x) = \widehat{C}'_{bt}(\ell_5)$. Also, assuming that $\widehat{C}'_{bt}(\ell_4) = \mathbf{D}$, by the same case analysis on the introduction of k, we obtain in all cases that $\widehat{C}_{bt}(\ell) = \mathbf{D}$ and thus $\widehat{C}'_{bt}(\ell_5) = \widehat{\rho}'_{bt}(x) = \widehat{C}_{bt}(\ell) = \mathbf{D}$. □

Theorem 4 and Lemma 8 show that we can transform 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 variables and trivial terms are preserved. In particular, this implies that no values are forced to be dynamic by the CPS transformation. It also implies that the static computations (applications, tests or base-type operations) in a direct-style program remain static as well in its CPS counterpart. We thus conclude that the same amount of specialization of the input program can be achieved after CPS transformation.

4.3 Reversing the transformation

Let us now show that it is not always possible to reverse the CPS transformation of binding times. There are cases when the least analysis of a CPS-transformed program produces strictly more static annotations than the least analysis of its direct-style counterpart. Here is a canonical example [11], where *inc* is the successor function. and the free variable z is considered to be dynamic:

$$
\mathbf{let}\ r = (\lambda^{\ell'} y.\mathbf{let}\ v = inc(z)\ \mathbf{in}\ 2)\ 1\ \mathbf{in}\ inc(r)
$$

In the least solution of the BTA on this term, even if the application of $\lambda^{\ell'} y.$ to 1 is classified as static, its result is classified as dynamic because of the dynamic header of the **let**-expression. Thus r is dynamic. Since the final increment depends on r, it is dynamic as well. In a realistic setting, simply discarding the dynamic computation $inc(z)$ might not be meaning-preserving since it can, for instance, yield an integer overflow at run-time.

The CPS counterpart of the canonical example above reads as follows:

$$
\begin{array}{l}
\lambda k.\mathbf{let}\ x_0 = (\lambda^{\ell'} y.\lambda k_1.\mathbf{let}\ v = inc(z)\ \mathbf{in}\ k_1\ 2)\ 1 \\
\quad \mathbf{in}\ x_0\ (\lambda r.\mathbf{let}\ r_1 = inc(r)\ \mathbf{in}\ k\ r_1)
\end{array}
$$

The continuation denoted by k_1 is static, and thus the application $k_1\ 2$ is performed statically (even if its result is dynamic). Thus, when k is static, r is static as well, and further computation based on r can be performed at specialization time.

Other binding-time improvements can be obtained when a dynamic test disables further computations based on its result. The canonical example is as follows (again, in a static context):

$$
\mathbf{let}\ v = \mathbf{if0}\ z\ 0\ 1\ \mathbf{in}\ inc(v)
$$

It is true that one benefits from such an improvement only by allowing code duplication. However, the code duplication happens at specialization time, not at BTA time. In contrast to Sabry and Felleisen's observation [28], the improvement in this case is thus not due to duplicating the analysis on the two branches.

4.4 Continuation-based partial evaluation

In the two examples above the binding-time improvements come from two constraints in the specification of the BTA (Figure 8): 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.

The binding-time constraint in the body of a **let** reflects the concern of which reduction can be safely performed by the specializer. In the context of the computational metalanguage [11], a named dynamic computation cannot be dis-

$$
\begin{array}{rcll}
[\![e^{\ell}]\!]^{Pgm} & = & (\lambda^{\ell_0} k.[\![e^{\ell}]\!]^{Exp} k)^{\ell_1} & \widehat{C}'_{bt}(\ell_1) = \mathbf{D} \quad \widehat{\rho}'_{bt}(k) = \mathbf{D} \\
\\
[\![n^{\ell}]\!]^{Triv} & = & n^{\ell} & \widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\ell) \\
[\![x^{\ell}]\!]^{Triv} & = & x^{\ell} & \widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\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}'_{bt}(\ell_2) = \widehat{C}_{bt}(\ell) \quad \widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\ell) \\
 & & & \widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(x) \quad \widehat{\rho}'_{bt}(k) = \widehat{C}_{bt}(\ell) \\
\\
[\![t^{\ell}]\!]^{Exp} k & = & (k^{\ell_1}\ [\![t^{\ell}]\!]^{Triv})^{\ell_0} & \widehat{C}'_{bt}(\ell_1) = \widehat{\rho}'_{bt}(k) \quad \widehat{C}'_{bt}(\ell_0) = \mathbf{D} \\
\\
[\![(t_0^{\ell_0}\ t_1^{\ell_1})^{\ell}]\!]^{Exp} k & = & (\mathbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_2} & \widehat{C}'_{bt}(\ell_5) = \widehat{\rho}'_{bt}(x) = \widehat{C}_{bt}(\ell) \\
 & & \mathbf{in}\ (x_0^{\ell_3}\ (\lambda^{\ell'} x.(k^{\ell_4}\ x^{\ell_5})^{\ell_6})^{\ell_7})^{\ell_8})^{\ell_9} & \widehat{C}'_{bt}(\ell_4) = \widehat{\rho}'_{bt}(k) \quad \widehat{C}'_{bt}(\ell_7) = \widehat{C}_{bt}(\ell_0) \\
 & & & \widehat{C}'_{bt}(\ell_3) = \widehat{\rho}'_{bt}(x_0) = \widehat{C}'_{bt}(\ell_2) = \widehat{C}_{bt}(\ell_0) \\
 & & & \widehat{C}'_{bt}(\ell_6) = \widehat{C}'_{bt}(\ell_8) = \widehat{C}'_{bt}(\ell_9) = \mathbf{D} \\
\\
[\![(op(t_0^{\ell_0}))^{\ell}]\!]^{Exp} k & = & (\mathbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell} & \widehat{C}'_{bt}(\ell_1) = \widehat{\rho}'_{bt}(k) \quad \widehat{C}'_{bt}(\ell_4) = \widehat{C}'_{bt}(\ell_3) = \mathbf{D} \\
 & & \mathbf{in}\ (k^{\ell_1}\ x^{\ell_2})^{\ell_3})^{\ell_4} & \widehat{C}'_{bt}(\ell_2) = \widehat{\rho}'_{bt}(x) = \widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\ell) \\
\\
[\![(\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}'_{bt}(\ell_3) = \mathbf{D} \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ \mathbf{in}\ e^{\ell_1})^{\ell}]\!]^{Exp} k & = & ((\lambda^{\ell'} x.[\![e^{\ell_1}]\!]^{Exp} k)^{\ell_2}\ [\![t_0^{\ell_0}]\!]^{Triv})^{\ell_3} & \widehat{C}'_{bt}(\ell_2) = \widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(x) \quad \widehat{C}'_{bt}(\ell_3) = \mathbf{D} \\
\\
\left[\!\!\left[(\mathbf{let}\ x = (t_0^{\ell_0}\ t_1^{\ell_1})^{\ell} \right.\right. & & & \widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(x) \quad \widehat{C}'_{bt}(\ell_6) = \widehat{C}_{bt}(\ell_0) \\
\left.\left. \mathbf{in}\ e^{\ell_2})^{\ell_3} \right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ x_0 = ([\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv})^{\ell_4} & \widehat{C}'_{bt}(\ell_7) = \widehat{C}'_{bt}(\ell_8) = \mathbf{D} \\
 & & \mathbf{in}\ (x_0^{\ell_5}\ (\lambda^{\ell'} x.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_6})^{\ell_7})^{\ell_8} & \widehat{C}'_{bt}(\ell_5) = \widehat{\rho}'_{bt}(x_0) = \widehat{C}'_{bt}(\ell_4) = \widehat{C}_{bt}(\ell_0) \\
\\
\left[\!\!\left[(\mathbf{let}\ x = (op(t_0^{\ell_0}))^{\ell} \right.\right. & & & \widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(x) \quad \widehat{C}'_{bt}(\ell) = \widehat{C}_{bt}(\ell) \\
\left.\left. \mathbf{in}\ e^{\ell_2})^{\ell_3} \right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ x = (op([\![t_0^{\ell_0}]\!]^{Triv}))^{\ell} & \widehat{C}'_{bt}(\ell_4) = \mathbf{D} \\
 & & \mathbf{in}\ [\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4} & \\
\\
\left[\!\!\left[(\mathbf{let}\ x = (\mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \right.\right. & & & \widehat{\rho}'_{bt}(k_1) = \widehat{C}'_{bt}(\ell_5) = \widehat{C}_{bt}(\ell_2) \\
\left.\left. \mathbf{in}\ e^{\ell_3})^{\ell_4} \right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ k_1 = (\lambda^{\ell'} x.[\![e^{\ell_3}]\!]^{Exp} k)^{\ell_5} & \widehat{C}'_{bt}(\ell_6) = \widehat{C}'_{bt}(\ell_7) = \mathbf{D} \quad \widehat{\rho}'_{bt}(x) = \widehat{\rho}_{bt}(\ell_2) \\
 & & \mathbf{in}\ (\mathbf{if0}\ [\![t^{\ell}]\!]^{Triv} [\![e_0^{\ell_0}]\!]^{Exp} k_1 & \\
 & & \qquad [\![e_1^{\ell_1}]\!]^{Exp} k_1)^{\ell_6})^{\ell_7} &
\end{array}
$$

Figure 9: Transformation of binding times from direct style to CPS

carded due to possible computational effects. Similarly, the constraint over the branches of an **if0** is introduced because one cannot decide statically which conditional branch should be selected.

The above-mentioned constraint on the body of a **let** can be relaxed if one uses a *continuation-based program specializer* [2, 11, 18]. The constraint connecting the branches of a test with the test itself can be relaxed as well if one allows the same continuation-based specializer to lift the test above the context, either by context duplication or by **let**-insertion. Given such a specializer, we can show that enhancing the BTA by relaxing the two special constraints voids the impact of CPS on the global result.

More formally, we consider the BTA of Figure 8, without the two constraints mentioned just above. Naming $\models^+_p$ the new satisfaction predicate, we modify the analysis of the **let** and **if0** constructs as specified in Figure 10.

Using the same proof technique as in Section 3, we can formally show that the CPS transformation has no effect on the enhanced analysis, i.e., it entails 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.

More formally, given the program p and a solution $(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \models^+_p p$ of the enhanced analysis, we can show that $\mathcal{C}^{bt}_p(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \models^+_{p'} p'$. Also, given an arbitrary solution $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt}) \models^+_{p'} p'$ we can as well define its direct-style binding times $(\widehat{C}_{bt}, \widehat{\rho}_{bt})$ such that $(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \models^+_p p$. The reverse binding-times transformation $\mathcal{D}^{bt}_p$ (omitted here) is essentially the same as the reverse flow transformation of Section 3.3: for each term we just extract the binding times its CPS counterpart.

We connect the binding times in direct style and in CPS as obtained by the enhanced analysis as follows:

THEOREM 5. *Given the direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \models^+_p p$ be the least solution of the enhanced BTA on p and let $(\widehat{C}'_{bt}, \widehat{\rho}'_{bt}) \models^+_{p'} p'$ be the least solution of the same enhanced BTA on p'. Then for all variables x in p, $\widehat{\rho}'(x) = \widehat{\rho}'_{bt}(x)$ and for all trivial terms t^{ℓ} in p, $\widehat{C}_{bt}(\ell) = \widehat{C}'_{bt}(\ell)$.*

$$(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ (\textbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} \iff (\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ t^\ell \wedge (\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ e_0^{\ell_0} \wedge (\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ e_1^{\ell_1} \wedge \widehat{C}_{bt}(\ell_0) = \widehat{C}_{bt}(\ell_1) = \widehat{C}_{bt}(\ell_2)$$
$$(\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ (\textbf{let}\ x = s^\ell\ \textbf{in}\ e^{\ell_1})^{\ell_2} \iff (\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ s^\ell \wedge (\widehat{C}_{bt}, \widehat{\rho}_{bt}) \vDash_p^+ e^{\ell_1} \wedge \widehat{\rho}_{bt}(x) = \widehat{C}_{bt}(\ell) \wedge \widehat{C}_{bt}(\ell_1) = \widehat{C}_{bt}(\ell_2)$$

Figure 10: Binding-time constraints for continuation-based partial evaluation

We thus conclude that the CPS transformation has no effect on the amount of specialization that can be performed when using continuation-based partial-evaluation.

5. RELATED WORK

5.1 Program analysis in general

With the exception of Nielson's early work on data-flow analysis [19],[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 [28].

Sabry and Felleisen conclude that CPS can improve results by duplication of the analysis over conditionals and can worsen results by confusing the return points of function calls. About their first point, our analyses (0-CFA and the extended BTA) do not duplicate the analysis over conditionals and thus do not yield more precise results on CPS programs. As for confusing the return points, Sabry and Felleisen's analysis introduced a limited form of "context dependency" where it makes a difference which application of a function is analyzed first. Because of this aspect, confusing the return points has an impact on their analysis. Our 0-CFA and modified BTA do not share this property. Already in direct style, the constraint-based approach (in the monovariant case) propagates the result of a function to all the places where the function is applied. Thus, the CPS transformation preserves the results of the two analyses.

Recently, Palsberg and Wand have conducted a similar study for 0-CFA [27], supporting Sabry and Felleisen's conclusion that the extra precision enabled by the CPS transformation is due to the duplication of the analysis. Palsberg and Wand's CPS transformation of flow types was developed independently, but is otherwise comparable to ours in Section 3. They implicitly share our concern about syntactic accidents, in that they mention that least solutions may or may not be preserved by administrative reductions of CPS-transformed programs.

5.2 Binding-time analysis and the CPS transformation

Binding-time improvements have always been customary for users of binding-time analysis [16, 20]. One of them amounts to considering source programs in CPS [4, 6], 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 conditionals), thereby providing a binding-time improvement. To this end, the CPS transformation itself is continuation-based [7], which paved the way to continuation-based partial evaluation [2, 18].

Hatcliff and Danvy have characterized the full effect of continuation-based partial evaluation as online let flattening in Moggi's computational meta-language [11]. This characterization justifies why offline let flattening is also, partially, a binding-time improvement [13]. In any case, offline let flattening is now known to be part of the CPS transformation [10].

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

6. CONCLUSION AND ISSUES

Observing that program analyses are vulnerable 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. Through a systematic construction of the CPS counterpart of flow information, similar to Palsberg and Wand's [27] 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 modified let and case rules, BTA is insensitive to the CPS transformation.

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 similar let and case rules. 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 and case rules, 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 vulnerable to syntactic accidents, we need to better understand the program-analysis perspective over syntactic landscapes. Two key questions arise, that

[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]Since we carried out the present study, though, we have become aware that besides binding-time improvements, the issue of syntactic accidents is not new: for Hanne Riis Nielson, it is folklore in the program-analysis community (personal communication, March 2000), and for Fritz Henglein (personal communication, March 2000), it has long been necessary to analyze invariance properties of program analysis with respect to certain classes of transformations.

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 reliable program analyses, possibly with some kind of subject reduction property or with some kind of intermediate language for program analysis.

7. ACKNOWLEDGMENTS

We are grateful to Torben Amtoft, Andrzej Filinski, Bernd Grobauer, Dan Hernest, Niels O. Jensen, Julia L. Lawall, Lasse R. Nielsen, Morten Rhiger, and Zhe Yang for comments. Thanks are also due to the anonymous referees.

8. 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 Susan L. Graham, editor, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 493–501, Charleston, South Carolina, January 1993. ACM Press.

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

[7] 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.

[8] J. Muylaert Filho and Geoffrey Burn. Continuation passing transformation and abstract interpretation. In G. Burn, S. Gay, and M. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 247–259. Springer-Verlag, 1993.

[9] 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.

[10] 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.

[11] 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.

[12] Nevin Heintze. Set-based program analysis of ML programs. In Talcott [30], pages 306–317.

[13] 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.

[14] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In Peter Lee, editor, *Proceedings of the Twenty-Second Annual ACM Symposium on Principles of Programming Languages*, pages 393–407, San Francisco, California, January 1995. ACM Press.

[15] 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.

[16] 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`.

[17] Ulrik Jørring and William L. Scherlis. Compilers and staging transformations. In Mark Scott Johnson and Ravi Sethi, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pages 86–96, St. Petersburg, Florida, January 1986. ACM Press.

[18] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Talcott [30].

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

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

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.