Basic Research in Computer Science

BRICS RS-00-15 Damian & Danvy: On the Impact of the CPS Transformation

Syntactic Accidents in Program Analysis: On the Impact of the CPS Transformation

Daniel Damian
Olivier Danvy

BRICS Report Series **RS-00-15**

ISSN 0909-0878 **June 2000**

**See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/00/15/`

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 ‡

June 2000

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 equivalent 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 effect 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 effect of syntactic change.

*Extended version of an article to appear in the Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP'00), September 18–20, 2000, Montreal, Canada.

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

‡Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.
E-mail: {`damian,danvy`}`@brics.dk`
Home pages: `http://www.brics.dk/~{damian,danvy}`

Contents

List of Figures

1 Introduction

1.1 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 re-association: 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 effect 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]. Therefore, the output of binding-time analyses has been intensively studied, especially in connection with syntactic changes in their input. As a result, "binding-time improvements" have been developed to milk out extra precision from binding-time analyses [16, Chapter 12], to the point that partial-evaluation users are encouraged to write programs in a very specific style [15]. That said, binding-time-improvements are not specific to offline partial evaluation—they are also routine in staging transformations [17] and 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 of source programs into continuation-passing style (CPS) [3, 32], people have wondered whether CPS may help program analysis in general. Nielson's early work on data-flow analysis [21] suggests so, since it shows that for a non-distributive analysis, a continuation semantics yields more precise results than a direct semantics. 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 [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 a direct-style program and analyzing its CPS counterpart yields incomparable results. They showed that CPS might increase precision by duplicating continuations, and also that CPS might decrease precision by confusing return points. These results are essentially confirmed by Palsberg and Wand's recent CPS transformation of flow information [29]. At any rate, except for continuation-based partial evaluation [10], there seems to have been no 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 program 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 effect on 0-CFA, i.e., analyzing a direct-style program and analyzing its CPS counterpart 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 effect 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.)

In the next section, we point out how the dependency induced by let-expressions leads to a loss of precision.

1.2 A loophole: the let rule

A binding-time analysis classifies a let expression to be dynamic if its header is dynamic, regardless of the binding time of its body. (Similarly, if a let header is classified to be of high security, the whole let expression is also classified to be of high security, regardless of the security level of its body.) The body of the following λ-abstraction is thus classified as dynamic if e is dynamic:

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

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) 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

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

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 function in continuation-passing style (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 [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.

Let us also point out that the CPS transformation leads to binding-time improvements for conditional expressions. 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 by the CPS transformation. This point is developed further in Section 4.4.

1.3 Overview

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, Section 4 addresses binding-time analysis, 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 follows a 'monadic style', i.e., it patterns the call-by-value encoding of a PCF-like language into Moggi's computational meta-language after let-flattening [9]. Terms in *Triv* represent values, while those in *Step* and *Exp* represent computations (i.e., value returns, applications, primitive operations and conditionals).

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

$$
\begin{array}{rcl}
p \in \mathit{Pgm} & ::= & e^{\ell} \\
e \in \mathit{Exp} & ::= & t \mid \textbf{let } x = s \textbf{ in } e^{\ell_1} \\
s \in \mathit{Step} & ::= & t^{\ell} \mid t_0^{\ell_0}\ t_1^{\ell_1} \mid \mathit{op}(t^{\ell}) \mid \textbf{if0 } t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} \\
t \in \mathit{Triv} & ::= & n \mid x \mid \lambda^{\pi} x.e^{\ell} \\
x \in \mathit{Ide} & & \text{(identifiers)} \\
n \in \mathit{Int} & & \text{(integers)} \\
\ell \in \mathit{Lab} & & \text{(term labels)} \\
\pi \in \mathit{Lam} & & (\lambda\text{-abstraction labels}) \\
\mathit{op} \in & & \text{an unspecified set of base-type operators}
\end{array}
$$

Figure 1: The direct-style language

$$
\begin{array}{rcl}
[\![e]\!]^{\mathit{Pgm}} & = & \lambda k.[\![e]\!]^{\mathit{Exp}} k \\
 & & \text{where } k \text{ is fresh} \\
 & & \\
[\![n]\!]^{\mathit{Triv}} & = & n \\
[\![x]\!]^{\mathit{Triv}} & = & x \\
[\![\lambda x.e]\!]^{\mathit{Triv}} & = & \lambda x.\lambda k.[\![e]\!]^{\mathit{Exp}} k \\
 & & \text{where } k \text{ is fresh} \\
 & & \\
[\![t]\!]^{\mathit{Exp}} k & = & k\ [\![t]\!]^{\mathit{Triv}} \\
[\![\textbf{let } x = t \textbf{ in } e]\!]^{\mathit{Exp}} k & = & \textbf{let } x = [\![t]\!]^{\mathit{Triv}} \textbf{ in } [\![e]\!]^{\mathit{Exp}} k \\
[\![\textbf{let } x = t_0\ t_1 \textbf{ in } e]\!]^{\mathit{Exp}} k & = & [\![t_0]\!]^{\mathit{Triv}}\ [\![t_1]\!]^{\mathit{Triv}}\ \lambda x.[\![e]\!]^{\mathit{Exp}} k \\
[\![\textbf{let } x = \mathit{op}(t) \textbf{ in } e]\!]^{\mathit{Exp}} k & = & \widetilde{\mathit{op}}\ [\![t]\!]^{\mathit{Triv}}\ \lambda x.[\![e]\!]^{\mathit{Exp}} k \\
[\![\textbf{let } x = \textbf{if0 } t\ e_0\ e_1 \textbf{ in } e]\!]^{\mathit{Exp}} k & = & \textbf{let } k_1 = \lambda x.[\![e]\!]^{\mathit{Exp}} k \\
 & & \textbf{in if0 } [\![t]\!]^{\mathit{Triv}}\ ([\![e_0]\!]^{\mathit{Exp}} k_1)\ ([\![e_1]\!]^{\mathit{Exp}} k_1) \\
 & & \text{where } k_1 \text{ is fresh}
\end{array}
$$

Figure 2: The CPS transformation

The CPS transformation of direct-style terms (given in Figure 2) yields terms in a CPS language.[2] CPS is a restriction of direct style. Thus, since we want to use the same program analysis, we embed the CPS language into the original direct-style language. For example, applications are transformed into let-expressions that name partially applied CPS λ-abstractions and intermediate computational steps. Figure 3 displays the corresponding CPS transformation and embedding.[3] (We have omitted the labels, because they only matter in the following sections. Suffice it to say that we label each CPS trivial term with the same label as its direct-style counterpart.)

[2] In Figure 2, $\widetilde{\mathit{op}}$ is the CPS counterpart of op, to ensure evaluation-order independence [30].

[3] In Figure 3, we use op instead of $\widetilde{\mathit{op}}$ since the direct-style language is call-by-value.

$$
\begin{aligned}
[\![e]\!]^{Pgm} &= \lambda k.[\![e]\!]^{Exp}\,k \\
&\quad \text{where } k \text{ is fresh} \\
\\
[\![n]\!]^{Triv} &= n \\
[\![x]\!]^{Triv} &= x \\
[\![\lambda x.e]\!]^{Triv} &= \lambda x.\lambda k.[\![e]\!]^{Exp}\,k \\
&\quad \text{where } k \text{ is fresh} \\
\\
[\![t]\!]^{Exp}\,k &= \textbf{let } x = k\ [\![t]\!]^{Triv} \textbf{ in } x \\
&\quad \text{where } x \text{ is fresh} \\
\\
[\![\textbf{let } x = t \textbf{ in } e]\!]^{Exp}\,k &= \textbf{let } x = [\![t]\!]^{Triv} \textbf{ in } [\![e]\!]^{Exp}\,k \\
\\
[\![\textbf{let } x = t_0\ t_1 \textbf{ in } e]\!]^{Exp}\,k &= \textbf{let } x_1 = [\![t_0]\!]^{Triv}\ [\![t_1]\!]^{Triv} \\
&\quad \textbf{in let } x_2 = x_1\ \lambda x.[\![e]\!]^{Exp}\,k \textbf{ in } x_2 \\
&\quad \text{where } x_1 \text{ and } x_2 \text{ are fresh} \\
\\
[\![\textbf{let } x = op(t) \textbf{ in } e]\!]^{Exp}\,k &= \textbf{let } x = op([\![t]\!]^{Triv}) \textbf{ in } [\![e]\!]^{Exp}\,k \\
\\
[\![\textbf{let } x = \textbf{if0 } t\ e_0\ e_1 \textbf{ in } e]\!]^{Exp}\,k &= \textbf{let } k_1 = \lambda x.[\![e]\!]^{Exp}\,k \textbf{ in} \\
&\quad \textbf{let } x_1 = \textbf{if0 } [\![t]\!]^{Triv}\ ([\![e_0]\!]^{Exp}\,k_1)\ ([\![e_1]\!]^{Exp}\,k_1) \textbf{ in } x_1 \\
&\quad \text{where } k_1 \text{ and } x_1 \text{ are fresh}
\end{aligned}
$$

Figure 3: CPS transformation and embedding into the direct-style language

3 Control-flow analysis

We consider a constraint-based, monovariant control-flow analysis (0-CFA) over direct-style programs [8, 14, 23, 26]. The constraint-based analysis is known to be equivalent to other flow analyses, based on different methods such as set-based analysis [11] and type inference [27]. For uniformity, we adopt the same definition and notation as in Nielson, Nielson and Hankin's recent textbook on program analysis [24].

The flow information computed by the analysis is a pair consisting of an abstract cache $\widehat{C}_{\mathrm{cf}}$ which maps terms to abstract values and an abstract environment $\widehat{\rho}_{\mathrm{cf}}$ which maps variables to abstract values. Abstract values are sets of labels of λ-abstractions to which a term can be reduced and a variable can be bound. The constraint-based control-flow analysis is specified as a relation $\models_{\mathrm{cf}}$ on caches, environments and terms. Given a term e, $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models_{\mathrm{cf}} e$ means that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ is a result of the control-flow analysis of e.[4]

In this work we use the syntax-directed variant of the analysis [24, Chapter 3], and we restrict its analysis relation to a relation $\models^{p}_{\mathrm{cf}}$ associated to each program being analyzed. Given a closed direct-style program p, the functionality of the associated relation $\models^{p}_{\mathrm{cf}}$ is defined in Figure 4. The analysis relation is defined in Figure 5 by induction over the syntax of the program.

[4] In the notation of Nielson, Nielson, and Hankin [24], $\models_{\mathrm{cf}}$ is simply $\models$.

$$
\begin{array}{rl}
Lam^p & \text{The set of } \lambda\text{-abstraction labels in } p \\
Var^p & \text{The set of identifiers in } p \\
Lab^p & \text{The set of term labels in } p
\end{array}
$$

$$
\begin{array}{rrll}
 & Val^p_{\mathrm{cf}} & = \mathcal{P}(Lam^p) & \text{Abstract values} \\
\widehat{C}_{\mathrm{cf}} \in & Cache^p_{\mathrm{cf}} & = Lab^p \to Val^p_{\mathrm{cf}} & \text{Abstract cache} \\
\widehat{\rho}_{\mathrm{cf}} \in & Env^p_{\mathrm{cf}} & = Var^p \to Val^p_{\mathrm{cf}} & \text{Abstract environment}
\end{array}
$$

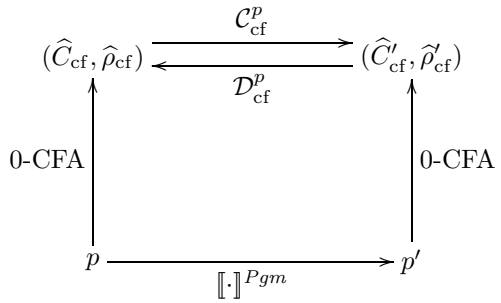
$$
\vDash^p_{\mathrm{cf}} \subseteq (Cache^p_{\mathrm{cf}} \times Env^p_{\mathrm{cf}}) \times Lab^p
$$

Figure 4: 0-CFA relation for a program p

Any solution $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ accepted by the relation $\vDash^p_{\mathrm{cf}}$ (i.e., such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \vDash^p_{\mathrm{cf}} p$ holds) is a conservative approximation of the exact flow information [24, Chapter 3]. Furthermore, the analysis relation $\vDash^p_{\mathrm{cf}}$ has a model-intersection property, i.e., the set of solutions accepted by $\vDash^p_{\mathrm{cf}}$ 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 p and an arbitrary solution of its associated analysis $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$, we construct a solution $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ of the analysis associated to the CPS counterpart of the program, p'. We then ensure that the construction $\mathcal{C}^p_{\mathrm{cf}}$ builds a valid solution (Section 3.2). We present a converse transformation, $\mathcal{D}^p_{\mathrm{cf}}$ (Section 3.3), which we also prove to be correct (Section 3.4).

Graphically:



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 of an arbitrary program is transformed into the least solution of the analysis of the 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.

$$
\begin{array}{lcl}
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} n^{\ell} & \iff & \mathit{true} \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} x^{\ell} & \iff & \widehat{\rho}_{\mathrm{cf}}(x) \subseteq \widehat{C}_{\mathrm{cf}}(\ell) \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} (\lambda^{\pi} x.e^{\ell_1})^{\ell} & \iff & \{\pi\} \subseteq \widehat{C}_{\mathrm{cf}}(\ell) \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e^{\ell_1} \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} (\mathbf{let}\ x = t^{\ell}\ \mathbf{in}\ e^{\ell_1})^{\ell_2} & \iff & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} t^{\ell} \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e^{\ell_1} \wedge \\
 & & \widehat{C}_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}_{\mathrm{cf}}(x) \wedge \widehat{C}_{\mathrm{cf}}(\ell_1) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_2) \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} (\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^{\ell_2})^{\ell_3} & \iff & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} t_0^{\ell_0} \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} t_1^{\ell_1} \wedge \\
 & & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e^{\ell_2} \wedge \widehat{C}_{\mathrm{cf}}(\ell_2) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_3) \wedge \\
 & & \forall (\lambda^{\pi} y.e_1^{\ell}) \in \widehat{C}_{\mathrm{cf}}(\ell_0). \\
 & & \quad (\widehat{C}_{\mathrm{cf}}(\ell_1) \subseteq \widehat{\rho}_{\mathrm{cf}}(y) \wedge \widehat{C}_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}_{\mathrm{cf}}(x)) \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} (\mathbf{let}\ x = op(t^{\ell})\ \mathbf{in}\ e^{\ell_1})^{\ell_2} & \iff & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} t^{\ell} \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e^{\ell_1} \wedge \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_1) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_2) \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} (\mathbf{let}\ x = \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} & \iff & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} t^{\ell} \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e_0^{\ell_0} \wedge \\
\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e_1^{\ell_1} \wedge (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} e^{\ell_2} \wedge \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_0) \subseteq \widehat{\rho}_{\mathrm{cf}}(x) \wedge \widehat{C}_{\mathrm{cf}}(\ell_1) \subseteq \widehat{\rho}_{\mathrm{cf}}(x) \wedge \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_2) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_3)
\end{array}
$$

Figure 5: Syntax-directed 0-CFA

3.1 CPS transformation of control flow

Given a solution $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ of the analysis of a program p (i.e., a cache-environment pair such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} p$ holds), we now construct in linear time a solution $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ of the analysis of $p' = [\![p]\!]^{Pgm}$, the CPS counterpart of p (i.e., such that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models^{p'}_{\mathrm{cf}} p'$ holds). By analogy, we refer to the construction of $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ out of $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ as the CPS transformation of $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ into $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$.

As mentioned in Section 2, we have designed the CPS transformation on labeled terms so that it preserves the labels of each trivial term. In addition, each direct-style λ-abstraction is annotated with the same label as its CPS counterpart. As a consequence, the abstract values in direct style are included into the abstract values in CPS, i.e., $\mathit{Lam}^{p} \subseteq \mathit{Lam}^{p'}$ and $\mathit{Val}^{p}_{\mathrm{cf}} \subseteq \mathit{Val}^{p'}_{\mathrm{cf}}$. The CPS transformation preserves all the variables defined in the original direct-style program. Therefore $\mathit{Var}^{p} \subseteq \mathit{Var}^{p'}$. 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 identical to the information found 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 $\{\lambda^{\pi^i} x_i.\lambda^{\pi^i_1} k_i.e_i | 1 \leq i \leq n\}$, for some n, then $\gamma(A) = \{\pi^i_1 | 1 \leq i \leq n\}$.

- ξ assigns flow information to each continuation identifier k introduced by the CPS transformation of a λ-abstraction from p. This information can be obtained

$$
\begin{array}{rcl}
[\![e^\ell]\!]^{Pgm} & = & (\lambda^\pi k.[\![e^\ell]\!]^{Exp}k)^{\ell_0} \qquad \widehat{C}'_{\mathrm{cf}}(\ell_0) = \{\pi\} \quad \widehat{\rho}'_{\mathrm{cf}}(k) = \emptyset \\
\\
[\![n^\ell]\!]^{Triv} & = & n^\ell \qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \\
[\![x^\ell]\!]^{Triv} & = & x^\ell \qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp}k)^{\ell_2})^\ell \\
& & \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \{\pi_1\} \\
& & \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \quad \widehat{\rho}'_{\mathrm{cf}}(k) = \xi(k) \\
\\
[\![t^\ell]\!]^{Exp}k & = & (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2} \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_0) = \widehat{\rho}'_{\mathrm{cf}}(k) \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_2) = \widehat{C}'_{\mathrm{cf}}(\ell_1) = \widehat{\rho}_{\mathrm{cf}}(x) = \emptyset \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ \mathbf{in}\ e^\ell)^{\ell_1}]\!]^{Exp}k & = & (\mathbf{let}\ x = [\![t_0^{\ell_0}]\!]^{Triv}\ \mathbf{in}\ [\![e^\ell]\!]^{Exp}k)^{\ell_2} \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_2) = \emptyset \quad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^\ell)^{\ell_2}]\!]^{Exp}k & = & (\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in} \\
& & \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{Exp}k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_3) = \widehat{\rho}'_{\mathrm{cf}}(x_0) = \gamma(\widehat{C}_{\mathrm{cf}}(\ell_0)) \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_4) = \{\pi\} \quad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_7) = \widehat{C}'_{\mathrm{cf}}(\ell_6) = \widehat{C}'_{\mathrm{cf}}(\ell_5) = \widehat{\rho}'_{\mathrm{cf}}(x_1) = \emptyset \\
\\
[\![(\mathbf{let}\ x = op(t^\ell)\ \mathbf{in}\ e^{\ell_0})^{\ell_1}]\!]^{Exp}k & = & (\mathbf{let}\ x = op([\![t^\ell]\!]^{Triv})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{Exp}k)^{\ell_2} \\
& & \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \quad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \emptyset \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} \\ \ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ k_1 = (\lambda^\pi x.[\![e^{\ell_2}]\!]^{Exp}k)^{\ell_4}\ \mathbf{in} \\
& & \ (\mathbf{let}\ x_1 = \mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ ([\![e_0^{\ell_0}]\!]^{Exp}k_1)\ ([\![e_1^{\ell_1}]\!]^{Exp}k_1) \\
& & \ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
& & \widehat{\rho}'_{\mathrm{cf}}(k_1) = \widehat{C}'_{\mathrm{cf}}(\ell_4) = \{\pi\} \quad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \\
& & \widehat{C}'_{\mathrm{cf}}(\ell_7) = \widehat{C}'_{\mathrm{cf}}(\ell_6) = \widehat{C}'_{\mathrm{cf}}(\ell_5) = \widehat{\rho}'_{\mathrm{cf}}(x_1) = \emptyset
\end{array}
$$

Figure 6: Flow transformation from direct style to CPS

from the direct-style flow information, since we can *syntactically* identify the continuation of the CPS counterpart of any direct-style application.

Given p, $\widehat{C}_{\mathrm{cf}}$, $\widehat{\rho}_{\mathrm{cf}}$, and a continuation identifier k introduced by the transformation of a λ-abstraction from p:

$$[\![\lambda^{\pi_1} x.e]\!]^{Triv} = \lambda x.\lambda k.[\![e]\!]^{Exp}k$$

we gather in $\xi(k)$ all the continuations that are passed at the program points where $\lambda^{\pi_1} x.e$ can be applied. Formally, $\xi(k)$ is defined as the set of all labels π such that in the CPS transformation of p into p' there exists a transformation

step

$$\left[\!\left[\begin{array}{l}\textbf{let } x = t_0^{\ell_0}\ t_1\\ \textbf{in } e\end{array}\right]\!\right]^{Exp} \quad k_1 = \begin{array}{l}\textbf{let } x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1]\!]^{Triv}\\ \textbf{in let } x_1 = x_0\ \lambda^{\pi} x.[\![e]\!]^{Exp} k_1\\ \textbf{in } x_1\end{array}$$

such that $\pi_1 \in \widehat{C}_{\mathrm{cf}}(\ell_0)$.

Using γ and ξ, we define $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ inductively, following Figure 6. In the right part, for each CPS-transformation step, we assign flow values into $\widehat{C}'_{\mathrm{cf}}$ and $\widehat{\rho}'_{\mathrm{cf}}$ using previously defined values.

In fact, the construction of flow information defines a function

$$\mathcal{C}^p_{\mathrm{cf}} : (Cache^p_{\mathrm{cf}} \times Env^p_{\mathrm{cf}}) \to (Cache^{p'}_{\mathrm{cf}} \times Env^{p'}_{\mathrm{cf}}).$$

It is easy to show that $\mathcal{C}^p_{\mathrm{cf}}$ is monotone.

3.2 Correctness of the transformation

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

Theorem 1. *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ be a solution of the 0-CFA of p (i.e., such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \vDash^p_{\mathrm{cf}} p$ holds) and let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = \mathcal{C}^p_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$. Then $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \vDash^{p'}_{\mathrm{cf}} p'$ holds.*

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, $\widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x)$; for all trivial terms t^{ℓ} in p, $\widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell)$; and for all expressions e^{ℓ} in p', $\widehat{C}'_{\mathrm{cf}}(\ell) = \emptyset$.*

For an arbitrary expression, we define the notion of return label to capture the return point 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 return label $\mathcal{R}[\![e^{\ell}]\!]$ of e^{ℓ} by structural induction as follows:*

$$\begin{array}{rcl}\mathcal{R}[\![t^{\ell}]\!] & = & \ell\\ \mathcal{R}[\![(\textbf{let } x = s \textbf{ in } e^{\ell_1})^{\ell}]\!] & = & \mathcal{R}[\![e^{\ell_1}]\!]\end{array}$$

Lemma 2. *Let e^{ℓ} be an arbitrary subexpression of p. Then $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell}]\!]) \subseteq \widehat{C}_{\mathrm{cf}}(\ell)$.*

A return label identifies the point where a continuation is called in the CPS-transformed program. Return labels thus provide a syntactic connection between the points where flow information is collected in direct style and the points where flow information is sent to continuations in CPS.

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

$$[\![\lambda^{\pi_1} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\pi_1} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

Then, for each $\lambda^{\pi} x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$, $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$.

$$
\begin{array}{rcl}
[\![e^\ell]\!]^{Pgm} & = & (\lambda^\pi k.[\![e^\ell]\!]^{Exp} k)^{\ell_0} \\
[\![n^\ell]\!]^{Triv} & = & n^\ell \qquad\qquad \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap Lam^p \\
[\![x^\ell]\!]^{Triv} & = & x^\ell \qquad\qquad \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap Lam^p \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell \\
 & & \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap Lam^p \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p \\
\\
[\![t^\ell]\!]^{Exp} k & = & (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2} \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ \mathbf{in}\ e^\ell)^{\ell_1}]\!]^{Exp} k & = & (\mathbf{let}\ x = [\![t_0^{\ell_0}]\!]^{Triv}\ \mathbf{in}\ [\![e^\ell]\!]^{Exp} k)^{\ell_2} \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_1) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^\ell)^{\ell_2}]\!]^{Exp} k & = & (\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in} \\
 & & \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{Exp} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_2) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p \\
\\
[\![(\mathbf{let}\ x = op(t^\ell)\ \mathbf{in}\ e^{\ell_0})^{\ell_1}]\!]^{Exp} k & = & (\mathbf{let}\ x = op([\![t^\ell]\!]^{Triv})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2} \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_1) = \widehat{C}_{\mathrm{cf}}(\ell_0) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} \\ \ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ k_1 = (\lambda^\pi x.[\![e^{\ell_2}]\!]^{Exp} k)^{\ell_4}\ \mathbf{in} \\
 & & \ (\mathbf{let}\ x_1 = \mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ ([\![e_0^{\ell_0}]\!]^{Exp} k_1)\ ([\![e_1^{\ell_1}]\!]^{Exp} k_1) \\
 & & \ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
 & & \widehat{C}_{\mathrm{cf}}(\ell_3) = \widehat{C}_{\mathrm{cf}}(\ell_2) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p
\end{array}
$$

Figure 7: Flow transformation from CPS to direct style

By the definition of ξ, it is immediate to show that $\widehat{C}_{\mathrm{cf}}(\ell_0) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$. From Lemma 2, $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_0)$.

The proof of Theorem 1 is presented in Appendix A.

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 of 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 a direct-style program p and its CPS counterpart p', and given $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ a valid solution of the analysis on p', we recover in linear time a valid solution $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ of the analysis of p.

Recovering a direct-style solution is straightforward. For variables and trivial terms in p, we are only "filtering out" the labels of continuations from the results of the analysis of p'. We define the direct-style solution by induction on the CPS transformation, following Figure 7. In the right part, for each CPS-transformation step, we assign flow values into $\widehat{C}_{\mathrm{cf}}$ and $\widehat{\rho}_{\mathrm{cf}}$. The left parts of Figures 6 and 7 are identical.

We can show that Figure 7 defines another function

$$\mathcal{D}^p_{\mathrm{cf}} : (Cache^{p'}_{\mathrm{cf}} \times Env^{p'}_{\mathrm{cf}}) \to (Cache^{p}_{\mathrm{cf}} \times Env^{p}_{\mathrm{cf}}).$$

It is also relatively easy to show that, like $\mathcal{C}^p_{\mathrm{cf}}$ in Section 3.2, $\mathcal{D}^p_{\mathrm{cf}}$ is monotone.

3.4 Correctness of the reverse transformation

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

Theorem 2. *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ be a solution of the 0-CFA of p' (i.e., such that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \vDash^{p'}_{\mathrm{cf}} p'$ holds) and let $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = \mathcal{D}^p_{\mathrm{cf}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$. Then $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \vDash^{p}_{\mathrm{cf}} p$ holds.*

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

Lemma 4. *For all $x \in Var^p$, $\widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap Lam^p$; and for all trivial terms t^ℓ in p, $\widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap Lam^p$.*

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

Lemma 5. *Let e^ℓ be an expression in p. Then $\widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^\ell]\!])$.*

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:*

$$[\![\lambda^{\pi_1} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\pi_1} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

Then, for each $\lambda^\pi x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$, $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$.

The proof of Theorem 2 is presented in Appendix A.

3.5 Equivalence of flow

Let p be an arbitrary direct-style program and $p' = [\![p]\!]^{Pgm}$ its CPS counterpart. It is a matter of tedious calculations to prove the following lemma:

Lemma 7. *Given $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ a solution of the 0-CFA of p (i.e., such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \vDash^{p}_{\mathrm{cf}} p$ holds), $\mathcal{D}^p_{\mathrm{cf}}(\mathcal{C}^p_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})) \subseteq (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$. Given $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ a solution of the 0-CFA of p', (i.e., such that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \vDash^{p'}_{\mathrm{cf}} p'$ holds), $\mathcal{C}^p_{\mathrm{cf}}(\mathcal{D}^p_{\mathrm{cf}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})) \subseteq (\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$.*

From these two properties the following main theorem follows directly.

Theorem 3 (Equivalence of flow). *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ be the least solution of the 0-CFA of p and let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ be the least solution of the 0-CFA of p'. Then $\mathcal{C}^p_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = (\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ and $\mathcal{D}^p_{\mathrm{cf}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$.*

$$
\begin{array}{rcll}
Val_{\mathrm{bt}} & = & \{\mathbf{S}, \mathbf{D}\} & \text{Abstract values} \\
\widehat{C}_{\mathrm{bt}} \in Cache^p_{\mathrm{bt}} & = & Lab^p \to Val_{\mathrm{bt}} & \text{Abstract cache} \\
\widehat{\rho}_{\mathrm{bt}} \in Env^p_{\mathrm{bt}} & = & Var^p \to Val_{\mathrm{bt}} & \text{Abstract environment}
\end{array}
$$

$$
\vDash^p_{\mathrm{bt}} \subseteq (Cache^p_{\mathrm{bt}} \times Env^p_{\mathrm{bt}}) \times Lab^p
$$

Figure 8: BTA relation for a program p

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, for 0-CFA as defined in Figure 5, no information is lost or gained by the CPS transformation.

4 Binding-Time Analysis

We consider a constraint-based binding-time analysis (BTA) [10, 25, 26, 28]. 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, 25]. 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 [26]. We have chosen to separate the flow analysis from the binding-time analysis in order to reuse the results from Section 3.

The formal definition of the analysis is similar to the definition of the 0-CFA of Section 3. The analysis is a relation defined on essentially the same domains (Figure 8); the difference is that the domain of abstract values is now the standard lattice $\{\mathbf{S} \sqsubseteq \mathbf{D}\}$ of static and dynamic annotations. The analysis relation is defined inductively over the syntax (Figure 9). At applications, the definition of the BTA refers to the flow information $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$, which is considered to be the least solution of the control-flow analysis of Section 3.

In contrast to the 0-CFA of Section 3, the BTA accepts non-closed terms. Following the tradition, we consider the program to be dynamic and its free variables to be dynamic as well. The flow information for the free variables 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-abstractions, conditionals and let-expressions. The significance 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 in Hatcliff and Danvy's work [10].

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 in linear time 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 $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ a solution of the BTA over p, we define $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ 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 10, where the left parts are identical to the left parts of Figures 6 and 7. In the right part, we assign binding times into $\widehat{C}'_{\mathrm{bt}}$ and $\widehat{\rho}'_{\mathrm{bt}}$. As in Section 3, we use $\mathcal{C}^p_{\mathrm{bt}}$ to denote the function induced by the transformation.

$$\mathcal{C}^p_{\mathrm{bt}} : (\mathit{Cache}^p_{\mathrm{bt}} \times \mathit{Env}^p_{\mathrm{bt}}) \to (\mathit{Cache}^{p'}_{\mathrm{bt}} \times \mathit{Env}^{p'}_{\mathrm{bt}}).$$

4.2 Correctness of the transformation

Let us show that the solution defined in Figure 10 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. *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ be an arbitrary solution of the BTA of p (i.e., such that $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^p_{\mathrm{bt}} p$ holds). If $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}}) = \mathcal{C}^p_{\mathrm{bt}}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ then $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}}) \vDash^{p'}_{\mathrm{bt}} p'$ holds.*

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

The binding time of an expression in p is equal to the binding time of its return point.

Lemma 9. *Let e^ℓ be an arbitrary subexpression of p. Then $\widehat{C}_{\mathrm{bt}}(\mathcal{R}[\![e^\ell]\!]) = \widehat{C}_{\mathrm{bt}}(\ell)$.*

The flow of the continuation abstractions connects the binding times of the return point 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.

$$
\begin{array}{lll}
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} n^{\ell} & \Longleftrightarrow & \mathit{true} \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} x^{\ell} & \Longleftrightarrow & \widehat{\rho}_{\mathrm{bt}}(x) = \widehat{C}_{\mathrm{bt}}(\ell) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} (\lambda^{\pi} x.e^{\ell_1})^{\ell} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e^{\ell_1} \wedge \\
 & & (\widehat{C}_{\mathrm{bt}}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} (\mathbf{let}\ x = t^{\ell} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_1})^{\ell_2} & & \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{C}_{\mathrm{bt}}(\ell_2) \wedge \\
 & & \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \mathbf{D} \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} (\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} t_0^{\ell_0} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} t_1^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e^{\ell_2} \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell_3) \wedge \\
 & & (\widehat{C}_{\mathrm{bt}}(\ell_0) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \wedge \\
 & & (\widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_2) = \mathbf{D}) \wedge \\
 & & \forall (\lambda^{\pi} y.e_1^{\ell}) \in \widehat{C}_{\mathrm{cf}}(\ell_0). \\
 & & \quad (\widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(y) \wedge \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x)) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} (\mathbf{let}\ x = op(t^{\ell}) & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_1})^{\ell_2} & & \widehat{C}_{\mathrm{bt}}(\ell) \sqsubseteq \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{C}_{\mathrm{bt}}(\ell_2) \wedge \\
 & & (\widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \mathbf{D}) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} (\mathbf{let}\ x = & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e_0^{\ell_0} \wedge \\
\quad \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e_1^{\ell_1} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} e^{\ell_2} \wedge \\
\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & \widehat{C}_{\mathrm{bt}}(\ell_0) = \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell_3) \\
 & & (\widehat{C}_{\mathrm{bt}}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_0) = \widehat{C}_{\mathrm{bt}}(\ell_1) = \mathbf{D}) \wedge \\
 & & (\widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_2) = \mathbf{D}) \\
 & & \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}} p & \Longleftrightarrow & (\forall x.x \text{ free in } p \Rightarrow \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \wedge \\
 & & (p = e^{\ell} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell) = \mathbf{D})
\end{array}
$$

Figure 9: Syntax-directed BTA

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

$$[\![\lambda^{\pi_1} x_1.e^{\ell_0}]\!]^{Triv} = \lambda^{\pi_1} x_1.\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

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

The proof of Theorem 4 is presented in Appendix A.

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.

$$
\begin{array}{rcl}
[\![e^\ell]\!]^{Pgm} & = & (\lambda^\pi k.[\![e^\ell]\!]^{Exp}\,k)^{\ell_0} \qquad \widehat{C}'_{\mathrm{bt}}(\ell_0) = \widehat{\rho}'_{\mathrm{bt}}(k) = \mathbf{D} \\
\\
[\![n^\ell]\!]^{Triv} & = & n^\ell \qquad \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
[\![x^\ell]\!]^{Triv} & = & x^\ell \qquad \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{Triv} & = & (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp}\,k)^{\ell_2})^\ell \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell) \quad \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
& & \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{\rho}'_{\mathrm{bt}}(k) = \widehat{C}_{\mathrm{bt}}(\ell) \\
\\
[\![t^\ell]\!]^{Exp}\,k & = & (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2} \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_0) = \widehat{\rho}'_{\mathrm{bt}}(k) \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_2) = \widehat{C}'_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ \mathbf{in}\ e^\ell)^{\ell_1}]\!]^{Exp}\,k & = & (\mathbf{let}\ x = [\![t_0^{\ell_0}]\!]^{Triv}\ \mathbf{in}\ [\![e^\ell]\!]^{Exp}\,k)^{\ell_2} \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_2) = \mathbf{D} \quad \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \\
\\
[\![(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^\ell)^{\ell_2}]\!]^{Exp}\,k & = & (\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in} \\
& & \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{Exp}\,k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_4) = \widehat{C}'_{\mathrm{bt}}(\ell_3) = \widehat{\rho}'_{\mathrm{bt}}(x_0) = \widehat{C}_{\mathrm{bt}}(\ell_0) \\
& & \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{\rho}'_{\mathrm{bt}}(x_1) = \mathbf{D} \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_7) = \widehat{C}'_{\mathrm{bt}}(\ell_6) = \widehat{C}'_{\mathrm{bt}}(\ell_5) = \mathbf{D} \\
\\
[\![(\mathbf{let}\ x = op(t^\ell)\ \mathbf{in}\ e^{\ell_0})^{\ell_1}]\!]^{Exp}\,k & = & (\mathbf{let}\ x = op([\![t^\ell]\!]^{Triv})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{Exp}\,k)^{\ell_2} \\
& & \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{C}'_{\mathrm{bt}}(\ell_2) = \mathbf{D} \\
\\
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} \\ \ \mathbf{in}\ e^{\ell_2})^{\ell_3}\end{array}\right]\!\!\right]^{Exp} k & = & (\mathbf{let}\ k_1 = (\lambda^\pi x.[\![e^{\ell_2}]\!]^{Exp}\,k)^{\ell_4}\ \mathbf{in} \\
& & \ (\mathbf{let}\ x_1 = \mathbf{if0}\ [\![t^\ell]\!]^{Triv}\ ([\![e_0^{\ell_0}]\!]^{Exp}\,k_1)\ ([\![e_1^{\ell_1}]\!]^{Exp}\,k_1) \\
& & \ \ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\
& & \widehat{\rho}'_{\mathrm{bt}}(k_1) = \widehat{C}'_{\mathrm{bt}}(\ell_4) = \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \\
& & \widehat{C}'_{\mathrm{bt}}(\ell_7) = \widehat{C}'_{\mathrm{bt}}(\ell_6) = \widehat{C}'_{\mathrm{bt}}(\ell_5) = \mathbf{D} \\
& & \widehat{\rho}'_{\mathrm{bt}}(x_1) = \mathbf{D}
\end{array}
$$

Figure 10: Transformation of binding times from direct style to CPS

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 [10], where *inc* is the successor function and the free variable z is considered to be dynamic:

$$\mathbf{let}\ r = (\lambda^\pi y.\mathbf{let}\ v = inc(z)\ \mathbf{in}\ 2)\ 1\ \mathbf{in}\ \mathbf{let}\ r_1 = inc(r)\ \mathbf{in}\ r_1$$

In the least solution of the BTA on this term, even if the application of λ^π to 1 is classified as static, its result is classified as dynamic because of the dynamic header of

$$
\begin{array}{lll}
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} n^{\ell} & \Longleftrightarrow & \mathit{true} \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} x^{\ell} & \Longleftrightarrow & \widehat{\rho}_{\mathrm{bt}}(x) = \widehat{C}_{\mathrm{bt}}(\ell) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} (\lambda^{\pi} x.e^{\ell_1})^{\ell} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e^{\ell_1} \wedge \\
 & & (\widehat{C}_{\mathrm{bt}}(\ell) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} (\mathbf{let}\ x = t^{\ell} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_1})^{\ell_2} & & \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{C}_{\mathrm{bt}}(\ell_2) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} (\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} t_0^{\ell_0} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} t_1^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e^{\ell_2} \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell_3) \wedge \\
 & & (\widehat{C}_{\mathrm{bt}}(\ell_0) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \wedge \\
 & & \forall(\lambda^{\pi} y.e_1^{\ell}) \in \widehat{C}_{\mathrm{cf}}(\ell_0). \\
 & & \quad (\widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(y) \wedge \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x)) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} (\mathbf{let}\ x = op(t^{\ell}) & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e^{\ell_1} \wedge \\
\quad \mathbf{in}\ e^{\ell_1})^{\ell_2} & & \widehat{C}_{\mathrm{bt}}(\ell) \sqsubseteq \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{C}_{\mathrm{bt}}(\ell_2) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} (\mathbf{let}\ x = & \Longleftrightarrow & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} t^{\ell} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e_0^{\ell_0} \wedge \\
\quad \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e_1^{\ell_1} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} e^{\ell_2} \wedge \\
\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & \widehat{C}_{\mathrm{bt}}(\ell_0) = \widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell_3) \\
 & & \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \vDash^{p}_{\mathrm{bt}^\star} p & \Longleftrightarrow & (\forall x.x \text{ free in } p \Rightarrow \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}) \wedge \\
 & & (p = e^{\ell} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell) = \mathbf{D})
\end{array}
$$

Figure 11: Syntax-directed BTA for continuation-based partial evaluation

the let-expression. Thus r is dynamic. Since the second increment operation depends on r, it is dynamic as well. In a realistic setting, simply discarding the dynamic computation $\mathit{inc}(z)$ 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 (without embedding it into direct style, for readability):

$$\lambda k.\ \begin{pmatrix} \lambda^{\pi} y.\lambda k_1.\mathbf{let}\ v = \mathit{inc}(z) \\ \mathbf{in}\ k_1\ 2 \end{pmatrix}\ 1\ \begin{pmatrix} \lambda r.\mathbf{let}\ r_1 = \mathit{inc}(r) \\ \mathbf{in}\ k\ r_1 \end{pmatrix}$$

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, 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:

$$\mathbf{let}\ v = \mathbf{if0}\ z\ 0\ 1\ \mathbf{in}\ \mathbf{let}\ v_1 = \mathit{inc}(v)\ \mathbf{in}\ v_1$$

It is true that one benefits from such an improvement only by allowing code duplication. However, the code duplication takes place at specialization time, not at BTA time. Thus in contrast to Sabry and Felleisen's observation [31], the improvement in this case is 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 9): the body of a let-expression has to be dynamic if the header is dynamic, and both branches of a conditional have to be dynamic if the test is dynamic.

The binding-time constraint in the body of a let-expression reflects the concern about which reductions can be performed safely by the specializer. In the context of the computational metalanguage [10], a named dynamic computation cannot be discarded due to possible computational effects. Similarly, the constraint over the branches of a conditional is introduced because one cannot decide statically which conditional branch should be selected.

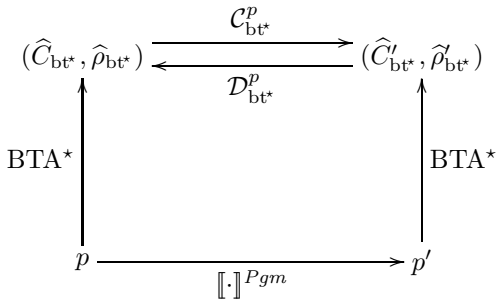
The above-mentioned constraint on the body of a let-expression can be relaxed if one uses a *continuation-based program specializer* [2, 10, 18]. The constraint connecting the branches of a test with the test itself can be relaxed as well if one allows the same continuation-based specializer to lift the test above the context, either by duplicating the context or by using a let-expression. 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 9, without the constraints mentioned just above. Naming $\vDash^p_{\mathrm{bt}^\star}$ the new relation, we replace the let rules as specified in Figure 11. The result is BTA*.

Using the same proof technique as in Section 3, we can formally show that the CPS transformation has no effect on BTA*, 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 precisely, we can define $\mathcal{C}^p_{\mathrm{bt}^\star}$, the CPS transformation of the binding times obtained by BTA*. The definition is only a slight modification of the definition of $\mathcal{C}^p_{\mathrm{bt}}$ in Section 4.1. Given the program p and a solution $(\widehat{C}_{\mathrm{bt}^\star}, \widehat{\rho}_{\mathrm{bt}^\star})$ of BTA* (i.e., such that $(\widehat{C}_{\mathrm{bt}^\star}, \widehat{\rho}_{\mathrm{bt}^\star}) \vDash^p_{\mathrm{bt}^\star} p$ holds), we can show that $\mathcal{C}^p_{\mathrm{bt}^\star}(\widehat{C}_{\mathrm{bt}^\star}, \widehat{\rho}_{\mathrm{bt}^\star}) \vDash^{p'}_{\mathrm{bt}^\star} p'$ holds. We can also define the reverse binding-time transformation $\mathcal{D}^p_{\mathrm{bt}^\star}$, which is essentially the same as the reverse flow transformation of Section 3.3 and also operates in linear time: for each term we just extract the binding time of its CPS counterpart. We can show that given a solution $(\widehat{C}'_{\mathrm{bt}^\star}, \widehat{\rho}'_{\mathrm{bt}^\star})$ of BTA* for p' (i.e., such that $(\widehat{C}'_{\mathrm{bt}^\star}, \widehat{\rho}'_{\mathrm{bt}^\star}) \vDash^{p'}_{\mathrm{bt}^\star} p'$ holds), $\mathcal{D}^p_{\mathrm{bt}^\star}(\widehat{C}'_{\mathrm{bt}^\star}, \widehat{\rho}'_{\mathrm{bt}^\star}) \vDash^p_{\mathrm{bt}^\star} p$ holds too.

Graphically:



We are now in position to connect the binding times in direct style and in CPS as obtained by BTA*:

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

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

The issue of syntactic accidents seems to be folklore in the program-analysis community (Hanne Riis Nielson, personal communication, March 2000). We are, however, only aware of three studies: Nielson's early work on data-flow analysis [21],[5] Henglein's invariance properties of polymorphic typing judgments with respect to let unfolding and folding and η-reduction [12], and Sabry and Felleisen's work on constant propagation which shows that performing a CPS transformation leads to incomparable results of the analysis [31].

Sabry and Felleisen conclude that CPS can (1) improve results by duplicating the analysis over conditionals and (2) worsen results by confusing the return points of function calls.

(1) None of the 0-CFA and BTAs we consider here duplicates the analysis over conditionals.

(2) Sabry and Felleisen's treatment of function calls distinguishes the order in which these calls are encountered. Confusing return points exerts an impact on their analysis (namely a loss in the precision of the results), because of this distinction. In contrast, the 0-CFA and the BTAs are not specified operationally but with constraints, and as such, they do not have such chronological dependencies. Already in direct style, the constraint-based approach (in the monovariant case) propagates the result of a function at once to all the application sites of this function. (This property enabled us to show that the CPS transformation preserves the results of both analyses.)

Recently, Palsberg and Wand have conducted a similar study for 0-CFA [29], supporting Sabry and Felleisen's conclusion that the extra precision enabled by the CPS transformation is due to the duplication of the analysis. In their study, they developed a CPS transformation of flow information comparable to the one of Figure 6, but independently and prior to us. Palsberg and Wand also mention that least solutions may or may not be preserved by administrative reductions of CPS-transformed programs. In that, they implicitly share our concern about syntactic accidents, even though their

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

primary goal was to transfer Wand's pioneer results on the CPS transformation of types [19, 34] to the CPS transformation of flow types.

5.2 Binding-time analysis and the CPS transformation

Binding-time improvements have always been customary for users of binding-time analysis [16, 22]. One of them amounts to considering source programs in CPS [4, 6], which suggests that source programs should be systematically CPS-transformed [3]. (Muylaert-Filho and Burn take the same stand for strictness analysis and the call-by-name CPS transformation [20].)

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 [10]. This characterization justifies why offline let flattening is also, partially, a binding-time improvement [13]. In any case, offline let flattening is known to be part of the CPS transformation [9].

What had not been shown before, 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 considered a radical syntactic change: a transformation into CPS. 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, 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 [10]. 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 understand better the program-analysis perspective over syntactic landscapes. Two key questions arise which

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 Andrzej Filinski and Julia L. Lawall for substantial discussions and comments. Thanks are also due to Torben Amtoft, Anindya Banerjee, Bernd Grobauer, Dan Hernest, Niels O. Jensen, Lasse R. Nielsen, Morten Rhiger, Amr Sabry, Zhe Yang, and the anonymous referees for further comments.

A Proofs

Proof of Theorem 1. The proof proceeds by induction on the transformation of p into p'. We sketch the induction steps.

We show that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \vDash^{p'}_{\mathrm{cf}} (\mathbf{let}\ x = k^{\ell_0}\ [\![t^{\ell}]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2}$ holds. For an arbitrary continuation $\lambda^{\pi} y.e^{\ell_3}$ in the set $\widehat{C}'_{\mathrm{cf}}(\ell_0) = \widehat{\rho}'_{\mathrm{cf}}(k)$, we show that the flow constraints for the application $k^{\ell_0}\ [\![t^{\ell}]\!]^{Triv}$ are satisfied.

The first constraint is $\widehat{C}'_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}'_{\mathrm{cf}}(y)$. By Lemma 1, $\widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell)$. We make a case analysis on the introduction of k by the CPS transformation.

If k is the top-level continuation, then the constraints are vacuously satisfied. If k is introduced by the transformation of a named conditional, then ℓ is the return point of one of the two branches of the test. From Lemma 2 and from the definition of $\widehat{\rho}'_{\mathrm{cf}}(y)$, $\widehat{C}_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}'_{\mathrm{cf}}(y)$. Otherwise, k comes from the transformation of a λ-abstraction $\lambda^{\pi_1} x_1.e^{\ell_4}$ from p, such that $\ell = \mathcal{R}[\![e^{\ell_4}]\!]$. We apply Lemma 3.

The second constraint is $\widehat{C}'_{\mathrm{cf}}(\ell_3) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$, which amounts to $\emptyset \subseteq \emptyset$, by Lemma 1.

For the transformation of a named application:

$$[\![(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^{\ell})^{\ell_2}]\!]^{Exp} k = (\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in}\ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^{\pi} x.[\![e^{\ell}]\!]^{Exp} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7}$$

let $\lambda^{\pi_1} y.e^{\ell_8}$ be a λ-abstraction in p such that $\pi_1 \in \widehat{C}_{\mathrm{cf}}(\ell_0)$, and let $\lambda^{\pi_1} y.\lambda^{\pi_2} k_1.e$ be its CPS translation. It is easy to show that $\widehat{C}'_{\mathrm{cf}}(\ell_1) \subseteq \widehat{\rho}'_{\mathrm{cf}}(y)$, and, from $\widehat{\rho}'_{\mathrm{cf}}(x_0) = \gamma(\widehat{C}_{\mathrm{cf}}(\ell_0))$, that $\pi \in \widehat{\rho}'_{\mathrm{cf}}(k_1)$. The rest of the constraints are trivially satisfied.

For the remaining induction steps, the induction hypotheses and the definition of γ suffice to show that the constraints are satisfied. □

Proof of Theorem 2. The proof is by induction on the transformation of p into p'. We sketch the induction steps.

For the transformation step $[\![t^{\ell}]\!]^{Triv}$, the constraints follow from the induction hypothesis. The same applies for the transformation step $[\![t^{\ell}]\!]^{Exp} k$.

For the transformation of a named application:

$$[\![(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1}\ \mathbf{in}\ e^{\ell})^{\ell_2}]\!]^{Exp} k = (\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in}\ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^{\pi} x.[\![e^{\ell}]\!]^{Exp} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7}$$

let $\lambda^{\pi_1} y.e^{\ell_8}$ be an arbitrary λ-abstraction from p such that $\pi_1 \in \widehat{C}_{\mathrm{cf}}(\ell_0)$. Let the CPS transformation of the λ-abstraction be $\lambda^{\pi_1} y.\lambda k_1.e$. Then $\pi \in \widehat{\rho}'_{\mathrm{cf}}(k_1)$. From Lemma 5 and Lemma 6 we obtain that $\widehat{C}_{\mathrm{cf}}(\ell_8) \subseteq \widehat{\rho}_{\mathrm{cf}}(x)$. The remaining induction steps are proven similarly, using Lemma 5 and Lemma 6. □

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 sketch the induction steps.

We show that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \vDash^{p'}_{\mathrm{cf}} (\mathbf{let}\ x = k^{\ell_0}\ [\![t^{\ell}]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2}$ holds. For this purpose, given an arbitrary $\lambda^{\pi} x.e^{\ell_3} \in \widehat{C}'_{\mathrm{cf}}(\ell_0) = \widehat{\rho}'_{\mathrm{cf}}(k)$ we must show that two equality constraints 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 $\widehat{C}'_{\mathrm{bt}}(\ell_0) = \mathbf{D} \Rightarrow \widehat{C}'_{\mathrm{bt}}(\ell) = \mathbf{D}$. Again, we make a case analysis on the introduction of k. The top-level case is trivial. The case where k is introduced by the transformation of a function $(\lambda y.e_1^{\ell_5})^{\ell_4}$ implies that $\widehat{C}_{\mathrm{bt}}(\ell_4) = \mathbf{D}$. Thus $\widehat{C}_{\mathrm{bt}}(\ell_5) = \mathbf{D}$ and then $\widehat{C}'_{\mathrm{bt}}(\ell) = \mathbf{D}$, since $\ell = \mathcal{R}[\![e_1^{\ell_5}]\!]$. The same reasoning follows for the case where k comes from the transformation of a named conditional.

The remaining cases follow directly from the induction hypotheses and the definition of $\widehat{C}'_{\mathrm{bt}}$, $\widehat{\rho}'_{\mathrm{bt}}$, $\widehat{C}_{\mathrm{cf}}$ and γ. □

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] 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.

[9] 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.

[10] 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.

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

[12] Fritz Henglein. Syntactic properties of polymorphic subtyping. Technical Report Semantics Report D-293, DIKU, Computer Science Department, University of Copenhagen, May 1996.

[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 [33].

[19] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi (summary). In Rohit Parikh, editor, *Logics of Programs – Proceedings*, number 193 in Lecture Notes in Computer Science, pages 219–224, Brooklyn, June 1985. Springer-Verlag.

[20] Juarez A. Muylaert-Filho and Geoffrey L. 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.

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

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

[26] 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.

[27] 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.

[28] 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.

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

[30] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[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.

[34] Mitchell Wand. Embedding type structure in semantics. In Mary S. Van Deusen and Zvi Galil, editors, *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 1–6, New Orleans, Louisiana, January 1985. ACM Press.

Recent BRICS Report Series Publications

RS-00-15 **Daniel Damian and Olivier Danvy.** ***Syntactic Accidents in Program Analysis: On the Impact of the CPS Transformation*****. June 2000. 26 pp. Extended version of an article to appear in** ***Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming*****, 2000.**

RS-00-14 **Ronald Cramer, Ivan B. Damgård, and Jesper Buus Nielsen.** ***Multiparty Computation from Threshold Homomorphic Encryption*****. June 2000. ii+38 pp.**

RS-00-13 **Ondřej Klíma and Jiří Srba.** ***Matching Modulo Associativity and Idempotency is NP-Complete*****. June 2000. 19 pp. To appear in** ***Mathematical Foundations of Computer Science: 25th International Symposium*****, MFCS '00 Proceedings, LNCS, 2000.**

RS-00-12 **Ulrich Kohlenbach.** ***Intuitionistic Choice and Restricted Classical Logic*****. May 2000. 9 pp.**

RS-00-11 **Jakob Pagter.** ***On Ajtai's Lower Bound Technique for R-way Branching Programs and the Hamming Distance Problem*****. May 2000. 18 pp.**

RS-00-10 **Stefan Dantchev and Søren Riis.** ***A Tough Nut for Tree Resolution*****. May 2000. 13 pp.**

RS-00-9 **Ulrich Kohlenbach.** ***Effective Uniform Bounds on the Krasnoselski-Mann Iteration*****. May 2000. 34 pp.**

RS-00-8 **Nabil H. Mustafa and Aleksandar Pekeč.** ***Democratic Consensus and the Local Majority Rule*****. May 2000. 38 pp.**

RS-00-7 **Lars Arge and Jakob Pagter.** ***I/O-Space Trade-Offs*****. April 2000. To appear in** ***7th Scandinavian Workshop on Algorithm Theory*****, SWAT '98 Proceedings, LNCS, 2000.**

RS-00-6 **Ivan B. Damgård and Jesper Buus Nielsen.** ***Improved Non-Committing Encryption Schemes based on a General Complexity Assumption*****. March 2000. 24 pp.**

RS-00-5 **Ivan B. Damgård and Mads J. Jurik.** ***Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes*****. March 2000. 19 pp.**

RS-00-4 **Rasmus Pagh.** ***A New Trade-off for Deterministic Dictionaries*****. February 2000.**