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

Daniel Damian
Olivier Danvy

BRICS Report Series **RS-01-54**

ISSN 0909-0878 **December 2001**

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 [§]

December 2001

Abstract

We show that a non-duplicating transformation into continuation-passing style (CPS) has no effect on control-flow analysis, a positive effect on binding-time analysis for traditional partial evaluation, and no effect on binding-time analysis for continuation-based partial evaluation: a monovariant control-flow analysis yields equivalent results on a direct-style program and on its CPS counterpart, a monovariant binding-time analysis yields less precise results on a direct-style program than on its CPS counterpart, and an enhanced monovariant binding-time analysis yields equivalent results on a direct-style program and on its CPS counterpart. Our proof technique amounts to constructing the CPS counterpart of flow information and of binding times.

Our results formalize and confirm a folklore theorem about traditional binding-time analysis, namely that CPS has a positive effect on binding times. What may be more surprising is that the benefit does not arise from a standard refinement of program analysis, as, for instance, duplicating continuations.

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 Journal of Functional Programming.
[†]Current affiliation: LION Bioscience Ltd., Compass House, 80-82 Newmarket Road, Cambridge CB5 8DZ, UK.
[‡]Basic Research in Computer Science (`www.brics.dk`), funded by 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 (and of region inference, see Section 8.1.1), 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 [6, 22]. 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 [22, Chapter 12], to the point that partial-evaluation users are encouraged to write programs in a particular style [21]. That said, binding-time-improvements are not specific to offline partial evaluation—they are also routine in staging transformations [23] and in the formal specification of programming languages for semantics-directed compiling [30, Section 8.2].

Since one of the most effective binding-time improvements is the transformation of source programs into continuation-passing style (CPS) [4, 42], people have wondered whether CPS may help program analysis in general. Nielson's early work on data-flow analysis [29] 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 1990's, Muylaert-Filho and Burn's work [28] was starting to provide 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?" [39], 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 [37]. At any rate, except for continuation-based partial evaluation [16], 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 constant propagation and is also believed to have a positive effect on binding-time analysis. Still, we do not know for sure whether this positive effect is truly positive, or whether it worsens binding times elsewhere in the source program. One may also wonder whether, besides distributive monotone frameworks, there exist other 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 (CFA) and BTA. Using a uniform proof technique, we formally show that:

(1) CPS has no effect on 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 can 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

Offline partial evaluation [22] is a staged technique for specializing programs. In a first phase, the binding times of a source program, i.e., which parts are static (and should be evaluated at partial-evaluation time) and which parts are dynamic (and should be part of the specialized program) are analyzed. In a second phase, specialization proper takes place (i.e., the static parts are evaluated and the dynamic parts are residualized). Binding-time analysis is thus a data-flow analysis and when source programs are higher-order, it is driven by control-flow information. Such information is in turn obtained by a control-flow analysis.

A partial evaluator is correct when the meaning of the residual program is the same as the meaning of the source program applied to the static input. In particular, if the source language includes computational effects (for instance non-termination), the specializer must ensure that all the dynamic side effects of the source program are identically exhibited by the residual program.

To ensure this contextual coherence, a binding-time analysis classifies a let expression to be dynamic if its header is dynamic, because of possible side effects in the header and regardless of the binding time of the 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.) Therefore, the body of the following λ-abstraction is classified as dynamic if e is dynamic:

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

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 independently of x, 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. (This intuition is formalized in the rest of this article.) Intuitively, b' also yields a static result and sends 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 [4] (see Section 8.2 for further detail). Nevertheless, 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 7, 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. Nevertheless, 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—and this is a point of this article—is also a global improvement.

This global improvement is distinct from the common method of improving precision of program analysis by duplicating the analysis over the same program points. Sabry and Felleisen, for example, said that any improvement in precision provided by CPS is solely due to continuation duplication [39]. This assessment is true for their analysis, but it does not hold in general, as we have just shown for binding-time analysis.

Other approaches to improving analysis results amount to refining the definition of the analysis by including more information, such as, for instance, context information [20, 31, 32, 41]. In contrast, CPS-transforming the source program naturally provides a representation of the context as a syntactic support for refinement to the (unchanged) analysis.

In his work on data-flow analysis [29], Nielson shows that duplicating the analysis over conditional branches improves the analysis results. Let us point out that the CPS transformation also leads to binding-time improvements for conditional expressions. Indeed, to ensure contextual coherence for conditionals, the binding-time analysis makes conditional branches dynamic if the test is dynamic. This approximation can be circumvented with a CPS transformation. Therefore, the improvement is not produced by duplicating the analysis, but merely by the context relocation induced by the CPS transformation. This point is developed further in Section 7.4.

$$\Lambda \xrightarrow[\text{encoding}]{\text{call-by-value}} \Lambda_{ml} \xrightarrow[let.assoc\,+\,let.\beta]{\text{normalization}} \Lambda_{mnf} \xrightarrow[\text{continuations}]{\text{introduction of}} \Lambda_{cps}$$

Figure 1: Staged CPS transformation

1.3 Overview

In this work we use a staged CPS transformation. Several equivalent methods exist for performing a global CPS transformation of a program. For example, one can use a Plotkin-style CPS transformation with administrative reductions [38], or one can stage the CPS transformation as normalization to a monadic normal form followed by introduction of continuations [15]. Palsberg and Wand use the former method [37], which can be extended to account for administrative reductions [8, 9]. We use the latter method here.

Elsewhere [8, 10], we have connected Danvy and Nielsen's CPS transformation [13] with program analysis. We have constructed the corresponding CPS transformation of control-flow information and confirmed the results reported in the present paper.

Therefore, we use a CPS transformation obtained as follows:

1. call-by-value embedding of the input program into Moggi's computational metalanguage [15, 27],
2. normalization under *let.assoc* and *let.β* (as defined in Hatcliff and Danvy's account of CPS [15]), and
3. introduction of continuations.

The staged transformation is visualized in the diagram of Figure 1.

The rest of this article is organized as follows: in Section 2 we define the input language, the transformation steps leading to CPS, and the program analyses. More specifically, in Section 2.1 we present the labeled language of input programs. In Section 2.2 we review the computational metalanguage and the corresponding call-by-value encoding of the input language. In Section 2.3 we recall the monadic let-reductions.

We continue by introducing the constraint-based analyses for the computational metalanguage. In Section 2.5 we specify the control-flow analysis. In Section 2.6 we specify the binding-time analysis corresponding to traditional partial evaluation. In Section 2.7 we specify the binding-time analysis corresponding to continuation-based partial evaluation.

In Section 3 we outline how to compare the results of a constraint-based program analysis across a program transformation.

In Section 4 we evaluate the effect on constraint-based analyses incurred by the normalization of the source program with respect to let-reductions: we investigate the effect of *let.β* (Section 4.1) and *let.assoc* (Section 4.2) reductions over each of the analyses. We conclude (Section 4.3) that linear let-reductions and let flattening do not change the result of the control-flow analysis, while they do improve the results of the traditional binding-time analysis.

In the remainder of the article, we evaluate the effect of introducing continuations (Section 5) over the result of control-flow analysis (Section 6), binding-time analysis

$$\begin{array}{rcll} e & \in & \mathit{Exp} & ::= \; x \mid n \mid \lambda x.e \mid \mathbf{rec}\ f(x).e \mid e_0\ e_1 \mid op(e) \mid \mathbf{if0}\ e\ e_0\ e_1 \\ x, f & \in & \mathit{Ide} & \quad \text{(identifiers)} \\ n & \in & \mathit{Int} & \quad \text{(integers)} \\ op & \in & \multicolumn{2}{l}{\text{(an unspecified set of base-type operators)}} \end{array}$$

Figure 2: The language Λ

for traditional partial evaluation (Sections 7.1 to 7.3) and binding-time analysis for continuation-based partial evaluation (Section 7.4). In Section 8 we review related work. In Section 9 we conclude and discuss further issues.

2 Constraint-based analyses for a computational metalanguage

We introduce the language of input programs and the individual transformations performed by the CPS transformation. We then present the three program analyses: CFA, BTA and BTA*.

2.1 The language Λ

We consider that programs are given in an untyped λ-language Λ. The terms of the language are expressions given by the grammar of Figure 2. The language includes literals, λ-abstractions, recursive function definitions, conditionals and base-type operators (for simplicity, we only consider unary operators here). We focus on call by value. Since the evaluation of terms in the language may not terminate, programs in Λ may exhibit non-termination as a computational effect.

A program p is a closed expression.

2.2 The computational metalanguage

The computational metalanguage Λ_{ml} [15] enforces the order of evaluation by introducing a **let** construct for naming intermediate computations and a **unit** construct for lifting a value into a computation.

$$e ::= \dots \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid \mathbf{unit}\ e$$

The computational metalanguage comes with a set of sound reasoning principles about programs which may have computational effects, such as non-termination. Such principles can be used to validate program transformations performed, for instance, inside a compiler. They can also be used to validate, for instance, a partial evaluator [16].

In order to make use of such principles, an input program in the language Λ is encoded into the computational metalanguage, enforcing its order of evaluation. For call by value, the encoding into the monadic metalanguage is defined in Figure 3.

Notice that, in addition to other known call-by-value encodings [2, 15, 40], we name the result of the application of two values (x_1 and x_2 in the translation of an application). This cosmetic change (indeed, it is only a $let.\eta$ expansion in the

$$
\begin{array}{rcl}
\mathcal{V}[\![x]\!] & = & \textbf{unit}\ x \\
\mathcal{V}[\![n]\!] & = & \textbf{unit}\ n \\
\mathcal{V}[\![\lambda x.e]\!] & = & \textbf{unit}\ \lambda x.\mathcal{V}[\![e]\!] \\
\mathcal{V}[\![\textbf{rec}\ f(x).e]\!] & = & \textbf{unit}\ \textbf{rec}\ f(x).e \\
\mathcal{V}[\![e_0\ e_1]\!] & = & \textbf{let}\ x_0 = \mathcal{V}[\![e_0]\!] \\
 & & \textbf{in let}\ x_1 = \mathcal{V}[\![e_1]\!] \\
 & & \quad \textbf{in let}\ x_2 = x_0\ x_1 \\
 & & \qquad \textbf{in unit}\ x_2 \\
\mathcal{V}[\![op(e)]\!] & = & \textbf{let}\ x_0 = \mathcal{V}[\![e]\!]\ \textbf{in let}\ x_1 = op(x_0)\ \textbf{in unit}\ x_1 \\
\mathcal{V}[\![\textbf{if0}\ e\ e_0\ e_1]\!] & = & \textbf{let}\ x_0 = \mathcal{V}[\![e]\!] \\
 & & \textbf{in let}\ x_1 = \textbf{if0}\ x_0\ \mathcal{V}[\![e_0]\!]\ \mathcal{V}[\![e_1]\!] \\
 & & \quad \textbf{in unit}\ x_1 \\
 & & \hfill \text{(where the } x_i \text{ are fresh)}
\end{array}
$$

Figure 3: Call-by-value encoding into the computational metalanguage

$$
\begin{array}{rcl}
\textbf{let}\ x = \textbf{unit}\ t\ \textbf{in}\ e & \rightarrow_{let.\beta} & e[t/x] \\
\textbf{let}\ x = e\ \textbf{in unit}\ x & \rightarrow_{let.\eta} & e \\
\textbf{let}\ x_2 = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2\ \textbf{in}\ e & \rightarrow_{let.assoc} & \textbf{let}\ x_1 = e_1\ \textbf{in let}\ x_2 = e_2\ \textbf{in}\ e
\end{array}
$$

Figure 4: The monadic let reductions

computational metalanguage) is part of our development of the CPS transformation of flow information.

2.3 The monadic let reductions

The call-by-value encoding leads to a separation of terms into two categories: trivial terms (noted with t) and serious terms (noted with s). Trivial terms represent values: constants, variables, λ-abstractions and recursive function definitions. Serious terms represent computations: applications, basic operations, conditionals, nesting of computations by naming intermediate results.

We recall the monadic let-reductions. Normalization under the let-reductions is the first step in a staged CPS transformation [15]. The let-β reduction, let-η reduction and the let-flattening reduction are presented in Figure 4.

2.4 Λ_v: a call-by-value subset of the computational metalanguage

In this paper, we focus on the call-by-value embedding. Therefore, we restrict ourselves to the subset of Λ_{ml} that forms the image of the call-by-value embedding of Λ. The language Λ_v of labeled terms is defined in Figure 5. Indeed, the call-by-value embedding produces either trivial terms ($t \in \mathit{Triv}$) or let-expressions. All serious terms ($s \in \mathit{Step}$) are named. Since for the call-by-value embedding the occurrences of the **unit** construct can be deduced from the context, we omit them in Λ_v terms.

Note that the language is such that the final result of a computation is also named, since we no longer perform *let*.η reductions in Λ_v before introducing continuations.

$$
\begin{array}{rcl}
p \in \mathit{Pgm} & ::= & e^{\ell} \\
e \in \mathit{Exp} & ::= & t \mid \mathbf{let}\ x = s\ \mathbf{in}\ e^{\ell} \\
s \in \mathit{Step} & ::= & t^{\ell} \mid t_0^{\ell_0}\ t_1^{\ell_1} \mid op(t^{\ell}) \mid \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} \mid (\mathbf{let}\ x = s\ \mathbf{in}\ e^{\ell_1})^{\ell_2} \\
t \in \mathit{Triv} & ::= & n \mid x \mid \lambda^{\pi} x.e^{\ell} \mid \mathbf{rec}^{\pi} f(x).e^{\ell} \\
x \in \mathit{Ide} & & \text{(identifiers)} \\
n \in \mathit{Int} & & \text{(integers)} \\
\ell \in \mathit{Lab} & & \text{(term labels)} \\
\pi \in \mathit{Lam} & & (\lambda\text{-abstraction labels}) \\
op \in & \multicolumn{2}{l}{\text{an unspecified set of base-type operators}}
\end{array}
$$

Figure 5: Λ_v: The call-by-value subset of the computational metalanguage

This aspect is part of our development of the CPS transformation of flow information, and will be illustrated further in Section 6.

For the purpose of program analysis, terms are labeled with labels ℓ taken from a countable set *Lab*. In addition, λ-abstractions and recursive functions are identified by 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*.

Definition 1 *A* properly labeled *expression is a labeled expression in which all labels are distinct and all variables are distinct.*

We should note that, for the purpose of control-flow analysis or binding-time analysis, it is not essential that the input program is properly labeled. But the precision of the analysis is increased if distinct program points have distinct labels, and distinct variables have distinct names. Since we want to compare the absolute precision of an analysis before and after program transformation, we consider the best results that the analysis can give over the program. For this reason, we consider only properly labeled programs and only transformations that lead to properly labeled programs.

2.5 Control-flow analysis for Λ_v

We consider a constraint-based, monovariant control-flow analysis (CFA) over programs in Λ_v. The constraint-based version [14, 20, 31, 34] is known to be equivalent to other versions, based on different methods such as set-based analysis [17] and type inference [35]; it is also known to be an instance of abstract interpretation [7]. For uniformity, we adopt the same definition and notation as in Nielson, Nielson and Hankin's recent textbook on program analysis [32].[1]

The flow information computed by the analysis is a pair consisting of an abstract cache $\widehat{C}_{\mathrm{cf}}$ mapping terms to abstract values and an abstract environment $\widehat{\rho}_{\mathrm{cf}}$ mapping 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.

[1]Nielson, Nielson and Hankin's CFA is developed for a call-by-value language with recursion and let-constructs. It is thus compatible with the language subset considered here.

Lam^p	The set of λ-abstraction labels in p
Var^p	The set of identifiers in p
Lab^p	The set of term labels in p

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

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

Figure 6: CFA relation for a program p

$$\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{rec}^\pi f(x).e^{\ell_1})^\ell & \iff & \{\pi\} \subseteq \widehat{C}_{\mathrm{cf}}(\ell) \wedge \{\pi\} \subseteq \widehat{\rho}_{\mathrm{cf}}(f) \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} & \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 \\
\qquad\qquad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\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) & \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 \\
\qquad\qquad \mathbf{in}\ e^{\ell_1})^{\ell_2} & & \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 = & \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 \\
\qquad\qquad \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} & & (\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 \\
\qquad\qquad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & \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) \\
(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^p_{\mathrm{cf}} (\mathbf{let}\ x = (\mathbf{let}\ x_1 = s & \iff & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^p_{\mathrm{cf}} (\mathbf{let}\ x_1 = s\ \mathbf{in}\ e_1^{\ell_1})^{\ell_2} \wedge \\
\qquad\qquad\qquad \mathbf{in}\ e_1^{\ell_1})^{\ell_2} & & (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^p_{\mathrm{cf}} e^{\ell_3} \wedge \widehat{C}_{\mathrm{cf}}(\ell_2) \subseteq \widehat{\rho}_{\mathrm{cf}}(x) \wedge \\
\qquad\qquad \mathbf{in}\ e^{\ell_3})^{\ell_4} & & \widehat{C}_{\mathrm{cf}}(\ell_3) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_4)
\end{array}$$

Figure 7: Control-flow analysis (CFA)

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.[2]

In this work we use the syntax-directed variant of the analysis [32, Chapter 3], and we restrict its analysis relation to a relation $\models^p_{\mathrm{cf}}$ associated to each program p being analyzed. Given a properly labeled program $p \in \Lambda_{ml}$, the functionality of the associated relation $\models^p_{\mathrm{cf}}$ is defined in Figure 6. The analysis relation is defined in Figure 7 by induction over the syntax of the program.

Any solution $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ accepted by the relation $\models^p_{\mathrm{cf}}$ (i.e., such that the state-

[2]In the notation of Nielson, Nielson, and Hankin [32], $\models_{\mathrm{cf}}$ is simply $\models$.

ment $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} p$ holds) is a conservative approximation of the exact flow information [32, Chapter 3]. Furthermore, the analysis relation $\models^{p}_{\mathrm{cf}}$ has a model-intersection property, i.e., the set of solutions accepted by $\models^{p}_{\mathrm{cf}}$ 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.

2.6 Binding-time analysis for Λ_v and traditional partial evaluation

We consider a constraint-based binding-time analysis (BTA) for the call-by-value subset Λ_v of the computational metalanguage. The analysis is an adaptation of Hatcliff and Danvy's BTA for the computational metalanguage [16], presented in constraint form [33, 34, 36]. The analysis determines binding times of program points and program variables. The binding-time information is used in offline partial evaluation [6, 22, 33]: the result of the analysis determines the static computations 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, which is known to give equivalent results [34]. We have chosen to separate the control-flow analysis from the binding-time analysis in order to investigate separately the effect of CPS on flow information and on binding times.

The formal definition of the analysis is similar to the definition of the CFA of Section 2.5. 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 application points, 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 2.5.

In contrast to the CFA of Section 2.5, 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 CFA to the program closed by abstraction over the free variables. Another difference with the CFA of Section 2.5 is that the constraints generated by the BTA are equality constraints.

Finally, additional constraints are generated for λ-abstractions, conditionals and let-expressions. For example, the argument and body of an abstraction are dynamic if the abstraction itself is dynamic. As mentioned in Section 1.2, the following binding-time constraints ensure contextual coherence. In each let expression, the body is constrained to be dynamic if the header is dynamic. In each conditional expression, both branches are constrained to be dynamic if the test is dynamic. Note that we allow static operations in dynamic contexts so that static computations can take place at partial-evaluation time. A proof of correctness of a specializer using the annotations obtained by this traditional BTA can be found in Hatcliff and Danvy's work [16].

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

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

Figure 8: BTA relation for a program p

$$
\begin{array}{lcl}
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} n^\ell & \iff & \mathit{true} \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} x^\ell & \iff & \widehat{\rho}_{\mathrm{bt}}(x) = \widehat{C}_{\mathrm{bt}}(\ell) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} (\lambda^\pi x.e^{\ell_1})^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^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}}) \models^p_{\mathrm{bt}} (\mathbf{rec}^\pi f(x).e^{\ell_1})^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e^{\ell_1} \wedge \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(f) \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}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x = t^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e^{\ell_1} \wedge \\
\qquad\qquad\qquad\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}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} t_0^{\ell_0} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} t_1^{\ell_1} \wedge \\
\qquad\qquad\qquad\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^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).(\widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(y) \wedge \\
 & & \qquad\qquad\qquad\qquad\quad \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x)) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x = \mathit{op}(t^\ell) & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e^{\ell_1} \wedge \\
\qquad\qquad\qquad\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}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x = & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e_0^{\ell_0} \wedge \\
\qquad\qquad\qquad\qquad \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e_1^{\ell_1} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e^{\ell_2} \wedge \\
\qquad\qquad\qquad\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) = \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}) \wedge \\
 & & \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell_3) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x = & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} (\mathbf{let}\ x_1 = s\ \mathbf{in}\ e_1^{\ell_1})^{\ell_2} \wedge \\
\qquad\qquad\qquad\qquad (\mathbf{let}\ x_1 = s & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} e^{\ell_3} \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \\
\qquad\qquad\qquad\qquad\ \ \mathbf{in}\ e_1^{\ell_1})^{\ell_2} & & (\widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D} \Rightarrow \widehat{C}_{\mathrm{bt}}(\ell_1) = \mathbf{D}) \wedge \\
\qquad\qquad\qquad\quad \mathbf{in}\ e^{\ell_3})^{\ell_4} & & \widehat{C}_{\mathrm{bt}}(\ell_3) = \widehat{C}_{\mathrm{bt}}(\ell_4) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}} p & \iff & (\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: Binding-time analysis for traditional partial evaluation (BTA)

2.7 Binding-time analysis for Λ_v and continuation-based partial evaluation

As mentioned in Section 1.2, the traditional binding-time analysis from Section 2.6 is overly conservative because of the context coherence constraint imposed in the let rule.

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

$$\models^p_{\mathrm{bt}^\star} \; \subseteq \; (\mathit{Cache}^p_{\mathrm{bt}} \times \mathit{Env}^p_{\mathrm{bt}}) \times \mathit{Lab}^p$$

Figure 10: BTA* relation for a program p

$$\begin{array}{lcl}
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} n^\ell & \iff & \mathit{true} \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} x^\ell & \iff & \widehat{\rho}_{\mathrm{bt}}(x) = \widehat{C}_{\mathrm{bt}}(\ell) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} (\lambda^\pi x.e^{\ell_1})^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^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}}) \models^p_{\mathrm{bt}^\star} (\mathbf{rec}^\pi f(x).e^{\ell_1})^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e^{\ell_1} \wedge \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(f) \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}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x = t^\ell & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e^{\ell_1} \wedge \\
\qquad\qquad\qquad\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}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} t_0^{\ell_0} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} t_1^{\ell_1} \wedge \\
\qquad\qquad\qquad\quad \mathbf{in}\ e^{\ell_2})^{\ell_3} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^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).(\widehat{C}_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(y) \wedge \\
 & & \qquad\qquad\qquad\qquad\qquad \widehat{C}_{\mathrm{bt}}(\ell) = \widehat{\rho}_{\mathrm{bt}}(x)) \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x = \mathit{op}(t^\ell) & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e^{\ell_1} \wedge \\
\qquad\qquad\qquad\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}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x = & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} t^\ell \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e_0^{\ell_0} \wedge \\
\qquad\qquad\qquad\qquad\quad \mathbf{if0}\ t^\ell\ e_0^{\ell_0}\ e_1^{\ell_1} & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e_1^{\ell_1} \wedge (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e^{\ell_2} \wedge \\
\qquad\qquad\qquad\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}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x = & \iff & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} (\mathbf{let}\ x_1 = s\ \mathbf{in}\ e_1^{\ell_1})^{\ell_2} \wedge \\
\qquad\qquad\qquad\qquad (\mathbf{let}\ x_1 = s & & (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} e^{\ell_3} \wedge \widehat{C}_{\mathrm{bt}}(\ell_2) = \widehat{\rho}_{\mathrm{bt}}(x) \wedge \\
\qquad\qquad\qquad\qquad\ \ \mathbf{in}\ e_1^{\ell_1})^{\ell_2} & & \widehat{C}_{\mathrm{bt}}(\ell_3) = \widehat{C}_{\mathrm{bt}}(\ell_4) \\
\qquad\qquad\qquad\quad \mathbf{in}\ e^{\ell_3})^{\ell_4} & & \\
(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^p_{\mathrm{bt}^\star} p & \iff & (\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: Binding-time analysis for continuation-based partial evaluation (BTA*)

(Compared to Figure 9, we disabled the context coherence constraints in the 5th, 6th, 7th, 8th and 9th case.)

The constraint reflects the concern about which reductions can be safely performed by the specializer. Indeed, in the computational metalanguage [16], a named dynamic computation cannot be discarded due to possible computational effects. Similarly, the contextual coherence constraint over the conditional branches is introduced because one cannot decide statically which conditional branch should be selected. We will show in Sections 4 and 7 that these context coherence constraints are the source of binding-time improvements by CPS transformation.

The context coherence constraint on the body of a let-expression can be relaxed if one uses a *continuation-based program specializer* [3, 16, 25]. The context coherence constraint connecting the conditional branches with the test 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 naming the continuation with a let-expression.

We consider a binding-time analysis which takes into account a continuation-based specializer. More formally, we consider the BTA of Figure 9, without the context coherence constraints mentioned above. The functionality of the new relation $\models^{p}_{\mathrm{bt}^\star}$ is defined in Figure 10, and it is identical to the functionality of the traditional BTA relation $\models^{p}_{\mathrm{bt}}$ (Figure 8). To define the new BTA relation, we replace the rules for let-expressions and conditional expressions as specified in Figure 11. The result is BTA*.

3 Comparing analysis results across program transformations

How do we compare the results of a program analysis before and after a program transformation? The result of an analysis is a function mapping labels and program variables to analysis information. For simplicity, we expect that the transformation preserves some of the labels and variables of the initial program. Under this assumption, we relate the results of the analysis by comparing the analysis information associated with the labels and variables preserved by the transformation.

Let us say that the program p is transformed into the program p'. Let us assume that the points (labels and variables) common to p and p' are identified as a set L. Let S be an arbitrary solution of the analysis of p and S' be an arbitrary solution of the analysis of p'. We consider that the solutions S and S' are equivalent if $S'|_L = S|_L$, where $S|_L$ is the restriction of the mapping S to the set L of common program points.

To establish a relationship between the two best analysis results we use a constructive technique. Given an arbitrary solution S of a constraint-based analysis of a program p, we show how to construct an equivalent solution S' of the analysis of the transformed program p'. We then show that the construction is valid, i.e., that S' is a valid solution of the analysis. Our construction induces a monotone mapping Φ between the two spaces of solutions. From the model-intersection property of the constraint-based analyses we conclude that the best result of the analysis of p' is at least as good as the results of the analysis of p. This situation is pictured in Figure 12.

In some cases, given a solution of the analysis of p', we are also able to construct an equivalent solution of the analysis of p, inducing an inverse mapping Ψ. When Φ and Ψ are both monotone and their composition in both ways leads to contractions (similarly to a Galois connection), we are able to show that the best result of the analysis of p is equivalent to the best result of the analysis of p'. In such cases we conclude that the specific program transformation has no impact on the result of the analysis.

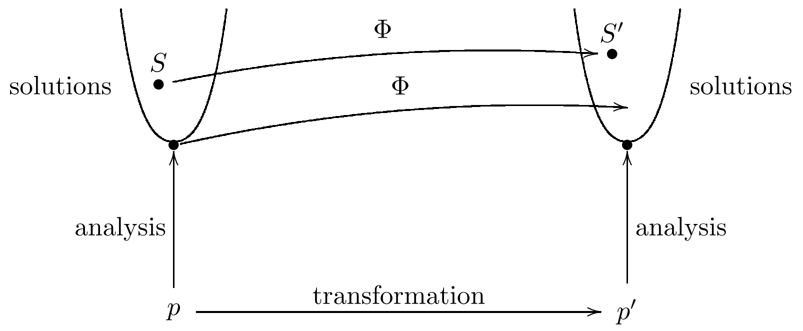

Figure 12: Comparing results of constraint-based analyses

4 Control-flow analysis, binding-time analysis and monadic let reductions

In order to avoid generating administrative redexes when introducing continuations, Λ_{ml}-programs need to be normalized with respect to the monadic let-reductions [15]. The Λ_v language is closed under the *let.β* and *let.assoc* reductions. In this section, we investigate the effect of each of the two reductions over the constraint-based analyses defined in Section 2.

According to the subject-reduction property of the control-flow or binding-time analyses [32, 33], a valid result of an analysis will also be a valid result of the analysis after a let-reduction (though not necessarily the least one). What is not clear, however, is whether the least (i.e., the best) result of the analysis is also the least result of the analysis after such reductions. We rely on the linearity of the transformations to show that flow information is not improved. We also show that let reductions may lead to strict binding-time improvements; we also show that the context coherence constraints are the cause of such improvements: disabling them leads to no improvements after a let reduction.

The let-expressions introduced by the call-by-value embedding of Figure 3 are linear: they do not duplicate or throw away code. Moreover, their linearity is preserved by the *let.β* and *let.assoc* reductions. In the following sections, we formalize the notion of linearity (Section 4.1), and use it to characterize the effect of the *let.β* and *let.assoc* reductions over CFA, BTA and BTA* (Sections 4.1.1 through 4.2.3).

4.1 Linear let-reduction

We formalize the notion of linear let-reduction as a *let.β* reduction such that the let body contains a unique occurrence of the variable named in the let header. The key observation, which we will prove in Section 4.1.1, is that linear reductions have no effect on the control-flow analysis. Linearity is essential: it is simple to show that non-linear (code-duplicating) reductions may improve the result of the control-flow analysis.

Definition 2 *A* linear context *is an expression with a unique hole* $[\cdot]$. *Linear contexts are defined by the following grammar:*

$$\begin{array}{rl} E & ::= T \mid (\textbf{let } x = S \textbf{ in } e^{\ell_1})^{\ell} \mid (\textbf{let } x = s \textbf{ in } E)^{\ell} \\ S & ::= T \mid T\ t_1^{\ell_1} \mid t_0^{\ell_0}\ T \mid op(T) \mid \textbf{if0}\ T\ e_0^{\ell_0}\ e_1^{\ell_1} \mid \textbf{if0}\ t^{\ell}\ E\ e_1^{\ell_1} \mid \textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ E \mid \\ & \quad (\textbf{let } x = S \textbf{ in } e^{\ell_1})^{\ell} \mid (\textbf{let } x = s \textbf{ in } E)^{\ell} \\ T & ::= [\cdot] \mid (\lambda^{\pi} x.E)^{\ell} \mid (\textbf{rec}^{\pi} f(x).E)^{\ell} \end{array}$$

We use linear contexts to identify contexts which are filled as the result of a *let.β* reduction. Note that linear contexts as defined in Definition 2 are more expressive than contexts that may result from the call-by-value embedding: the CPS transformation does not extract terms from inside lambda-expressions and conditional branches. Nevertheless, the results we are presenting hold in this enlarged setting.

We also formalize the notion of a let-context as a context where a let-reduction might take place.

Definition 3 *A* let context *is an expression which contains a unique hole* $[\cdot]$ *in the place of a let-expression. Let-contexts are defined by the following grammar:*

$$\begin{array}{rl} E & ::= [\cdot] \mid T \mid (\textbf{let } x = S \textbf{ in } e^{\ell_1})^{\ell} \mid (\textbf{let } x = s \textbf{ in } E)^{\ell} \\ S & ::= [\cdot] \mid T \mid T\ t_1^{\ell_1} \mid t_0^{\ell_0}\ T \mid op(T) \mid \textbf{if0}\ T\ e_0^{\ell_0}\ e_1^{\ell_1} \mid \textbf{if0}\ t^{\ell}\ E\ e_1^{\ell_1} \mid \\ & \quad \textbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ E \mid (\textbf{let } x = S \textbf{ in } e^{\ell_1})^{\ell} \mid (\textbf{let } x = s \textbf{ in } E)^{\ell} \\ T & ::= (\lambda^{\pi} x.E)^{\ell} \mid (\textbf{rec}^{\pi} f(x).E)^{\ell} \end{array}$$

Given a linear context E and a trivial term t^{ℓ}, we use $E[t^{\ell}]$ to denote the context E with the hole $[\cdot]$ replaced with t^{ℓ}. It is trivial to see that $E[t^{\ell}]$ is a well-formed expression. We use the same notation for plugging a labeled let-expression into a let context. Again, the operation is well defined.

We use $FV(e)$ to denote the set of free variables of the expression e. This notation naturally extends to contexts, by considering the hole $[\cdot]$ to contain no free variables. We also use L as the function extracting the label of an expression. By definition, for any labeled expression e^{ℓ}, $L(e^{\ell}) = \ell$.

Definition 4 *A* linear let *is an expression of the form* **let** $x = s$ **in** e^{ℓ} *such that* e^{ℓ} *contains a unique free occurrence of* x.

It is immediate to see that if a let-expression **let** $x = s$ **in** e^{ℓ} is linear, then there exists a linear context E and a label ℓ_1 such that $e^{\ell} = E[x^{\ell_1}]$.

Definition 5 *A* linear *let.β* reduction *is a let.β reduction of a linear let.*

It is relevant to notice that all the *let.β* redexes introduced by the call-by-value embedding are linear and that reducing any of these redexes does not change this property.

4.1.1 Linear *let.β* reduction and CFA

Let us show that a linear *let.β* reduction does not alter the results of the CFA. Let p be a properly labeled program such that there exist a let context E and a linear context E_1 such that

$$p = E[(\textbf{let } x = t^{\ell} \textbf{ in } E_1[x^{\ell_1}])^{\ell_2}]$$

Let p' be the program p after performing the linear *let.β* reduction:

$$p' = E[E_1[t^\ell]]$$

It is immediate to see that p' is a properly labeled program.

We show that the least solution of the flow analysis of p is equivalent to the least solution of the analysis of p'. In fact, the least solution for p' is obtained from the least solution for p by projection on the labels and variables preserved by the transformation.

We define the following functions:

- $\Phi_{\mathrm{cf}}^{let.\beta} : (\mathit{Cache}_{\mathrm{cf}}^{p} \times \mathit{Env}_{\mathrm{cf}}^{p}) \to (\mathit{Cache}_{\mathrm{cf}}^{p'} \times \mathit{Env}_{\mathrm{cf}}^{p'})$ such that

$$\Phi_{\mathrm{cf}}^{let.\beta}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = (\widehat{C}_{\mathrm{cf}}|_{Lab^{p'}}, \widehat{\rho}'_{\mathrm{cf}}|_{Var^{p'}})$$

- $\Psi_{\mathrm{cf}}^{let.\beta} : (\mathit{Cache}_{\mathrm{cf}}^{p'} \times \mathit{Env}_{\mathrm{cf}}^{p'}) \to (\mathit{Cache}_{\mathrm{cf}}^{p} \times \mathit{Env}_{\mathrm{cf}}^{p})$ such that, if $\Psi_{\mathrm{cf}}^{let.\beta}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$, then
  - $\widehat{C}_{\mathrm{cf}} = \widehat{C}'_{\mathrm{cf}} \sqcup [\ell_1 \mapsto \widehat{C}'_{\mathrm{cf}}(\ell), \ell_2 \mapsto \widehat{C}'_{\mathrm{cf}}(L(E_1[t^\ell]))]$
  - $\widehat{\rho}_{\mathrm{cf}} = \widehat{\rho}'_{\mathrm{cf}} \sqcup [x \mapsto \widehat{C}'_{\mathrm{cf}}(\ell)]$.

The two functions mediate between solutions for p and p'.

Lemma 4.1 *If* $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p} p$ *then* $\Phi_{\mathrm{cf}}^{let.\beta}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p'} p'$*, and if* $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p'} p'$ *then* $\Psi_{\mathrm{cf}}^{let.\beta}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p} p$*.*

It is immediate to show that $\Phi_{\mathrm{cf}}^{let.\beta}$ and $\Psi_{\mathrm{cf}}^{let.\beta}$ form an embedding/projection pair. The following lemma is a direct consequence.

Lemma 4.2 *If* $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ *is the least solution of the CFA of* p *and* $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ *is the least solution of the CFA of* p'*, then* $\Phi_{\mathrm{cf}}^{let.\beta}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = (\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ *and* $\Psi_{\mathrm{cf}}^{let.\beta}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$*.*

Lemma 4.2 says that the result of the CFA is preserved by a linear *let.β* reduction.

4.1.2 Linear *let.β* reduction and BTA

We show that a linear *let.β* reduction may improve the results of the BTA. Let p and p' be as defined in the previous section. We show that the least binding times of p' are as good and possibly better than the binding times of p.

We define the function $\Phi_{\mathrm{bt}}^{let.\beta} : (\mathit{Cache}_{\mathrm{bt}}^{p} \times \mathit{Env}_{\mathrm{bt}}^{p}) \to (\mathit{Cache}_{\mathrm{bt}}^{p'} \times \mathit{Env}_{\mathrm{bt}}^{p'})$ as

$$\Phi_{\mathrm{bt}}^{let.\beta}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) = (\widehat{C}_{\mathrm{bt}}|_{Lab^{p'}}, \widehat{\rho}'_{\mathrm{bt}}|_{Var^{p'}})$$

Lemma 4.3 *If* $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models_{\mathrm{cf}}^{p} p$ *then* $\Phi_{\mathrm{bt}}^{let.\beta}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models_{\mathrm{cf}}^{p'} p'$*.*

Lemma 4.3 says that the binding times are not worsened by a linear *let.β* reduction. Yet the analysis can yield strictly better results after a linear *let.β* reduction. In some cases, the binding times of the reduced program are strictly better than the binding times of the initial program.

For example, the call-by-value embedding of the term $(\lambda x.2)\ z$ followed by one linear *let.β* reduction yields the term:

$$\begin{array}{l}\mathbf{let}\ x_1 = z\ \mathbf{in}\\ \mathbf{let}\ x_2 = (\lambda x.2)\ x_1\ \mathbf{in}\ x_2\end{array}$$

Considering z to be dynamic, the let-rule forces the variable x_2 to be dynamic. Therefore, the constant 2 has to be dynamic as well, and, consequently, it will be residualized at specialization time. In contrast, after one more (linear) *let.β* reduction we obtain the term

$$\mathbf{let}\ x_2 = (\lambda x.2)\ z\ \mathbf{in}\ x_2$$

and we can see that, in a global static context, the value 2 is no longer coerced to be dynamic.

The context coherence constraint seems unjustified in the above case since evaluating the variable z has no side effects. But it is the call-by-value embedding which forces the variable z into a computation. The BTA has to impose the constraint in such cases as well [16]. At any rate, this initial loss of precision is avoided by performing the second *let.β* reduction.

In the next section we show that disabling the context coherence constraints leads to no loss or gain in the precision of the binding times.

4.1.3 Linear *let.β* reduction and BTA*

Let us show that the context coherence constraints from the standard BTA are the source of the benefit obtained by a linear *let.β* reduction. To do so, we show that a linear *let.β* reduction does not alter the results of the binding-time analysis for continuation-based partial evaluation, BTA*. We use the constructive technique outlined in Section 3. The function $\Phi^{let.\beta}_{\mathrm{bt}^\star} : (\mathit{Cache}^p_{\mathrm{bt}} \times \mathit{Env}^p_{\mathrm{bt}}) \rightarrow (\mathit{Cache}^{p'}_{\mathrm{bt}} \times \mathit{Env}^{p'}_{\mathrm{bt}})$ is identical to the one from Section 4.1.2. The function $\Psi^{let.\beta}_{\mathrm{bt}^\star} : (\mathit{Cache}^{p'}_{\mathrm{bt}} \times \mathit{Env}^{p'}_{\mathrm{bt}}) \rightarrow (\mathit{Cache}^p_{\mathrm{bt}} \times \mathit{Env}^p_{\mathrm{bt}})$ is defined similarly to $\Psi^{let.\beta}_{\mathrm{cf}}$ in Section 4.1.1. It is immediate to show that $\Psi^{let.\beta}_{\mathrm{bt}^\star} \circ \Phi^{let.\beta}_{\mathrm{bt}^\star} = id$ and $\Phi^{let.\beta}_{\mathrm{bt}^\star} \circ \Psi^{let.\beta}_{\mathrm{bt}^\star} = id$. The following lemma is a direct consequence:

Lemma 4.4 *If $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ is the least solution of the BTA* of p and $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ is the least solution of the BTA* of p', then $\Phi^{let.\beta}_{\mathrm{bt}^\star}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) = (\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ and $\Psi^{let.\beta}_{\mathrm{bt}^\star}(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ $= (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$.*

Lemma 4.4 says that the binding times obtained with BTA* are preserved by a linear *let.β* reduction.

4.2 Let flattening

We show that a *let.assoc* reduction has no effect on the CFA and on BTA*, and that it can improve and will not degrade the results of the standard BTA.

4.2.1 Let flattening and CFA

Let us show that a *let.assoc* reduction does not alter the results of the CFA. Let p be a properly labeled program as a let context E such that

$$p = E[(\textbf{let } x_1 = (\textbf{let } x = s \textbf{ in } e_1^{\ell_1})^{\ell} \textbf{ in } e_2^{\ell_2})^{\ell_3}]$$

Let p' be the program p after reassociating the let constructs:

$$p' = E[(\textbf{let } x = s \textbf{ in } (\textbf{let } x_1 = e_1^{\ell_1} \textbf{ in } e_2^{\ell_2})^{\ell_4})^{\ell_3}]$$

It is immediate to see that p' is a properly labeled program.

Again, we show that the least solution of the flow analysis of p is equivalent to the least solution of the analysis of p'. The least solution for p' is obtained from the least solution for p by projection on the labels and variables preserved by the transformation.

As in Section 4.1.1, we define the following functions:

- $\Phi_{\text{cf}}^{let.assoc} : (Cache_{\text{cf}}^{p} \times Env_{\text{cf}}^{p}) \rightarrow (Cache_{\text{cf}}^{p'} \times Env_{\text{cf}}^{p'})$ such that

$$\Phi_{\text{cf}}^{let.assoc}(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}}) = (\widehat{C}_{\text{cf}}|_{Lab^{p}\setminus\{\ell\}} \sqcup [\ell_4 \mapsto \widehat{C}_{\text{cf}}(\ell_2)], \widehat{\rho}_{\text{cf}}).$$

- $\Psi_{\text{cf}}^{let.assoc} : (Cache_{\text{cf}}^{p'} \times Env_{\text{cf}}^{p'}) \rightarrow (Cache_{\text{cf}}^{p} \times Env_{\text{cf}}^{p'})$ such that

$$\Psi_{\text{cf}}^{let.assoc}(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}}) = (\widehat{C}'_{\text{cf}}|_{Lab^{p'}\setminus\{\ell_4\}} \sqcup [\ell \mapsto \widehat{C}'_{\text{cf}}(\ell_1)], \widehat{\rho}'_{\text{cf}}).$$

The two functions mediate between solutions of the analysis of p and p'.

Lemma 4.5 *If* $(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}}) \models_{\text{cf}}^{p} p$ *then* $\Phi_{\text{cf}}^{let.assoc}(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}}) \models_{\text{cf}}^{p'} p'$*, and if* $(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}}) \models_{\text{cf}}^{p'} p'$ *then* $\Psi_{\text{cf}}^{let.assoc}(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}}) \models_{\text{cf}}^{p} p$*.*

Following the constructive technique from Section 3, we can easily prove the following lemma.

Lemma 4.6 *If* $(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}})$ *is the least solution of the CFA of* p *and* $(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}})$ *is the least solution of the CFA of* p'*, then* $\Phi_{\text{cf}}^{let.assoc}(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}}) = (\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}})$ *and* $\Psi_{\text{cf}}^{let.assoc}(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}}) = (\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}})$*.*

Lemma 4.6 says that the result of the CFA is preserved by let flattening.

4.2.2 Let flattening and BTA

Let us show that an isolated let flattening may improve the results of the BTA. Let p and p' be as defined in Section 4.2.1.

Again, we show that for any binding times of p there exist equivalent binding times of p'. We define the function $\Phi_{\text{bt}}^{let.assoc} : (Cache_{\text{bt}}^{p} \times Env_{\text{bt}}^{p}) \rightarrow (Cache_{\text{bt}}^{p} \times Env_{\text{bt}}^{p})$ such that

$$\Phi_{\text{bt}}^{let.assoc}(\widehat{C}_{\text{bt}}, \widehat{\rho}_{\text{bt}}) = (\widehat{C}_{\text{bt}}|_{Lab^{p}\setminus\{\ell\}} \sqcup [\ell_4 \mapsto \widehat{C}_{\text{bt}}(\ell_2)], \widehat{\rho}_{\text{bt}}).$$

Obviously $\Phi_{\text{bt}}^{let.assoc}(\widehat{C}_{\text{bt}}, \widehat{\rho}_{\text{bt}})$ is the equivalent of $(\widehat{C}_{\text{bt}}, \widehat{\rho}_{\text{bt}})$. The following lemma shows that $\Phi_{\text{bt}}^{let.assoc}$ constructs valid solutions.

Lemma 4.7 *If* $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^{p}_{\mathrm{bt}} p$ *then* $\Phi^{let.assoc}_{\mathrm{bt}}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^{p'}_{\mathrm{bt}} p'$.

Since $\Phi^{let.assoc}_{\mathrm{bt}}$ constructs valid equivalent solutions, by the considerations of Section 3, it follows that the binding times are not worsened by a let-flattening. The analysis, however, can yield strictly better results. In some cases, the binding times after a let-flattening are strictly better than the binding times of the initial program.

For example, the call-by-value embedding of the program $succ((\lambda x.2)\ (pred(z)))$, after a few $let.\beta$ and one $let.assoc$ reductions, leads to:

$$\begin{array}{l}
\mathbf{let}\ x_1 = \mathbf{let}\ x_2 = pred(z) \\
\qquad\qquad \mathbf{in\ let}\ x_3 = (\lambda x.2)\ x_2\ \mathbf{in}\ x_3 \\
\mathbf{in\ let}\ x_4 = succ(x_1)\ \mathbf{in}\ x_4
\end{array}$$

The program above reassociates to:

$$\begin{array}{l}
\mathbf{let}\ x_2 = pred(z) \\
\mathbf{in\ let}\ x_1 = \mathbf{let}\ x_3 = (\lambda x.2)\ x_2\ \mathbf{in}\ x_3 \\
\quad \mathbf{in\ let}\ x_4 = succ(x_1)\ \mathbf{in}\ x_4
\end{array}$$

In the first program, the let rule forces x_1 to be dynamic and the $succ(x_1)$ computation is dynamic. In the second program x_1 can be static, and the $succ(x_1)$ computation may be performed statically, and only its result (3) will be residualized.

4.2.3 Let flattening and BTA*

Let us show that for the $let.assoc$ reduction (similarly to the $let.\beta$ reduction in Section 4.1.3), all binding-time improvements come from the context coherence constraints. To do so, we show that a $let.assoc$ reduction has no effect on the binding-time analysis for continuation-based partial evaluation BTA*.

Taking p and p' as defined in Section 4.2.1, we define two functions $\Phi^{let.assoc}_{\mathrm{bt}^\star} : (Cache^{p}_{\mathrm{bt}} \times Env^{p}_{\mathrm{bt}}) \rightarrow (Cache^{p'}_{\mathrm{bt}} \times Env^{p'}_{\mathrm{bt}})$ and $\Psi^{let.assoc}_{\mathrm{bt}^\star} : (Cache^{p'}_{\mathrm{bt}} \times Env^{p'}_{\mathrm{bt}}) \rightarrow (Cache^{p}_{\mathrm{bt}} \times Env^{p}_{\mathrm{bt}})$ which map solutions of BTA* for p into solutions of BTA* for p' and vice-versa. The functions are essentially defined as in Section 4.2.1. One can show that $\Phi^{let.assoc}_{\mathrm{bt}^\star} \circ \Psi^{let.assoc}_{\mathrm{bt}^\star} = id$ and $\Phi^{let.assoc}_{\mathrm{bt}^\star} \circ \Psi^{let.assoc}_{\mathrm{bt}^\star} = id$. The following lemma is an immediate consequence:

Lemma 4.8 *If* $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ *is the least solution of the BTA** *of* p *and* $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ *is the least solution of the BTA** *of* p'*, then* $\Phi^{let.assoc}_{\mathrm{bt}^\star}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) = (\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}})$ *and*
$\Psi^{let.assoc}_{\mathrm{bt}^\star}(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}}) = (\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$.

4.3 Summary and conclusions

We have shown that, once the input program is embedded into the computational metalanguage, $let.\beta$ and $let.assoc$-normalization can yield binding-time improvements. At the same time linear $let.\beta$ and $let.assoc$ preserve the quality of flow information. This property confirms that monadic normal forms are a valuable intermediate representation in a program transformer and in an optimizing compiler.

$$\begin{array}{rcl} p \in \mathit{Pgm} & ::= & e^{\ell} \\ e \in \mathit{Exp} & ::= & t \mid \mathbf{let}\ x = s\ \mathbf{in}\ e^{\ell} \\ s \in \mathit{Step} & ::= & t_0^{\ell_0}\ t_1^{\ell_1} \mid op(t^{\ell}) \mid \mathbf{if0}\ t^{\ell}\ e_0^{\ell_0}\ e_1^{\ell_1} \\ t \in \mathit{Triv} & ::= & n \mid x \mid \lambda^{\pi} x.e^{\ell} \mid \mathbf{rec}^{\pi} f(x).e^{\ell} \end{array}$$

Figure 13: Λ_{mnf}: The subset of Λ_v normalized with respect to $let.\beta$ and $let.assoc$

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

Figure 14: Introducing continuations

5 Introducing continuations

The language resulting from normalizing terms in Λ_v under the $let.\beta$ and $let.assoc$ reductions is the language Λ_{mnf} defined in Figure 13. The effect of the normalization is to eliminate naming of trivial values and to flatten all nested computations. Therefore, in Λ_{mnf} a computational step can no longer be a trivial value or a nested computation.

The language Λ_{mnf} is the support for introducing continuations by the transformation shown in Figure 14. Introducing continuations leads to terms in a CPS language.[3] CPS is a restriction of direct style. In order to use the same program analysis, we therefore embed the CPS language into the Λ_v language. For example, applications are transformed into let-expressions that name partially applied CPS λ-abstractions and intermediate computations. Figure 15 displays the corresponding CPS transformation and embedding.[4] (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.)

We can apply now the constraint-based analyses of Section 2 on both the ($let.\beta +$ $let.assoc$)-normalized program and on its CPS counterpart given by the transformation of Figure 15.

[3]In Figure 14, $\widetilde{op}$ is the CPS counterpart of op, to ensure evaluation-order independence [38].

[4]In Figure 15, we use op instead of $\widetilde{op}$ since the direct-style language is call-by-value.

$$
\begin{array}{rcll}
[\![e]\!]^{Pgm} & = & \lambda k.[\![e]\!]^{Exp}k & \text{where } k \text{ is fresh} \\[1em]
[\![n]\!]^{Triv} & = & n & \\
[\![x]\!]^{Triv} & = & x & \\
[\![\lambda x.e]\!]^{Triv} & = & \lambda x.\lambda k.[\![e]\!]^{Exp}k & \text{where } k \text{ is fresh} \\
[\![\mathbf{rec}\ f(x).e]\!]^{Triv} & = & \mathbf{rec}\ f(x).\lambda k.[\![e]\!]^{Exp}k & \text{where } k \text{ is fresh} \\[1em]
[\![t]\!]^{Exp}k & = & \mathbf{let}\ x = k\ [\![t]\!]^{Triv}\ \mathbf{in}\ x & \text{where } x \text{ is fresh} \\[1em]
[\![\mathbf{let}\ x = t_0\ t_1\ \mathbf{in}\ e]\!]^{Exp}k & = & \mathbf{let}\ x_0 = [\![t_0]\!]^{Triv}\ [\![t_1]\!]^{Triv} & \\
 & & \mathbf{in}\ \mathbf{let}\ x_1 = x_0\ \lambda x.[\![e]\!]^{Exp}k\ \mathbf{in}\ x_1 & \\
 & & & \text{where } x_0 \text{ and } x_1 \text{ are fresh} \\[1em]
[\![\mathbf{let}\ x = op(t)\ \mathbf{in}\ e]\!]^{Exp}k & = & \mathbf{let}\ x = op([\![t]\!]^{Triv})\ \mathbf{in}\ [\![e]\!]^{Exp}k & \\[1em]
[\![\mathbf{let}\ x = \mathbf{if0}\ t\ e_0\ e_1\ \mathbf{in}\ e]\!]^{Exp}k & = & \mathbf{let}\ k_1 = \lambda x.[\![e]\!]^{Exp}k\ \mathbf{in} & \\
 & & \mathbf{let}\ x_1 = \mathbf{if0}\ [\![t]\!]^{Triv}\ ([\![e_0]\!]^{Exp}k_1)\ ([\![e_1]\!]^{Exp}k_1) & \\
 & & \mathbf{in}\ x_1 & \\
 & & & \text{where } k_1 \text{ and } x_1 \text{ are fresh}
\end{array}
$$

Figure 15: Introducing continuations and embedding into the Λ_v language

6 Control-flow analysis and the introduction of continuations

In order to compare the results of the CFA before and after introducing continuations, we follow the constructive technique outlined in Section 3. Therefore, the rest of this section is organized as follows. First, we show how to CPS-transform control-flow information (Section 6.1). Given a direct-style program p and an arbitrary solution of its associated analysis $(\widehat{C}_{\text{cf}}, \widehat{\rho}_{\text{cf}})$, we construct a solution $(\widehat{C}'_{\text{cf}}, \widehat{\rho}'_{\text{cf}})$ of the analysis associated to p', the CPS counterpart of p. We ensure that the construction $\Phi^{\text{CPS}}_{\text{cf}}$ builds a valid solution (Section 6.2). We present a converse transformation, $\Psi^{\text{CPS}}_{\text{cf}}$ (Section 6.3), which we also prove to be correct (Section 6.4). We then show that the two constructions preserve leastness (Section 6.5).

6.1 CPS transformation of control flow

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

As mentioned in Section 2.1, 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 $\mathit{Val}^p_{\mathrm{cf}} \subseteq \mathit{Val}^{p'}_{\mathrm{cf}}$. When introducing continuations, all the variables defined in the original direct-style program are preserved. 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, given A a set of λ-abstractions from the program p', $\gamma(A)$ is defined as the set of λ-abstractions $\lambda^{\pi^1} k.e^\ell$ such that $\lambda^\pi x.\lambda^{\pi^1} k.e^\ell \in A$ or such that $\mathbf{rec}^\pi f(x).\lambda^{\pi^1} k.e^\ell \in A$.
- ξ assigns flow information to each continuation identifier k introduced by the CPS transformation of p (at λ-abstractions and recursive function definitions). This information can be obtained 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:

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

  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

  $$\begin{aligned}[\![\mathbf{let}\ x = t_0^{\ell_0}\ t_1\ \mathbf{in}\ e]\!]^{\mathit{Exp}} k_1 = {} & \mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{\mathit{Triv}}\ [\![t_1]\!]^{\mathit{Triv}} \\ & \mathbf{in\ let}\ x_1 = x_0\ \lambda^\pi x.[\![e]\!]^{\mathit{Exp}} k_1\ \mathbf{in}\ x_1\end{aligned}$$

  such that $\pi_1 \in \widehat{C}_{\mathrm{cf}}(\ell_0)$. We make a similar definition for the continuation identifiers introduced at recursive function definitions.

Using γ and ξ, we define $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ inductively, following Figure 16. 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.

The construction of flow information defines a function

$$\Phi^{\mathrm{CPS}}_{\mathrm{cf}} : (\mathit{Cache}^p_{\mathrm{cf}} \times \mathit{Env}^p_{\mathrm{cf}}) \to (\mathit{Cache}^p_{\mathrm{cf}} \times \mathit{Env}^p_{\mathrm{cf}}).$$

It is easy to show that $\Phi^{\mathrm{CPS}}_{\mathrm{cf}}$ is monotone.

6.2 Correctness of the transformation

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

Theorem 6.1 *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{\mathit{Pgm}}$, let $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ be a solution of the CFA of p (i.e., such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^p_{\mathrm{cf}} p$ holds) and let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = \Phi^{\mathrm{CPS}}_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$. Then $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models^{p'}_{\mathrm{cf}} p'$ holds.*

$$
\begin{aligned}
[\![e^\ell]\!]^{Pgm} &= (\lambda^\pi k.[\![e^\ell]\!]^{Exp} k)^{\ell_0} \qquad \widehat{C}'_{\mathrm{cf}}(\ell_0) = \{\pi\} \quad \widehat{\rho}'_{\mathrm{cf}}(k) = \emptyset \\
[\![n^\ell]\!]^{Triv} &= n^\ell \qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \\
[\![x^\ell]\!]^{Triv} &= x^\ell \qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{Triv} &= (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \{\pi_1\} \\
&\qquad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \quad \widehat{\rho}'_{\mathrm{cf}}(k) = \xi(k)
\end{aligned}
$$

$$
\begin{aligned}
[\![(\mathbf{rec}^\pi f(x).e^{\ell_0})^\ell]\!]^{Triv} &= (\mathbf{rec}^\pi f(x).(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \{\pi_1\} \\
&\qquad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \quad \widehat{\rho}'_{\mathrm{cf}}(f) = \widehat{\rho}_{\mathrm{cf}}(f) \quad \widehat{\rho}'_{\mathrm{cf}}(k) = \xi(k)
\end{aligned}
$$

$$
\begin{aligned}
[\![t^\ell]\!]^{Exp} k &= (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2} \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell_0) = \widehat{\rho}'_{\mathrm{cf}}(k) \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \widehat{C}'_{\mathrm{cf}}(\ell_1) = \widehat{\rho}'_{\mathrm{cf}}(x) = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} \\ \ \mathbf{in}\ e^\ell)^{\ell_2}\end{array}\right]\!\!\right]^{Exp} k &= \begin{array}{l}(\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in} \\ \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{Exp} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7}\end{array} \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell_3) = \widehat{\rho}'_{\mathrm{cf}}(x_0) = \gamma(\widehat{C}_{\mathrm{cf}}(\ell_0)) \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell_4) = \{\pi\} \quad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \\
&\qquad \widehat{C}'_{\mathrm{cf}}(\ell_7) = \widehat{C}'_{\mathrm{cf}}(\ell_6) = \widehat{C}'_{\mathrm{cf}}(\ell_5) = \widehat{\rho}'_{\mathrm{cf}}(x_1) = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = op(t^\ell) \\ \ \mathbf{in}\ e^{\ell_0})^{\ell_1}\end{array}\right]\!\!\right]^{Exp} k &= (\mathbf{let}\ x = op([\![t^\ell]\!]^{Triv})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2} \\
&\qquad \widehat{\rho}'_{\mathrm{cf}}(x) = \widehat{\rho}_{\mathrm{cf}}(x) \quad \widehat{C}'_{\mathrm{cf}}(\ell_2) = \emptyset
\end{aligned}
$$

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

Figure 16: Transformation of control flow from direct style to CPS

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

Lemma 6.2 *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 CFA collects flow information, as shown just below in Lemma 6.3.

Definition 6 *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:*

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

Lemma 6.3 *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 6.4 *Let* k *be a continuation identifier introduced by the CPS transformation of a* λ*-abstraction from* p*:*

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

Then, for each $\lambda^\pi x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$*,* $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$*. Let* k *be a continuation identifier introduced by the CPS transformation of a recursive function definition from* p*:*

$$[\![\mathbf{rec}^{\pi_1} f(x_1).e^{\ell_0}]\!]^{\mathit{Triv}} = \mathbf{rec}^{\pi_1} f(x_1).\lambda k.[\![e^{\ell_0}]\!]^{\mathit{Exp}} k$$

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

Let us consider the first case. By the definition of ξ, the only possibility such that $\lambda^\pi x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$ is that the function is the continuation of an application point where $\lambda^{\pi_1} x_1.e^{\ell_0}$ is applied. Focusing on the application point, we show that $\widehat{C}_{\mathrm{cf}}(\ell_0) \subseteq \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x)$. From Lemma 6.3, $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{C}_{\mathrm{cf}}(\ell_0)$.

The proof of Theorem 6.1 is sketched in Appendix A.

6.3 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 17. 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 16 and 17 are identical.

We can show that Figure 17 defines another function

$$\Psi^{\mathrm{CPS}}_{\mathrm{cf}} : (\mathit{Cache}^p_{\mathrm{cf}} \times \mathit{Env}^p_{\mathrm{cf}}) \to (\mathit{Cache}^p_{\mathrm{cf}} \times \mathit{Env}^p_{\mathrm{cf}}).$$

It is also easy to show that, like $\Phi^{\mathrm{CPS}}_{\mathrm{cf}}$ in Section 6.2, $\Psi^{\mathrm{CPS}}_{\mathrm{cf}}$ is monotone.

$$
\begin{aligned}
[\![e^\ell]\!]^{\mathit{Pgm}} &= (\lambda^\pi k.[\![e^\ell]\!]^{\mathit{Exp}} k)^{\ell_0} \\
[\![n^\ell]\!]^{\mathit{Triv}} &= n^\ell & \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap \mathit{Lam}^p \\
[\![x^\ell]\!]^{\mathit{Triv}} &= x^\ell & \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap \mathit{Lam}^p \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{\mathit{Triv}} &= (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{\mathit{Exp}} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap \mathit{Lam}^p \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap \mathit{Lam}^p \\
[\![(\mathbf{rec}^\pi f(x).e^{\ell_0})^\ell]\!]^{\mathit{Triv}} &= (\mathbf{rec}^\pi f(x).(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{\mathit{Exp}} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap \mathit{Lam}^p \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap \mathit{Lam}^p \\
&\qquad \widehat{\rho}_{\mathrm{cf}}(f) = \widehat{\rho}'_{\mathrm{cf}}(f) \cap \mathit{Lam}^p
\end{aligned}
$$

$$
[\![t^\ell]\!]^{\mathit{Exp}} k = (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{\mathit{Triv}}\ \mathbf{in}\ x^{\ell_1})^{\ell_2}
$$

$$
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} \\ \ \mathbf{in}\ e^\ell)^{\ell_2}\end{array}\right]\!\!\right]^{\mathit{Exp}} k = \begin{array}{l}(\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{\mathit{Triv}}\ [\![t_1^{\ell_1}]\!]^{\mathit{Triv}}\ \mathbf{in} \\ \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{\mathit{Exp}} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7} \\ \qquad \widehat{C}_{\mathrm{cf}}(\ell_2) = \widehat{C}_{\mathrm{cf}}(\ell) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap \mathit{Lam}^p\end{array}
$$

$$
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = op(t^\ell) \\ \ \mathbf{in}\ e^{\ell_0})^{\ell_1}\end{array}\right]\!\!\right]^{\mathit{Exp}} k = \begin{array}{l}(\mathbf{let}\ x = op([\![t^\ell]\!]^{\mathit{Triv}})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{\mathit{Exp}} k)^{\ell_2} \\ \qquad \widehat{C}_{\mathrm{cf}}(\ell_1) = \widehat{C}_{\mathrm{cf}}(\ell_0) \quad \widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap \mathit{Lam}^p\end{array}
$$

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

Figure 17: Transformation of control flow from CPS to direct style

6.4 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 6.5 *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{\mathit{Pgm}}$, let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ be a solution of the CFA of p' (i.e., such that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p'} p'$ holds) and let $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = \Psi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$. Then $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models_{\mathrm{cf}}^{p} p$ holds.*

As in Section 6.2, we use intermediate results to prove Theorem 6.5. Working under the assumptions of the theorem, we observe two immediate properties of the reverse transformation:

Lemma 6.6 *For all $x \in \mathit{Var}^p$, $\widehat{\rho}_{\mathrm{cf}}(x) = \widehat{\rho}'_{\mathrm{cf}}(x) \cap \mathit{Lam}^p$; and for all trivial terms t^ℓ in p, $\widehat{C}_{\mathrm{cf}}(\ell) = \widehat{C}'_{\mathrm{cf}}(\ell) \cap \mathit{Lam}^p$.*

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

Lemma 6.7 *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 6.4, the following lemma connects the flow at the return points of functions with the flow collected for the variables declared by continuations.

Lemma 6.8 *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 $\lambda^{\pi} x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$, $\widehat{C}_{\mathrm{cf}}(\mathcal{R}[\![e^{\ell_0}]\!]) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$. Let k be a continuation identifier introduced by the transformation of a recursive function definition from p:

$$[\![\mathbf{rec}^{\pi_1} f(x_1).e^{\ell_0}]\!]^{Triv} = \mathbf{rec}^{\pi_1} f(x_1).\lambda k.[\![e^{\ell_0}]\!]^{Exp} k$$

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 6.5 is sketched in Appendix A.

6.5 Equivalence of flow

Let p be an arbitrary direct-style program and $p' = [\![p]\!]^{Pgm}$ its CPS counterpart. By simple unfoldings of definitions, we prove the following lemma.

Lemma 6.9 *Given $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$ a solution of the CFA of p (i.e., such that $(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) \models^{p}_{\mathrm{cf}} p$ holds), $\Psi^{\mathrm{CPS}}_{\mathrm{cf}}(\Phi^{\mathrm{CPS}}_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})) \subseteq (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$. Given $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ a solution of the CFA of p', (i.e., such that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models^{p'}_{\mathrm{cf}} p'$ holds), then it holds that $\Phi^{\mathrm{CPS}}_{\mathrm{cf}}(\Psi^{\mathrm{CPS}}_{\mathrm{cf}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})) \subseteq (\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$.*

From these two properties the following main theorem follows directly.

Theorem 6.10 (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 CFA of p and let $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ be the least solution of the CFA of p'. Then $\Phi^{\mathrm{CPS}}_{\mathrm{cf}}(\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}}) = (\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}})$ and $\Psi^{\mathrm{CPS}}_{\mathrm{cf}}(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) = (\widehat{C}_{\mathrm{cf}}, \widehat{\rho}_{\mathrm{cf}})$.*

6.6 Summary and conclusions

Theorem 6.10 shows that the best flow information obtainable by a constraint-based control-flow analysis on a direct-style program is equivalent to the best flow information obtainable by the same analysis on the CPS counterpart of this program and vice versa. Lemma 6.2 and Lemma 6.6 show that the two solutions are equal on the variables and program points common to the two programs. We conclude that, for CFA as defined in Figure 7, no information is lost or gained by the CPS transformation.

7 Binding-time analysis and the introduction of continuations

We describe the effect of the introduction of continuations on the result of the BTA of a program in Λ_{mnf}. First, we define a CPS transformation of binding times (Section 7.1), which we show to be correct and to preserve the quality of the binding times

(Section 7.2). Unlike for CFA, however, we show examples where BTA on CPS terms gives more precise results than on the corresponding direct-style terms, thus showing that introducing continuations may lead to more specialization opportunities (Section 7.3). Finally (Section 7.4) we show that if we relax the constraints of the BTA to take into account continuation-based partial evaluation, then, just like CFA, no loss and no gain of information can be observed after the introduction of continuations.

7.1 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. 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 introduction of continuations and is given in Figure 18, where the left parts are identical to the left parts of Figures 16 and 17. In the right part, we assign binding times into $\widehat{C}'_{\mathrm{bt}}$ and $\widehat{\rho}'_{\mathrm{bt}}$. As in Section 6, we use $\Phi^{\mathrm{CPS}}_{\mathrm{bt}}$ to denote the function induced by the transformation:

$$\Phi^{\mathrm{CPS}}_{\mathrm{bt}} : (\mathit{Cache}^{p}_{\mathrm{bt}} \times \mathit{Env}^{p}_{\mathrm{bt}}) \to (\mathit{Cache}^{p'}_{\mathrm{bt}} \times \mathit{Env}^{p'}_{\mathrm{bt}}).$$

7.2 Correctness of the transformation

Let us show that the solution defined in Figure 18 is indeed a valid solution of the BTA. We follow the same technique as in Section 6.2. The correctness of the transformation is established by the following theorem.

Theorem 7.1 *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 $(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}}) \models^{p}_{\mathrm{bt}} p$ holds). If $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}}) = \Phi^{\mathrm{CPS}}_{\mathrm{bt}}(\widehat{C}_{\mathrm{bt}}, \widehat{\rho}_{\mathrm{bt}})$ then $(\widehat{C}'_{\mathrm{bt}}, \widehat{\rho}'_{\mathrm{bt}}) \models^{p'}_{\mathrm{bt}} p'$ 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 6.2. For instance, the binding time for expressions in CPS is equal to the binding time of the result of the program, which, as mentioned in Section 2.6, is dynamic.

Lemma 7.2 *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 7.3 *Let e^{ℓ} be an arbitrary subexpression of p. Then $\widehat{C}_{\mathrm{bt}}(\mathcal{R}[\![e^{\ell}]\!]) = \widehat{C}_{\mathrm{bt}}(\ell)$.*

$$
\begin{aligned}
[\![e^\ell]\!]^{Pgm} &= (\lambda^\pi k.[\![e^\ell]\!]^{Exp} k)^{\ell_0} && \widehat{C}'_{\mathrm{bt}}(\ell_0) = \widehat{\rho}'_{\mathrm{bt}}(k) = \mathbf{D} \\
[\![n^\ell]\!]^{Triv} &= n^\ell && \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
[\![x^\ell]\!]^{Triv} &= x^\ell && \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
[\![(\lambda^\pi x.e^{\ell_0})^\ell]\!]^{Triv} &= (\lambda^\pi x.(\lambda^{\pi_1} k.[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell) \quad \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
&\qquad \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{\rho}'_{\mathrm{bt}}(k) = \widehat{C}_{\mathrm{bt}}(\ell)
\end{aligned}
$$

$$
\begin{aligned}
[\![(\mathbf{rec}^\pi f(x).e^{\ell_0})^\ell]\!]^{Triv} &= (\lambda^\pi x.(\mathbf{rec}^{\pi_1} f(k).[\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2})^\ell \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_2) = \widehat{C}_{\mathrm{bt}}(\ell) \quad \widehat{C}'_{\mathrm{bt}}(\ell) = \widehat{C}_{\mathrm{bt}}(\ell) \\
&\qquad \widehat{\rho}'_{\mathrm{bt}}(f) = \widehat{\rho}_{\mathrm{bt}}(f) \quad \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{\rho}'_{\mathrm{bt}}(k) = \widehat{C}_{\mathrm{bt}}(\ell)
\end{aligned}
$$

$$
\begin{aligned}
[\![t^\ell]\!]^{Exp} k &= (\mathbf{let}\ x = k^{\ell_0}\ [\![t^\ell]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2} \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_0) = \widehat{\rho}'_{\mathrm{bt}}(k) \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_2) = \widehat{C}'_{\mathrm{bt}}(\ell_1) = \widehat{\rho}_{\mathrm{bt}}(x) = \mathbf{D}
\end{aligned}
$$

$$
\begin{aligned}
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = t_0^{\ell_0}\ t_1^{\ell_1} \\ \ \mathbf{in}\ e^\ell)^{\ell_2}\end{array}\right]\!\!\right]^{Exp} k &= \begin{array}{l}(\mathbf{let}\ x_0 = [\![t_0^{\ell_0}]\!]^{Triv}\ [\![t_1^{\ell_1}]\!]^{Triv}\ \mathbf{in} \\ \ (\mathbf{let}\ x_1 = x_0^{\ell_3}\ (\lambda^\pi x.[\![e^\ell]\!]^{Exp} k)^{\ell_4}\ \mathbf{in}\ x_1^{\ell_5})^{\ell_6})^{\ell_7}\end{array} \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_4) = \widehat{C}'_{\mathrm{bt}}(\ell_3) = \widehat{\rho}'_{\mathrm{bt}}(x_0) = \widehat{C}_{\mathrm{bt}}(\ell_0) \\
&\qquad \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{\rho}'_{\mathrm{bt}}(x_1) = \mathbf{D} \\
&\qquad \widehat{C}'_{\mathrm{bt}}(\ell_7) = \widehat{C}'_{\mathrm{bt}}(\ell_6) = \widehat{C}'_{\mathrm{bt}}(\ell_5) = \mathbf{D}
\end{aligned}
$$

$$
\begin{aligned}
\left[\!\!\left[\begin{array}{l}(\mathbf{let}\ x = op(t^\ell) \\ \ \mathbf{in}\ e^{\ell_0})^{\ell_1}\end{array}\right]\!\!\right]^{Exp} k &= (\mathbf{let}\ x = op([\![t^\ell]\!]^{Triv})\ \mathbf{in}\ [\![e^{\ell_0}]\!]^{Exp} k)^{\ell_2} \\
&\qquad \widehat{\rho}'_{\mathrm{bt}}(x) = \widehat{\rho}_{\mathrm{bt}}(x) \quad \widehat{C}'_{\mathrm{bt}}(\ell_2) = \mathbf{D}
\end{aligned}
$$

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

Figure 18: Transformation of binding times from direct style to CPS

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.

Lemma 7.4 *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 $\lambda^\pi x.e^{\ell_1} \in \widehat{\rho}'_{\mathrm{cf}}(k)$, $\widehat{C}_{\mathrm{bt}}(\mathcal{R}[\![e^{\ell_0}]\!]) = \widehat{\rho}'_{\mathrm{bt}}(x)$. Let k be a continuation *identifier introduced by the transformation of a recursive function definition from p:*

$$[\![\mathbf{rec}^{\pi_1} f(x_1).e^{\ell_0}]\!]^{\mathit{Triv}} = \mathbf{rec}^{\pi_1} f(x_1).\lambda k.[\![e^{\ell_0}]\!]^{\mathit{Exp}} k$$

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 7.1 is sketched in Appendix A.

Theorem 7.1 and Lemma 7.2 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. This preservation implies that no values are forced to be dynamic just by introducing continuations. 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 introducing continuations.

7.3 Reversing the transformation

We 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 [16], where *succ* is the successor function, and the free variable f and z are considered to be dynamic (f might denote a potentially diverging function):

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

In the least solution of the BTA on this term, even if the application of $\lambda^\pi y.\ldots$ to 1 is classified as static, its result is classified as dynamic because of the dynamic application in the header of its inner let-expression. Thus r is dynamic. Since the second increment operation depends on r, it is dynamic as well. Simply discarding the dynamic computation $f\ z$ is not meaning-preserving since the computation may diverge.

The CPS counterpart of the canonical example above reads as follows (without embedding it into direct style, for readability):

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

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:

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

It is true that one benefits from such an improvement only by allowing code duplication. But the code duplication takes place at specialization time, not at BTA time. Thus in contrast to Sabry and Felleisen's analysis [39], the improvement in precision is not due to duplicating the analysis on the two branches.

7.4 Continuation-based partial evaluation

In the two examples above the binding-time improvements come from the context coherence 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.

In this section, we show that these contextual coherence constraints are the only ones leading to binding-time improvements. Using the same proof technique as in Section 6, we formally show that introducing continuations 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 $\Phi^{\text{CPS}}_{\text{bt}^\star}$, the CPS transformation of the binding times obtained by BTA*. The definition is only a slight modification of the definition of $\Phi^{\text{CPS}}_{\text{bt}}$ in Section 7.1. Given the program p and a solution $(\widehat{C}_{\text{bt}^\star}, \widehat{\rho}_{\text{bt}^\star})$ of BTA* (i.e., such that $(\widehat{C}_{\text{bt}^\star}, \widehat{\rho}_{\text{bt}^\star}) \models^{p}_{\text{bt}^\star} p$ holds), we can show that $\Phi^{\text{CPS}}_{\text{bt}^\star}(\widehat{C}_{\text{bt}^\star}, \widehat{\rho}_{\text{bt}^\star}) \models^{p'}_{\text{bt}^\star} p'$ holds. We can also define the reverse binding-time transformation $\Psi^{\text{CPS}}_{\text{bt}^\star}$, which is essentially the same as the reverse flow transformation of Section 6.3 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 $(\widehat{C}'_{\text{bt}^\star}, \widehat{\rho}'_{\text{bt}^\star})$ of BTA* for p' (i.e., such that $(\widehat{C}'_{\text{bt}^\star}, \widehat{\rho}'_{\text{bt}^\star}) \models^{p'}_{\text{bt}^\star} p'$ holds), $\Psi^{\text{CPS}}_{\text{bt}^\star}(\widehat{C}'_{\text{bt}^\star}, \widehat{\rho}'_{\text{bt}^\star}) \models^{p}_{\text{bt}^\star} p$ holds too.

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

Theorem 7.5 *Given a direct-style program p and its CPS counterpart $p' = [\![p]\!]^{Pgm}$, let $(\widehat{C}_{\text{bt}^\star}, \widehat{\rho}_{\text{bt}^\star})$ be the least solution of BTA* for p and let $(\widehat{C}'_{\text{bt}^\star}, \widehat{\rho}'_{\text{bt}^\star})$ be the least solution of BTA* for p'. Then for all variables x in p, $\widehat{\rho}_{\text{bt}^\star}(x) = \widehat{\rho}'_{\text{bt}^\star}(x)$ and for all trivial terms t^ℓ in p, $\widehat{C}_{\text{bt}^\star}(\ell) = \widehat{C}'_{\text{bt}^\star}(\ell)$.*

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

7.5 Summary and conclusions

We have shown that, given an input program as a call-by-value encoding of a Λ-program, introducing continuations does not degrade and may improve the results of the BTA for traditional partial-evaluation. We have also shown that introducing continuations does not affect the results of the BTA for continuation-based partial evaluation.

We therefore conclude that, unless one is willing to use continuation-based partial evaluation, a complete CPS transformation of the program is beneficial to the quality of the results of the BTA.

8 Related work

8.1 Program analysis in general

Even though the issue of syntactic accidents is not treated in textbooks and tutorials on program analysis, it appears to be folklore in the program-analysis community. An outstanding recent example is region inference (Section 8.1.1). To some extent, a similar situation occurs in programming practice: who has never modified a program with the sole purpose of improving its performance?

We are only aware of three other studies of the effect of continuations on program analysis: an early work by Nielson [29], Sabry and Felleisen's PLDI'94 paper [39], and Palsberg and Wand's recent work [37].

Nielson's work compares the precision of two data-flow analyses: one based on a direct-style semantics and the other on a continuation semantics. In contrast, we compare the precision of the (same) analysis of a program and of its CPS counterpart. Sabry and Felleisen's work shows that a CPS transformation leads to incomparable results for a constant propagation analysis (Section 8.1.2). Palsberg and Wand's work is similar to ours since it involves a CPS transformation of flow information (Section 8.1.3).

8.1.1 Region inference and the CPS transformation

Region inference [44] aims at detecting program points where run-time storage can be deallocated—typically at exit points for blocks and at return points for functions. To overcome syntactic accidents, a programming discipline has therefore been developed to make region inference yield better results.

We note that region improvements and binding-time improvements may come at cross purpose. For example, consider let reassociation:

$$\begin{array}{l} \mathbf{let}\ x_2 = \mathbf{let}\ x_1 = e^{\ell_1} \\ \qquad\qquad\quad \mathbf{in}\ e^{\ell_2} \\ \mathbf{in}\ e^{\ell_3} \end{array} \quad \begin{array}{c} \xrightarrow{\text{let flattening}} \\ \\ \xleftarrow[\text{let ``deepening''}]{} \end{array} \quad \begin{array}{l} \mathbf{let}\ x_1 = e^{\ell_1} \\ \mathbf{in}\ \mathbf{let}\ x_2 = e^{\ell_2} \\ \quad\ \mathbf{in}\ e^{\ell_3} \end{array}$$

Let flattening allows the region for x_1 to be released after the region for x_2. Let deepening allows the region for x_1 to be released earlier and requires the region for x_2 to be allocated earlier. Therefore, let deepening provides a region improvement, especially if e^{ℓ_3} contains a recursive call. But on the other hand, and as pointed out by an anonymous reviewer, if e^{ℓ_1} contains a recursive call, it is let flattening that provides a region improvement. Similarly, for functions, the CPS transformation yields a binding-time improvement whereas the direct-style transformation yields a region improvement (since in CPS, functions "never return").

8.1.2 Data-flow analysis and the CPS transformation

In their PLDI'94 paper [39], Sabry and Felleisen have shown that after a CPS transformation, a data-flow analysis may confuse the continuations used at return points, as already noted by Shivers [41, page 33]. An example of confusion of return points is given by the term

$$\begin{array}{l}\textbf{let } x_1 = f\ 1\\ \textbf{in let } x_2 = f\ 2\\ \quad\textbf{in } x_1\end{array}$$

and its CPS counterpart

$$\lambda k.f\ 1\ (\lambda^{\pi_1} x_1.f\ 2\ (\lambda^{\pi_2} x_2.k\ x_1))$$

analyzed in contexts where f is bound to $\lambda x.x$ and to its CPS counterpart $\lambda x.\lambda k_1.k_1\ x$, respectively. The analysis of the direct-style term starts by examining the first application and detects that x and afterwards x_1 evaluate to the constant 1. Then, by analyzing the second application, the analysis approximates that the value of x is not constant (it can evaluate to both 1 and 2). The value of x_2 is also considered unknown. Nevertheless, x_1 is still considered constant, and the analysis is able to deduce that the whole expression evaluates to the constant 1.

In the CPS program, the analysis of the first application determines that the continuation k_1 evaluates to π_1, and, afterwards, that x_1 evaluates to 1. After the analysis of the second application, the continuation k_1 evaluates to both π_1 and π_2. The variable x evaluates to both 1 and 2 and is approximated as unknown. The approximation is passed by the application $k_1\ x$, into both x_1 and x_2. Therefore, a loss of precision occurs: the result of the whole expression is no longer detected as being a constant.

One can observe, however, that in a constant-propagation analysis the chronological order of the two applications may affect the result. In direct style, the first application of the function f is analyzed in a different context than the second application. Interchanging the two let bindings leads to a different result of the analysis, for an essentially equivalent program. Therefore a limited form of context dependency is built in the constant-propagation analysis considered by Sabry and Felleisen. In contrast, the constraint-based analyses (in the monovariant case) propagate the result of a function at once to all the application sites of this function. These analyses do not exhibit the sequentiality dependency of the constant propagation, and therefore, no precision is lost after a source CPS transformation.

Sabry and Felleisen also present examples where the analysis of a program is improved after the CPS transformation, reflecting that the constant-propagation analysis is not distributive [24, 29]. The improvements are attributed to the fact that the constant-propagation analysis is duplicated over conditional branches (and their corresponding continuations). In contrast, the constraint-based analyses propagate results from one branch of a conditional to another, and therefore, no precision is gained by the CPS transformation.

To summarize, Sabry and Felleisen's analysis depends on the order in which the source program is traversed and it is duplicated over conditional branches. These two properties led Sabry and Felleisen to conclude that the CPS transformation does not preserve the result of constant propagation. In contrast, our monovariant constraint-based analyses do not depend on the order in which constraints are solved and the analyses are not duplicated over conditional branches. These two properties led us to conclude that the CPS transformation does preserve the results of CFA and of BTA*.

8.1.3 CPS transformation of flow information

Recently, Palsberg and Wand have conducted a study of CFA [37], supporting Sabry and Felleisen's conclusion that the extra precision enabled by the CPS transformation is due to the duplication of the analysis. They developed a CPS transformation of flow information comparable to the one of Figure 16, 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 primary goal was to transfer Wand's pioneer results on the CPS transformation of types [26, 45] to the CPS transformation of flow types. Since then, we have shown that least solutions are preserved by administrative reductions of CPS-transformed programs [8, 9].

8.2 Binding-time analysis and the CPS transformation

Binding-time improvements have always been customary for users of binding-time analysis [22, 30]. One of them amounts to considering source programs in CPS [5, 11], which suggests that source programs should be systematically CPS-transformed [4]. (Muylaert-Filho and Burn take the same stand for strictness analysis and the call-by-name CPS transformation [28].)

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 [12], which paved the way to continuation-based partial evaluation [3, 25].

Hatcliff and Danvy have characterized the full effect of continuation-based partial evaluation as online let flattening in Moggi's computational metalanguage [16]. This characterization justifies why offline let flattening is also, partially, a binding-time improvement [19]. In any case, offline let flattening is known to be part of the CPS transformation [15].

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

9 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 (CFA) and binding-time analysis. Through a systematic construction of the CPS counterpart of flow information, we have found that constraint-based CFA is insensitive to continuation-passing, and that the CPS transformation does improve binding times for traditional partial evaluation. Using the same technique, we have also found that the binding-time analysis for continuation-based partial evaluation 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 disabling the context coherence constraints for let expressions (and for conditionals as well, 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 [16]. 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 disabling the context coherence constraints in 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, i.e., program analyses endowed with invariants under program change (be such change a particular type of reduction or other kind of meaning-preserving transformations). Henglein's invariance properties of polymorphic typing judgments with respect to let unfolding and folding and η-reduction [18] is a step in this direction. Alternatively, one could develop an intermediate language for reasoning about program analysis and program transformation.

Acknowledgments: We are grateful to Andrzej Filinski and Julia L. Lawall for substantial discussions and comments in the course of this work, and to Amr Sabry for extensive comments on the present version of this article. Thanks are also due to Torben Amtoft, Anindya Banerjee, Bernd Grobauer, Dan Hernest, Niels O. Jensen, Torben Mogensen, Lasse R. Nielsen, Morten Rhiger, Olin Shivers, Zhe Yang, and the anonymous reviewers for their feedback.

A Proofs

Proof: 1 (Proof of Theorem 6.1) The proof proceeds by induction on the transformation of p into p'. We sketch the induction steps.

We show that $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models^{p'}_{\mathrm{cf}} (\mathbf{let}\ x = k^{\ell_0}\ [\![t^{\ell}]\!]^{Triv}\ \mathbf{in}\ x^{\ell_1})^{\ell_2}$ 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 two flow constraints are satisfied.

The first constraint is $\widehat{C}'_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}'_{\mathrm{cf}}(y)$. By Lemma 6.2, $\widehat{C}'_{\mathrm{cf}}(\ell) = \widehat{C}_{\mathrm{cf}}(\ell)$. 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. Obviously $\widehat{C}_{\mathrm{cf}}(\ell) \subseteq \widehat{\rho}'_{\mathrm{cf}}(y)$. 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 6.4.

The second constraint is $\widehat{C}'_{\mathrm{cf}}(\ell_3) \subseteq \widehat{\rho}'_{\mathrm{cf}}(x)$. Following Lemma 6.2, it amounts to $\emptyset \subseteq \emptyset$.

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

Proof: 2 (Proof of Theorem 6.5) The proof is by induction on the transformation of p into p'. We sketch the induction steps.

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

For the transformation of a named application:

$$[\![\textbf{let } x = t_0^{\ell_3}\ t_1 \textbf{ in } e_2]\!]^{\mathit{Exp}} k = \begin{array}{l}\textbf{let } x_0 = [\![t_0^{\ell_3}]\!]^{\mathit{Triv}}\ [\![t_1]\!]^{\mathit{Triv}} \\ \textbf{in let } x_1 = x_0\ \lambda^{\pi} x.e^{\ell_2} \textbf{ in } x_1\end{array}$$

let $\lambda^{\pi_1} y.e_1^{\ell_4}$ be an arbitrary λ-abstraction from p such that $\pi_1 \in \widehat{C}_{\mathrm{cf}}(\ell_3)$. Let the CPS transformation of the λ-abstraction be $\lambda^{\pi_1} y.\lambda k_1.e_2$. Then $\pi \in \widehat{\rho}'_{\mathrm{cf}}(k_1)$. From Lemma 6.7 and Lemma 6.8 we obtain that $\widehat{C}_{\mathrm{cf}}(\ell_4) \subseteq \widehat{\rho}_{\mathrm{cf}}(x)$.

Proof: 3 (Proof of Theorem 7.1) The proof is an adaptation of the proof of Theorem 6.1 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 $(\widehat{C}'_{\mathrm{cf}}, \widehat{\rho}'_{\mathrm{cf}}) \models^{p'}_{\mathrm{cf}} (\textbf{let } x = k^{\ell_0}\ [\![t^\ell]\!]^{\mathit{Triv}} \textbf{ in } x^{\ell_1})^{\ell_2}$ 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 6.10, we make a case analysis on the introduction of k, using Lemma 7.3 and Lemma 7.4 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] Nick Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In Edmund M. Clarke, editor, *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Patrick Cousot and Radhia Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In Simon Peyton Jones, editor, *Proceedings of the Seventh ACM Conference on Functional Programming and Computer Architecture*, pages 170–181, La Jolla, California, June 1995. ACM Press.

[8] Daniel Damian. *On Static and Dynamic Control-Flow Information in Program Analysis and Transformation.* PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, July 2001. BRICS DS-01-5.

[9] Daniel Damian and Olivier Danvy. CPS transformation of flow information, part II: Administrative reductions. Technical Report BRICS RS-01-40, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, October 2001.

[10] Daniel Damian and Olivier Danvy. A simple CPS transformation of control-flow information. Technical Report BRICS RS-01-55, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, December 2001.

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

[12] 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.

[13] Olivier Danvy and Lasse R. Nielsen. A first-order one-pass CPS transformation. In Mogens Nielsen, editor, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, Lecture Notes in Computer Science, Grenoble, France, April 2002. Springer-Verlag. Extended version available as the technical report BRICS RS-01-49.

[14] 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.

[15] 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.

[16] 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.

[17] Nevin Heintze. Set-based program analysis of ML programs. In Talcott [43], pages 306–317.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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`.

[23] 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.

[24] John B. Kam and Jeffrey D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:305–317, 1977.

[25] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Talcott [43].

[26] 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.

[27] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[28] Juarez A. Muylaert-Filho and Geoffrey L. Burn. Continuation passing transformation and abstract interpretation. In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, Workshops in Computing Series, pages 247–259, Isle of Thorns, Sussex, 1993. Springer-Verlag.

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

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

[31] 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.

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

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

[34] 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.

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

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

[37] Jens Palsberg and Mitchell Wand. CPS transformation of flow information. Unpublished manuscript, available at `http://www.cs.purdue.edu/~palsberg/publications.html`, June 2001.

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

[39] 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.

[40] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, 1997.

[41] Olin Shivers. *Control-Flow Analysis of Higher-Order Languages or Taming Lambda.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1991. Technical Report CMU-CS-91-145.

[42] 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.

[43] 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.

[44] Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Højfeld Olesen, Peter Sestoft, and Peter Bertelsen. Programming with regions in the ML Kit. DIKU Rapport 97/12, University of Copenhagen, Copenhagen, Denmark, 1997.

[45] 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.