Basic Research in Computer Science

BRICS RS-01-55 Damian & Danvy: A Simple CPS Transformation of Control-Flow Information

A Simple CPS Transformation of Control-Flow Information

Daniel Damian
Olivier Danvy

BRICS Report Series **RS-01-55**

ISSN 0909-0878 **December 2001**

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

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

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

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/01/55/`

A Simple CPS Transformation of Control-Flow Information

Daniel Damian* and Olivier Danvy

BRICS †
Department of Computer Science
University of Aarhus ‡

December 2001

Abstract

We build on Danvy and Nielsen's first-order program transformation into continuation-passing style (CPS) to design a new CPS transformation of flow information that is simpler and more efficient than what has been presented in previous work. The key to simplicity and efficiency is that our CPS transformation constructs the flow information in one go, instead of first computing an intermediate result and then exploiting it to construct the flow information.

More precisely, we show how to compute control-flow information for CPS-transformed programs from control-flow information for direct-style programs and vice-versa. As a corollary, we confirm that CPS transformation has no effect on the control-flow information obtained by constraint-based control-flow analysis. The transformation has immediate applications in assessing the effect of the CPS transformation over other analyses such as, for instance, binding-time analysis.

Keywords

Program analysis, control-flow analysis, constraints, continuations, continuation-passing style (CPS), CPS transformation, administrative reductions, one-pass CPS transformation.

*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

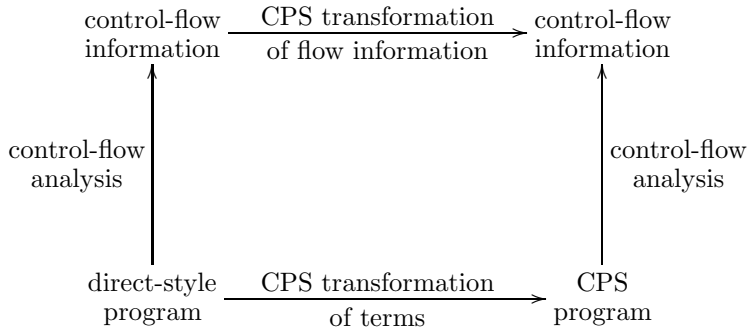
List of Figures

1 Introduction

The continuation-passing-style (CPS) transformation is a source-to-source program transformation of λ-terms that makes explicit the continuation of each λ-expression [36, 44]. Continuations have been discovered in many contexts [37] and form an active area of research [10, 39] with many applications, e.g., in compiler construction [1, 24, 43], program transformation [46], partial evaluation [19, 25], multi-processing [2, 15, 49], and, recently, goal-directed evaluation [12] and program security [50].

The call-by-value and call-by-name CPS transformations are due to Plotkin [36] and yield λ-terms that are independent on the order of evaluation. The CPS transformation has been extended to types [26, 47], which has led to the discovery of its logical content [17, 28]. Over the last two years, both Palsberg and Wand [35] and Damian and Danvy [6, 7, 9] have developed a CPS transformation of control-flow information. They have used it to show that a CPS transformation does not affect the control-flow information collected by a monovariant constraint-based control-flow analysis.

Graphically:



The canonical motivation for transferring the result of a program analysis across a program transformation is that the transfer is likely to be cheaper than analyzing the transformed program. In the present case, (1) the time complexity of control-flow analysis is cubic in the size of the analyzed program and (2) the time complexity of CPS-transforming control-flow information is linear in the size of the control-flow information, which is again linear in the size of the analyzed program.

CPS transformations of flow information are based on CPS transformations of terms.

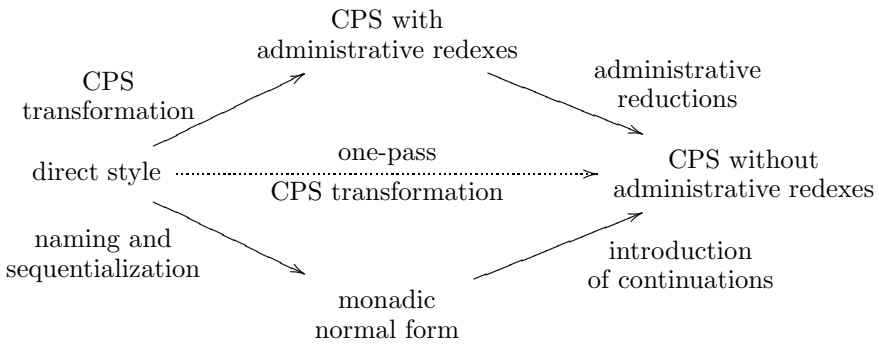
1.1 CPS transformation of terms

The CPS transformation has motivated a long line of research. Plotkin [36] and Steele [43] observed that it gives rise to large residual terms, due to so-called administrative redexes. Both theoretically and practically, these administrative redexes are in the way. For example, in his proof, Plotkin needs to interleave administrative and essential reductions. Yet a practically useful CPS-transformed program need not contain these redexes, and indeed, in his compiler, Steele performs all administrative reductions immediately after the CPS transformation. As an alternative to adminis-

trative post-reduction, compact CPS programs can also be obtained by first bringing the source program into monadic normal form and then introducing continuations [18].

Administrative redexes may be avoided altogether by using a one-pass CPS transformation. Existing one-pass CPS transformations use a higher-order accumulator [1, 11, 48] or are based on evaluation contexts [40, 42].

Graphically:



A one-pass CPS transformation makes it possible to reason directly over CPS-transformed terms. Unfortunately, existing one-pass CPS transformations are not immediate to use, either because they are higher-order or because they are not compositional. A higher-order accumulator requires a logical relation [13]. A non-compositional transformation requires well-founded induction rather than ordinary structural induction [40]. Fortunately, Danvy and Nielsen have recently discovered a one-pass CPS transformation that is both first-order and compositional [14, 30].

1.2 CPS transformation of flow information

In our initial work [7], we considered only one step of the CPS transformation, namely the introduction of continuations on terms in monadic normal form. We then turned to transforming source terms into monadic normal form [6, 9].

In a related work [35], Palsberg and Wand considered the first phase of the CPS transformation. In a followup work [6, 8], we addressed administrative reductions.

Therefore, the existing CPS transformations of flow information operate in two passes. The first pass computes an intermediate result and the second pass exploits it to construct the flow information.

In this article, we build on Danvy and Nielsen's new one-pass CPS transformation [14, 30] and we present a new and simpler CPS transformation of control-flow information that does not construct any intermediate result and thus is more efficient to use. It is also simpler to prove correct. Indeed, proving predicates defined by structural induction on a CPS-transformed program is simplest done with a first-order and compositionally-defined one-pass CPS transformation.

1.3 This work

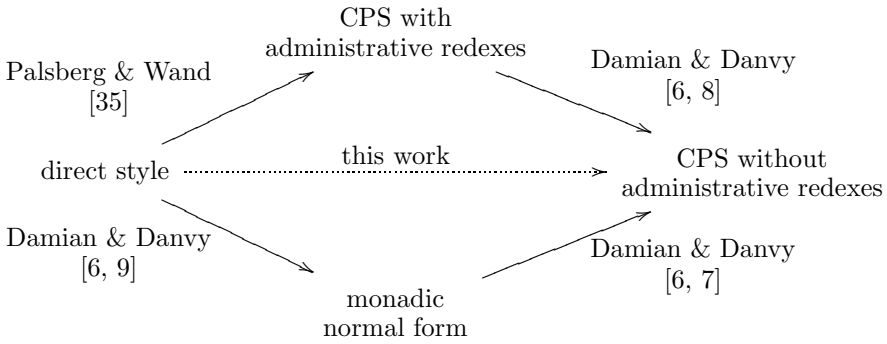
We show how to directly construct control-flow information for a CPS program after administrative reductions, without the need for an intermediate form. Our construction confirms that the CPS transformation does not affect the result of a monovariant

$e \in \mathit{Expr}$	(terms)	$e ::= s \mid t$
$s \in \mathit{Comp}$	(serious terms, i.e., computations)	$s ::= e_0^{\ell_0}\ e_1^{\ell_1}$
$t, K \in \mathit{Triv}$	(trivial terms, i.e., values)	$t ::= x \mid \lambda^{\pi} x.e^{\ell}$
$x \in \mathit{Ide}$	(identifiers)	
$\ell \in \mathit{Lab}$	(term labels)	
$\pi \in \mathit{Lam}$	(λ-abstraction labels)	

Figure 1: The language of labeled λ-terms

constraint-based control-flow analysis [6, 7, 9, 35]. It also opens the way to directly investigating the effect of the CPS transformation on other analyses, as for instance, binding-time analysis.

Graphically:



Our CPS transformation of control flow is simpler than previous versions and addresses the λ-calculus without the need for an intermediate form or administrative reductions. The proofs of correctness are similar to the ones in our earlier work, but here source terms need not be in monadic normal form. They are also slightly simpler than Palsberg and Wand's since programs contain no administrative redexes.

2 Control-flow analysis for λ-terms

2.1 The language of λ-terms

We consider the language of labeled λ-terms defined in Figure 1. Following Reynolds [38] and Moggi [27], we distinguish among trivial terms t that denote values and serious terms s that may denote computations. Expressions are annotated with distinct labels ℓ from a countable set *Lab*. Each λ-abstraction has a unique associated label π. A program p is a closed labeled expression e^{ℓ}.

2.2 Control-flow analysis

We consider a standard constraint-based control-flow analysis (CFA) on λ-terms [5, 16, 20, 21, 22, 32, 33, 34].

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

$$
\begin{array}{rll}
Triv^p & = \mathcal{P}(Lam^p) & \text{Abstract values} \\
\widehat{C} \in Cache^p & = Lab^p \to Triv^p & \text{Abstract cache} \\
\widehat{\rho} \in Env^p & = Var^p \to Triv^p & \text{Abstract environment}
\end{array}
$$

$$
\vDash^p \subseteq (Cache^p \times Env^p) \times Lab^p
$$

Figure 2: Control-flow analysis relation for a program p

$$
\begin{array}{lcl}
(\widehat{C}, \widehat{\rho}) \vDash^p x^\ell & \iff & \widehat{\rho}(x) \subseteq \widehat{C}(\ell) \\
(\widehat{C}, \widehat{\rho}) \vDash^p (\lambda^\pi x.e^\ell)^{\ell_1} & \iff & (\widehat{C}, \widehat{\rho}) \vDash^p e^\ell \wedge \pi \in \widehat{C}(\ell_1) \\
(\widehat{C}, \widehat{\rho}) \vDash^p (e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \iff & (\widehat{C}, \widehat{\rho}) \vDash^p e_0^{\ell_0} \wedge (\widehat{C}, \widehat{\rho}) \vDash^p e_1^{\ell_1} \wedge \\
 & & \forall \lambda^\pi x.e^\ell \in \widehat{C}(\ell_0).\widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \\
 & & \widehat{C}(\ell) \subseteq \widehat{C}(\ell_2)
\end{array}
$$

Figure 3: Control-flow analysis

Specifically, we consider the CFA specified in Nielson, Nielson, and Hankin's textbook [33]. Given an input program p, the functionality of the syntax-directed control-flow analysis relation $\vDash^p$ is defined in Figure 2. The analysis relation is defined inductively in Figure 3.

The relation is defined on a pair of a tuple $(\widehat{C}, \widehat{\rho})$ and a labeled expression e^ℓ. In the relation, $\widehat{C}$ is a cache mapping each expression label to a set of λ-abstractions that the expression might evaluate to, while $\widehat{\rho}$ is an environment mapping each program variable to a set of λ-abstractions that the variable might denote. It is known [33, Chapter 3] that a pair $(\widehat{C}, \widehat{\rho})$ satisfying the relation $(\widehat{C}, \widehat{\rho}) \vDash^p p$ is a safe analysis of the program p.

Given a source program p, solutions of the analysis of p always exist. The set of solutions of the analysis of p is closed under intersection: the pointwise intersection of two solutions always exists. Therefore, there exists a least solution of the analysis of p. The least solution can be computed with a standard work-list based algorithm [33, Chapter 3]. Through the rest of this article we use "the result of the analysis of p" to refer to the least result of the analysis.

2.3 Control-flow analysis: an example

An example of CFA analysis is presented in Figure 4. The (labeled) λ-term T applies the identity function to itself. The control-flow analysis from Figure 3 on the term T results in the cache/environment pair also presented in Figure 4.

We can see that the λ-abstraction π_2 is detected to flow into the variable y and from there into the variable x and as a result of the application. In the following section

$$T = ((\lambda^{\pi_1} y.(y^{\ell_1}\ y^{\ell_2})^{\ell_3})^{\ell_4}\ (\lambda^{\pi_2} x.x^{\ell_5})^{\ell_6})^{\ell_7}$$

$\widehat{C}$

ℓ_1	$\to$	$\{\pi_2\}$	ℓ_5	$\to$	$\{\pi_2\}$
ℓ_2	$\to$	$\{\pi_2\}$	ℓ_6	$\to$	$\{\pi_2\}$
ℓ_3	$\to$	$\{\pi_2\}$	ℓ_7	$\to$	$\{\pi_2\}$
ℓ_4	$\to$	$\{\pi_1\}$			

$\widehat{\rho}$

y	$\to$	$\{\pi_2\}$
x	$\to$	$\{\pi_2\}$

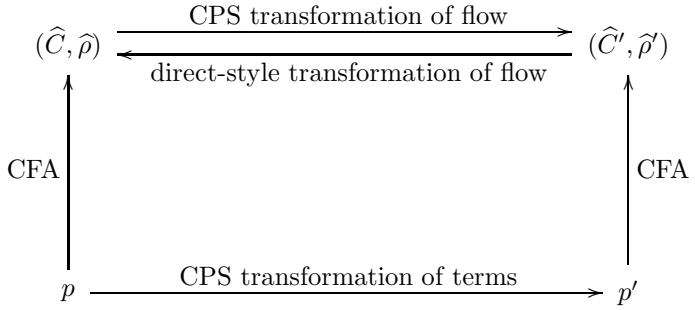
Figure 4: CFA example

we illustrate the CPS transformation of the term T and how the flow information for the resulting CPS term can be computed from the flow information for T displayed in Figure 4.

3 CPS transformation and control-flow analysis

We show that the CPS transformation preserves the result of the control-flow analysis defined in Section 2.2. To this end, we define a transformation from control-flow information for a direct-style program into control-flow information for the CPS counterpart of this program. We also define a transformation of control-flow information for a CPS-transformed program into control-flow information for the direct-style counterpart of the program. Using the monotonicity of the two transformations, we show that the least analysis of a direct-style program is equivalent to the least analysis of its CPS counterpart and vice-versa.

Graphically:



3.1 CPS transformation of terms

In this article, CPS programs are obtained using Danvy and Nielsen's first-order CPS transformation [14, 30]. The CPS transformation for (unlabeled) λ-terms is defined in Figure 5. As in our earlier work [7, 9], we consider a transformation with η-expanded tail calls: the continuation passed at a function call is always a syntactic λ-abstraction.

The CPS transformation of a program preserves all the original variables of the program. In turn, as in our earlier work [7, 9], we design the CPS transformation of labeled terms to preserve the labels of all trivial terms.

$$
\begin{array}{l}
\mathcal{E} : \mathit{Expr} \times \mathit{Ide} \to \mathit{Comp} \\
\qquad \begin{array}{rcl}
\mathcal{E}[\![t]\!]k & = & k\ \mathcal{T}[\![t]\!] \\
\mathcal{E}[\![s]\!]k & = & \mathcal{S}[\![s]\!](\lambda x.k\ x)
\end{array} \\
\\
\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \to \mathit{Comp} \\
\qquad \begin{array}{rcl}
\mathcal{S}[\![t_0\ t_1]\!]K & = & \mathcal{T}[\![t_0]\!]\ \mathcal{T}[\![t_1]\!]\ K \\
\mathcal{S}[\![t_0\ s_1]\!]K & = & \mathcal{S}[\![s_1]\!](\lambda x_1.\mathcal{T}[\![t_0]\!]\ x_1\ K) \\
\mathcal{S}[\![s_0\ t_1]\!]K & = & \mathcal{S}[\![s_0]\!](\lambda x_0.x_0\ \mathcal{T}[\![t_1]\!]\ K) \\
\mathcal{S}[\![s_0\ s_1]\!]K & = & \mathcal{S}[\![s_0]\!](\lambda x_0.\mathcal{S}[\![s_1]\!](\lambda x_1.x_0\ x_1\ K))
\end{array} \\
\\
\mathcal{T} : \mathit{Triv} \to \mathit{Triv} \\
\qquad \begin{array}{rcl}
\mathcal{T}[\![x]\!] & = & x \\
\mathcal{T}[\![\lambda x.e]\!] & = & \lambda x.\lambda k.\mathcal{E}[\![e]\!]k
\end{array}
\end{array}
$$

Figure 5: First-order one-pass CPS transformation (labels omitted)

Danvy and Nielsen's one-pass CPS transformation yields CPS terms without administrative redexes. In Section 3.2, using this CPS transformation as a syntactic support, we define the CPS transformation of control-flow information for CPS programs without administrative redexes. In Section 3.3, we define the direct-style transformation of control-flow information from CPS programs without administrative redexes. In Section 3.4, with the same technique described in the first author's PhD thesis [6, Section 2.3], the variables and labels common to the original program and to its CPS counterpart are used to establish the preservation of flow information across CPS transformation.

3.2 CPS transformation of control flow

We define a CPS transformation of control-flow information following the CPS transformation of Figure 5. Let us show how control-flow information for a direct-style term can be used to compute control-flow information for the CPS transformed program.

To transformation relies on two auxiliary functions:

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

- ξ assigns flow information to each continuation identifier k introduced by the CPS transformation of a λ-abstraction from p. This information can be obtained from the direct-style flow information, since we can syntactically identify the continuation of the CPS counterpart of any direct-style application.

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

  $$\mathcal{T}[\![\lambda^{\pi} x.e^{\ell}]\!] = \lambda^{\pi} x.\lambda k.\mathcal{E}[\![e]\!]k$$

  we define $\xi(k)$ as the union of all sets $\widehat{C}'(\ell)$ such that in the CPS transformation

$\mathcal{E} : \mathit{Expr} \times \mathit{Lab} \times \mathit{Ide} \to \mathit{Comp} \times \mathit{Lab}$

$$\begin{array}{rcl}
\mathcal{E}[\![t^\ell]\!]k & = & (k^{\ell_0}\ (\mathcal{T}[\![t]\!])^\ell)^{\ell_1} \qquad \widehat{C}'(\ell_0) = \widehat{\rho}'(k) \\
& & \widehat{C}'(\ell) = \widehat{C}'(\ell) \quad \widehat{C}'(\ell_1) = \emptyset \\
\mathcal{E}[\![s^\ell]\!]k & = & (\mathcal{S}[\![s]\!](\lambda^\pi x.(k^{\ell_0}\ x^\ell)^{\ell_1})^{\ell_2})^{\ell_3} \\
& & \widehat{C}'(\ell_0) = \widehat{\rho}'(k) \quad \widehat{C}'(\ell) = \widehat{\rho}'(x) = \widehat{C}(\ell) \\
& & \widehat{C}'(\ell_2) = \{\pi\} \quad \widehat{C}'(\ell_3) = \widehat{C}'(\ell_1) = \emptyset
\end{array}$$

$\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \times \mathit{Lab} \to \mathit{Comp}$

$$\begin{array}{rcl}
\mathcal{S}[\![t_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell & = & ((\mathcal{T}[\![t_0]\!])^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell \\
& & \widehat{C}'(\ell_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{C}(\ell_1) \\
& & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
\mathcal{S}[\![t_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_1]\!](\lambda^\pi x_1.(((\mathcal{T}[\![t_0]\!])^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} \\
& & \widehat{C}'(\ell_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{\rho}'(x_1) = \widehat{C}(\ell_1) \\
& & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
& & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi\} \\
\mathcal{S}[\![s_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.((x_0^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} \\
& & \widehat{C}'(\ell_0) = \widehat{\rho}'(x_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{C}(\ell_1) \\
& & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
& & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi\} \\
\mathcal{S}[\![s_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.(\mathcal{S}[\![s_1]\!](\lambda^{\pi_1} x_1.((x_0^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4})^{\ell_5})^{\ell_6} \\
& & \widehat{C}'(\ell_0) = \widehat{\rho}'(x_0) = \widehat{C}(\ell_0) \\
& & \widehat{C}'(\ell_1) = \widehat{\rho}'(x_1) = \widehat{C}(\ell_1) \quad \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
& & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi_1\} \\
& & \widehat{C}'(\ell_5) = \emptyset \quad \widehat{C}'(\ell_6) = \{\pi\}
\end{array}$$

$\mathcal{T} : \mathit{Triv} \to \mathit{Triv}$

$$\begin{array}{rcll}
\mathcal{T}[\![x]\!] & = & x & \\
\mathcal{T}[\![\lambda^\pi x.e^\ell]\!] & = & \lambda^\pi x.(\lambda^{\pi_1} k.\mathcal{E}[\![e^\ell]\!]k)^{\ell_0} & \widehat{C}'(\ell_0) = \{\pi_1\} \quad \widehat{\rho}'(k) = \xi(k)
\end{array}$$

Figure 6: Transformation of control flow from direct style to CPS

of p into p' there exists a transformation step

$$\mathcal{S}[\![e_0^{\ell_0}\ e_1^{\ell_1}]\!]K^\ell = \ldots$$

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

We construct the CPS control-flow information in two steps. First, in a recursive descent on the tree of the transformation, we compute $\widehat{C}'(\ell)$ for each label ℓ attached on the newly introduced λ-abstractions (continuations) and we construct the function ξ.

The second step consists of another recursive descent on the tree of the transformation. We assign control-flow information recursively, as defined for each step in Figure 6. At each transformation step, on the right-hand side, we construct the labeled CPS term corresponding to the left-hand side. We then assign flow information for each fresh label or variable. Trivial terms preserve their label and their flow

information. Flow information for serious terms is transferred through calls to continuations. Fresh continuation identifiers are assigned flow information as computed by the ξ function.

Note that in contrast to the CPS transformation of unlabeled terms of Figure 5, the transformation of labeled serious terms takes an extra argument, namely the label of the syntactic continuation being passed as an argument. At each case in Figure 6, we do not make the label explicit: we rather place it directly over the constructed continuation. Similarly, the CPS transformation of a labeled expression returns a serious term and its enclosing label.

The CPS transformation of control flow is therefore defined as a monotone function:

$$\Phi_{\mathrm{cf}}^{\mathrm{CPS}} : (Cache^{p} \times Env^{p}) \rightarrow (Cache^{p'} \times Env^{p'}).$$

Theorem 3.1. *Let $p = e^{\ell}$ be a uniquely labeled program. If $(\widehat{C}, \widehat{\rho}) \vDash^{p} e^{\ell}$ then $(\Phi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}, \widehat{\rho})) \vDash^{p'} \lambda^{\pi} k.\mathcal{E}[\![e^{\ell}]\!]k$.*

Proof. By structural induction on the resulting CPS program. The proof is similar with the proof of Theorem 6.1 of our previous work [9] which addressed terms in monadic normal form. In this proof, however, the main induction predicate states that a CPS-transformed serious term satisfies the relation when the term passed as a continuation is also satisfying the relation. The main induction predicate relies on:

a) an auxiliary predicate stating that the translation of a trivial term together with its associated label satisfies the flow constraints in CPS if the associated label is preserved;

b) an auxiliary predicate stating that the translation of an expression with its syntactic continuation satisfies the flow constraints in CPS.

At each iteration step we make use of an auxiliary Lemma (similar to Lemma 6.4 of the same previous work [9]) stating that the flow information extracted at an application point is passed into each possible continuation for the CPS equivalent of the application. □

The proof is also slightly simpler than Palsberg and Wand's proof [35] since programs contain no administrative redexes.

3.3 Direct-style transformation of control flow

The CPS transformation of flow from Figure 6 shows that the analysis of a CPS-transformed term can be at least as good as the analysis of the direct-style original term. The resulting CPS solution is the equivalent of the direct-style one, but may not be the best. We show that the direct-style and CPS analysis results are equivalent by exhibiting a direct-style transformation of flow.

We thus define a direct-style transformation of control-flow information. In other words, we transform control-flow information for the CPS-transformed term into control-flow information for the original direct-style term. The transformation is defined recursively in Figure 7. At each transformation step, on the right-hand side we construct flow information $(\widehat{C}, \widehat{\rho})$ for the direct-style program from the flow information $(\widehat{C}', \widehat{\rho}')$ for the CPS program.

$$
\begin{array}{l}
\mathcal{E} : \mathit{Expr} \times \mathit{Ide} \to \mathit{Comp} \times \mathit{Lab} \\
\quad \mathcal{E}[\![t^\ell]\!]k = (k^{\ell_0}\ (\mathcal{T}[\![t]\!])^\ell)^{\ell_1} \qquad \widehat{C}(\ell) = \widehat{C}'(\ell) \quad \widehat{C}(\ell_1) = \emptyset \\
\quad \mathcal{E}[\![s^\ell]\!]k = (\mathcal{S}[\![s]\!](\lambda^\pi x.(k^{\ell_0}\ x^\ell)^{\ell_1})^{\ell_2})^{\ell_3} \qquad \widehat{C}(\ell) = \widehat{C}'(\ell) \\
\\
\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \times \mathit{Lab} \to \mathit{Comp} \\
\quad \mathcal{S}[\![t_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell = ((\mathcal{T}[\![t_0]\!])^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{C}'(\ell_0) \quad \widehat{C}(\ell_1) = \widehat{C}'(\ell_1) \\
\quad \mathcal{S}[\![t_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell = \mathcal{S}[\![s_1]\!](\lambda^\pi x_1.(((\mathcal{T}[\![t_0]\!])^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{C}'(\ell_0) \quad \widehat{C}(\ell_1) = \widehat{\rho}'(x_1) \\
\quad \mathcal{S}[\![s_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell = \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.((x_0^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{\rho}'(x_0) \quad \widehat{C}(\ell_1) = \widehat{C}'(\ell_1) \\
\quad \mathcal{S}[\![s_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell = \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.(\mathcal{S}[\![s_1]\!](\lambda^{\pi_1} x_1.((x_0^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4})^{\ell_5})^{\ell_6} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{\rho}'(x_0) \quad \widehat{C}(\ell_1) = \widehat{\rho}'(x_1) \\
\\
\mathcal{T} : \mathit{Triv} \to \mathit{Triv} \\
\quad \mathcal{T}[\![x]\!] = x \\
\quad \mathcal{T}[\![\lambda^\pi x.e^\ell]\!] = \lambda^\pi x.(\lambda^{\pi_1} k.\mathcal{E}[\![e^\ell]\!]k)^{\ell_0}
\end{array}
$$

Figure 7: Transformation of control flow from CPS into direct style

Since at each function call the continuation is an explicit syntactic continuation, we are able to determine the control-flow information returned by each expression. In particular, at a transformation step

$$\mathcal{E}[\![s^\ell]\!]k = (\mathcal{S}[\![s]\!](\lambda^\pi x.(k^{\ell_0}\ x^\ell)^{\ell_1})^{\ell_2})^{\ell_3}$$

we are able to assign control-flow information for the return label ℓ from the control-flow information collected by the continuation $\lambda^\pi x.(k^{\ell_0}\ x^\ell)^{\ell_1}$.

Control-flow information can therefore be constructed bottom-up. The direct-style transformation of control flow is thus defined as a monotone function:

$$\Psi_{\mathrm{cf}}^{\mathrm{CPS}} : (\mathit{Cache}^{p'} \times \mathit{Env}^{p'}) \to (\mathit{Cache}^{p} \times \mathit{Env}^{p})$$

Theorem 3.2. *Let $p = e^\ell$ be a uniquely labeled program.*
If $(\widehat{C}', \widehat{\rho}') \vDash^p \lambda^\pi k.\mathcal{E}[\![e^\ell]\!]k$ and $\widehat{\rho}'(k) = \emptyset$, then $\Psi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}', \widehat{\rho}') \vDash^{p'} e^\ell$.

Proof. By structural induction on the direct-style source program. Again, the proof is similar to the proof of Theorem 6.5 of our earlier work [9]. In this proof the main induction predicate states that the constructed solution satisfies the flow constraints for any serious sub-term considered together with its enclosing label. The proof relies on:

a) an auxiliary predicate stating that a trivial term together with its associated label satisfies the flow constraints if it satisfies the constraints in CPS, considered together with its associated label;

b) an auxiliary predicate stating that an expression satisfies the flow constraints if the translation satisfies the flow constraints in CPS (considered together with its syntactic continuation).

At each iteration step we make use of an auxiliary Lemma (similar to Lemma 6.8 of the same previous work [9]) stating that the flow information extracted at an application point includes the flow information collected by each possible continuation for the CPS equivalent of the application. □

3.4 Preservation of flow

Following the construction of the CPS control-flow information in Figure 6, it is immediate to see that the flow information assigned to the program's original variables in CPS is identical to the one extracted from the direct-style original program. The same is valid for the reverse transformation of Figure 7: the control-flow information assigned to direct-style variables is identical to the one extracted from the CPS program.

Theorem 3.3 follows from the monotonicity of the two transformations of control flow.

Theorem 3.3. *Let p be a direct-style program and p' its CPS counterpart.*

i) Let $(\widehat{C}, \widehat{\rho})$ be the solution of the control-flow analysis of p. Then $\Psi_{\mathrm{cf}}^{\mathrm{CPS}}(\Phi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}, \widehat{\rho})) = (\widehat{C}, \widehat{\rho})$.

ii) Let $(\widehat{C}', \widehat{\rho}')$ be the solution of the control-flow analysis of p'. Then $\Phi_{\mathrm{cf}}^{\mathrm{CPS}}(\Psi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}', \widehat{\rho}')) = (\widehat{C}', \widehat{\rho}')$.

3.5 CPS transformation of flow: an example

Let us now consider the CPS transformation of the term T in the example of Section 2.3. The CPS equivalent T' of the term T is illustrated in Figure 8. Even if the term T' is administratively reduced, the number of labels becomes difficult to manage without an automated calculation. The generated example illustrates the equivalence of flow information obtained by the CFA analysis of the original term T and of the CPS term T'.

As specified in Section 1.1, the CPS term T' maintains all the λ-abstraction labels and trivial-term labels of the original term T'. As specified by Theorem 3.3, the flow information associated to the labels of the trivial terms (i.e., $\ell_1, \ell_2, \ell_4, \ell_5$ and ℓ_6) are identical. Similarly, the variables of the original term (x and y) are preserved and their associated flow information is identical. We can observe that the labels ℓ_3 and ℓ_7 have disappeared, their associated flow information being transferred into the variables abstracted by continuations v_2 and v_1 respectively. The remainder of the labels are either final answer labels, and their associated flow information is empty, either labels surrounding a continuation in which case the associated flow information is a singleton containing the label of the continuation.

Therefore, given the flow information from Figure 4 we can avoid re-analyzing the CPS term T' by computing the flow information of Figure 8 according to the transformation function $\Phi_{\mathrm{cf}}^{\mathrm{CPS}}$, with a provably lower complexity. Similarly, given the flow information of Figure 8, we can avoid re-analyzing the CPS term T' by computing the flow information of Figure 4 according to the transformation function $\Phi_{\mathrm{cf}}^{\mathrm{CPS}}$, again with a provably lower complexity.

$$
\begin{array}{rcl}
T' & = & (((\lambda^{\pi_1} y.(\lambda^{\pi_4} k_2.((y^{\ell_1}\ y^{\ell_2})^{\ell_{16}}\ (\lambda^{\pi_5} v_2.(k_2^{\ell_{12}}\ v_2^{\ell_{13}})^{\ell_{14}})^{\ell_{15}})^{\ell_{17}})^{\ell_{18}})^{\ell_4} \\
 & & \quad (\lambda^{\pi_2} x.(\lambda^{\pi_6} k_3.(k_3^{\ell_{19}}\ x^{\ell_5})^{\ell_{20}})^{\ell_{21}})^{\ell_6})^{\ell_{22}} \\
 & & (\lambda^{\pi_3} v_1.(k_1^{\ell_8}\ v_1^{\ell_9})^{\ell_{10}})^{\ell_{11}})^{\ell_{23}}
\end{array}
$$

ξ

k_2	$\{\pi_3\}$
k_3	$\{\pi_5\}$

$\widehat{C}$

ℓ_1	$\{\pi_2\}$	ℓ_{10}	$\{\}$	ℓ_{17}	$\{\}$
ℓ_2	$\{\pi_2\}$	ℓ_{11}	$\{\pi_3\}$	ℓ_{18}	$\{\pi_4\}$
ℓ_4	$\{\pi_1\}$	ℓ_{12}	$\{\pi_3\}$	ℓ_{19}	$\{\pi_5\}$
ℓ_5	$\{\pi_2\}$	ℓ_{13}	$\{\pi_2\}$	ℓ_{20}	$\{\}$
ℓ_6	$\{\pi_2\}$	ℓ_{14}	$\{\}$	ℓ_{21}	$\{\pi_6\}$
ℓ_8	$\{\}$	ℓ_{15}	$\{\pi_5\}$	ℓ_{22}	$\{\pi_4\}$
ℓ_9	$\{\pi_2\}$	ℓ_{16}	$\{\pi_6\}$	ℓ_{23}	$\{\}$

$\widehat{\rho}$

k_1	$\{\}$
k_2	$\{\pi_3\}$
k_3	$\{\pi_5\}$
x	$\{\pi_2\}$
y	$\{\pi_2\}$
v_1	$\{\pi_2\}$
v_2	$\{\pi_2\}$

Figure 8: CPS transformation and analysis result

4 Conclusions and future work

We have presented a one-pass CPS transformation of control-flow information. Our transformation improves both on our earlier CPS transformation and on Palsberg and Wand's which operate in two passes. This line of work aims at transferring the results of program analyses across program transformations as an alternative to analyzing transformed programs from scratch. The interaction between CPS and program analysis has been explored by a number of authors [3, 4, 19, 23, 25, 29, 31], sometimes leading to mixed results [41].

The complete CPS transformation of control flow can be used to assess the impact of the CPS transformation on the result of other program analyses, e.g., binding-times analysis. In a previous work [7, 9], we have shown that introducing continuations (1) does not worsen and (2) can improve the results of the standard binding-time analysis for traditional partial evaluation [23]. Transforming programs into monadic normal form can also lead to further binding-time improvements [6, 19]. Our initial investigations show that the current transformation of control flow can be used to characterize in one single theorem the binding-time improvements obtained by the CPS transformation [6].

Let us finish on the relation between tail-call optimization and control-flow analysis. In the CPS transformation of Figure 5, the η-expanded tail calls provide an explicit continuation for each function call for which we can extract control-flow information. More precisely, the CPS transformation of an expression introduces an explicit continuation:

$$\mathcal{E}[\![s]\!]k = \mathcal{S}[\![s]\!](\lambda x.k\ x)$$

The presence of such an explicit continuation facilitates the definition of the CPS transformation of control flow. We are currently investigating whether η-reducing these tail-calls (i.e., defining $\mathcal{E}[\![s]\!]k$ as $\mathcal{S}[\![s]\!]k$) also preserves control-flow information.

References

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

[2] Edoardo Biagioni, Ken Cline, Peter Lee, Chris Okasaki, and Chris Stone. Safe-for-space threads in Standard ML. *Higher-Order and Symbolic Computation*, 11(2):209–225, 1998.

[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] 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.

[6] 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.

[7] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. In Philip Wadler, editor, *Proceedings of the 2000 ACM SIGPLAN International Conference on Functional Programming*, SIGPLAN Notices, Vol. 35, No. 9, pages 209–220, Montréal, Canada, September 2000. ACM Press. Extended version to appear in the Journal of Functional Programming.

[8] 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.

[9] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. *Journal of Functional Programming*, 2002. To appear. Extended version available as the technical report BRICS-RS-01-54.

[10] Olivier Danvy, editor. *Proceedings of the Second ACM SIGPLAN Workshop on Continuations*, Technical report BRICS-NS-96-13, University of Aarhus, Paris, France, January 1997.

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

[12] Olivier Danvy, Bernd Grobauer, and Morten Rhiger. A unifying approach to goal-directed evaluation. *New Generation Computing*, 20(1):53–73, 2002. Preliminary version available in the proceedings of SAIG 2001 (LNCS 2196). Extended version available as the technical report BRICS RS-01-29.

[13] Olivier Danvy and Lasse R. Nielsen. A higher-order colon translation. In Herbert Kuchen and Kazunori Ueda, editors, *Fifth International Symposium on Functional and Logic Programming*, number 2024 in Lecture Notes in Computer Science, pages 78–91, Tokyo, Japan, March 2001. Springer-Verlag. Extended version available as the technical report BRICS RS-00-33.

[14] 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*, number 2303 in Lecture Notes in Computer Science, Grenoble, France, April 2002. Springer-Verlag. Extended version available as the technical report BRICS RS-01-49.

[15] Richard P. Draves, Brian N. Bershad, Richard F. Rashid, and Randall W. Dean. Using continuations to implement thread management and communication in operating systems. Technical Report CMU-CS-91-115, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, October 1991. Also appears in the proceedings of the Thirteenth Symposium on Operating Systems Principles (SOSP), Asilomar, California, October 1991.

[16] 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.

[17] Timothy G. Griffin. A formulae-as-types notion of control. In Paul Hudak, editor, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, January 1990. ACM Press.

[18] 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.

[19] 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.

[20] Nevin Heintze. Set-based program analysis of ML programs. In Talcott [45], pages 306–317.

[21] Fritz Henglein. Simple closure analysis. Technical Report Semantics Report D-193, DIKU, Computer Science Department, University of Copenhagen, 1992.

[22] 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.

[23] 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`.

[24] David Kranz, Richard Kesley, Jonathan Rees, Paul Hudak, Jonathan Philbin, and Norman Adams. Orbit: An optimizing compiler for Scheme. In Stuart I. Feldman, editor, *Proceedings of the 1986 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 21, No 7, pages 219–233, Palo Alto, California, June 1986. ACM Press.

[25] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Talcott [45].

[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] Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1990.

[29] 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.

[30] Lasse R. Nielsen. *A study of defunctionalization and continuation-passing style*. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, July 2001. BRICS DS-01-7.

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

[32] 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.

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

[34] Jens Palsberg and Michael I. Schwartzbach. Object-oriented type inference. In *Proceedings of OOPSLA'91, the ACM SIGPLAN Sixth Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 146–161, Phoenix, Arizona, October 1991.

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

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

[37] John C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–247, 1993.

[38] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998. Reprinted from the proceedings of the 25th ACM National Conference (1972).

[39] Amr Sabry, editor. *Proceedings of the Third ACM SIGPLAN Workshop on Continuations*, Technical report 545, Computer Science Department, Indiana University, London, England, January 2001.

[40] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.

[41] 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.

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

[43] 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.

[44] Christopher Strachey and Christopher P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. *Higher-Order and Symbolic Computation*, 13(1/2):135–152, 2000. Reprint of the technical monograph PRG-11, Oxford University Computing Laboratory (1974).

[45] 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.

[46] Mitchell Wand. Continuation-based program transformation strategies. *Journal of the ACM*, 27(1):164–180, January 1980.

[47] 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.

[48] Mitchell Wand. Correctness of procedure representations in higher-order assembly language. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Proceedings of the 7th International Conference on Mathematical Foundations of Programming Semantics*, number 598 in Lecture Notes in Computer Science, pages 294–311, Pittsburgh, Pennsylvania, March 1991. Springer-Verlag.

[49] Mitchell Wand. Continuation-based multiprocessing. *Higher-Order and Symbolic Computation*, 12(3):285–299, 1999. Reprinted from the proceedings of the 1980 Lisp Conference.

[50] Steve Zdancewic and Andrew Myers. Secure information flow and CPS. In David Sands, editor, *Proceedings of the Tenth European Symposium on Programming*, number 2028 in Lecture Notes in Computer Science, pages 46–61, Genova, Italy, April 2001. Springer-Verlag.

Recent BRICS Report Series Publications

RS-01-55 **Daniel Damian and Olivier Danvy.** ***A Simple CPS Transformation of Control-Flow Information*****. December 2001. 18 pp.**

RS-01-54 **Daniel Damian and Olivier Danvy.** ***Syntactic Accidents in Program Analysis: On the Impact of the CPS Transformation*****. December 2001. 41 pp. To appear in the** ***Journal of Functional Programming*****. This report supersedes the earlier BRICS report RS-00-15.**

RS-01-53 **Zoltán Ésik and Masami Ito.** ***Temporal Logic with Cyclic Counting and the Degree of Aperiodicity of Finite Automata*****. December 2001. 31 pp.**

RS-01-52 **Jens Groth.** ***Extracting Witnesses from Proofs of Knowledge in the Random Oracle Model*****. December 2001. 23 pp.**

RS-01-51 **Ulrich Kohlenbach.** ***On Weak Markov's Principle*****. December 2001. 10 pp.**

RS-01-50 **Jiří Srba.** ***Note on the Tableau Technique for Commutative Transition Systems*****. December 2001. 19 pp. To appear in Nielsen and Engberg, editors,** ***Foundations of Software Science and Computation Structures*****, FoSSaCS '02 Proceedings, LNCS 2303, 2002.**

RS-01-49 **Olivier Danvy and Lasse R. Nielsen.** ***A First-Order One-Pass CPS Transformation*****. December 2001. 21 pp. Extended version of a paper to appear in Nielsen and Engberg, editors,** ***Foundations of Software Science and Computation Structures*****, FoSSaCS '02 Proceedings, LNCS 2303, 2002.**

RS-01-48 **Mogens Nielsen and Frank D. Valencia.** ***Temporal Concurrent Constraint Programming: Applications and Behavior*****. December 2001. 36 pp.**

RS-01-47 **Jesper Buus Nielsen.** ***Non-Committing Encryption is Too Easy in the Random Oracle Model*****. December 2001. 20 pp.**

RS-01-46 **Lars Kristiansen.** ***The Implicit Computational Complexity of Imperative Programming Languages*****. November 2001. 46 pp.**

RS-01-45 **Ivan B. Damgård and Gudmund Skovbjerg Frandsen.** ***An Extended Quadratic Frobenius Primality Test with Average Case Error Estimates*****. November 2001. 43 pp.**