Reflections on Reflections

Gilles Barthe[1], John Hatcliff[2], and Morten Heine Sørensen[3]

[1] Centrum voor Wiskunde en Informatica (CWI)[†]
[2] Computer Science Department, Oklahoma State University[‡]
[3] Department of Computer Science, University of Copenhagen (DIKU)[§]

Abstract. In the functional programming literature, compiling is often expressed as a translation between source and target program calculi. In recent work, Sabry and Wadler proposed the notion of a *reflection* as a basis for relating the source and target calculi. A reflection elegantly describes the situation where there is a kernel of the source language that is isomorphic to the target language. However, we believe that the reflection criteria is so strong that it often excludes the usual situation in compiling where one is compiling from a higher-level to a lower-level language.
We give a detailed analysis of several translations commonly used in compiling that fail to be reflections. We conclude that, in addition to the notion of reflection, there are several relations weaker a reflection that are useful for characterizing translations. We show that several familiar translations (that are not naturally reflections) form what we call a *reduction correspondence*. We introduce the more general notion of a $(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4)$-correspondence as a framework for describing relations between source and target calculi.

1 Introduction

In the functional programming literature, compiling is often expressed as a translation between source and target program calculi. The target calculus is typically based on continuation-passing style (CPS) terms [1,27], monadic-style terms [3,12], A-normal forms [10], or some other sort of intermediate language that makes explicit things such as intermediate values and closure operations. In recent work [23,24], Sabry and Wadler question: what is the appropriate relationship between source and target calculi?

Program calculi are often presented as equational theories. So one might answer the above question by stating that compiling should preserve the equality relation of the source calculi. That is,

$$M =_S N \text{ implies } M^* =_T N^* \tag{1}$$

where $=_S$ and $=_T$ denote the convertibility relation in the source and target calculi (respectively), and where M^* denotes the compilation of M. One might

[†] PO Box 94079, 1090 GB Amsterdam, The Netherlands, gilles@cwi.nl
[‡] 219 Math Sciences, Stillwater, OK, USA, 74078, hatcliff@a.cs.okstate.edu
[§] Universitetsparken 1, DK-2100 Copenhagen, Denmark, rambo@diku.dk

even require that the converse of the above implication holds (*i.e.*, that compiling preserves *and reflects* $=_S$).

However, it is usually more fruitful to focus on the reduction theories which typically underlie the equational theories. For example, one might require that

$$M \longrightarrow\!\!\!\!\!\rightarrow_S N \text{ implies } M^* \longrightarrow\!\!\!\!\!\rightarrow_T N^*. \tag{2}$$

If reductions are taken to represent computational steps or optimizations, this states that a series of computational steps in the source language can be expressed in the target language. Again, the converse may also be of interest: one would like that every optimization expressible in the target calculus be reflected in the source calculus as well. This is especially advantageous if the conceptual complexity of the target language (*e.g.*, CPS) is significantly greater than that of the source language. Such a result would allow one to reason about the optimizations in the target language while working only with the source language.

Sabry and Felleisen [22] popularized the use of a *decompiling* translation for establishing the converse of the implications. For example, for the latter implication at line (2), one might define a decompiling translation $\sharp$ which maps target language terms back to source terms and require that

$$P \longrightarrow\!\!\!\!\!\rightarrow_T Q \text{ implies } P^\sharp \longrightarrow\!\!\!\!\!\rightarrow_S Q^\sharp.$$

This, along with some other simple properties describing the interaction between $*$ and $\sharp$ is sufficient to establish the converse of line (2).

Now that there are two calculi (source and target) and two translations (compiling and decompiling) under consideration, there are are variety of ways in which these four components can be related. Sabry and Wadler sought to emphasize reduction properties over equational properties, and they adopted the topological notion of *Galois connection* as their foundation for judging the possible relationships. After noting that some Galois connections describe undesirable computational relationships, they proposed a special case of a Galois connection called a *reflection* as a basis for relating source and target calculi. They then showed that several compiling-related translations could be viewed as reflections between various source and target calculi.

However, a close examination of the definitions of Galois connection and reflection reveals what we feel are some overly strong computational properties. A reflection elegantly describes the situation where there is a kernel of the source language that is isomorphic to the target language. Yet we believe that the reflection criteria is so strong that it often excludes the usual situation in compiling where one is compiling from a higher-level to a lower-level language. In many such cases, it is simply impossible to obtain a reflection naturally.

The present work has two goals.

1. We reflect upon Sabry and Wadler's proposal of Galois connection and reflection, and give what we feel are the strengths and weaknesses of these properties as criteria for translations used in compiling. We conclude that, in addition to the notion of reflection, there are several relations weaker than

reflections that are useful for characterizing translations. We introduce the more general notion of a $(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4)$-correspondence as a framework for describing these relations.

2. To support the assessment of relations between calculi, we study several well-known translations used in compiling that fail to be reflections.
   - *Simulating call-by-name under call-by-value using thunks:* Even though this familiar compiling technique is not a reflection, we show that it does establish a very tight relationship (which we call a *reduction correspondence*) between the reduction theories of the source and target calculi. This strengthens previous results reported by Hatcliff and Danvy [13].
   - *Simulating call-by-name under call-by-value using Plotkin's CPS translation:* We note that Plotkin's well-known call-by-name (CBN) CPS translation cannot form a reflection. We give an optimizing version of the CPS translation such as one would normally use in compiling, and show that it forms a reduction correspondence. This strengthens the results given by Plotkin to include a notion of decompiling (*i.e.*, a *direct-style* translation [5]), and a description of the relationship between source and target reduction theories.
   - *Simulating call-by-name under call-by-value using a Fischer-style CPS translation:* We present a Fischer-style CBN CPS translation where, in contrast to Plotkin's translation, a continuation is passed as the first argument to a function. We show that this translation produces terms with slightly different reductions properties than those produced by Plotkin's translation. Specifically, neither a reflection, nor a reduction correspondence holds. However, there is still a strong relationship between the source and target reduction theories, and one can show that the translation forms an *equational correspondence*. We also study the reduction properties of an optimizing Fischer-style CBN CPS translation. This appears to be the first in-depth study of Fischer-style call-by-name translations.
   - *Compiling call-by-value using CPS:* We return to Sabry and Wadler's result of a reflection between the call-by-value (CBV) sublanguage of Moggi's computational meta-language and CPS terms. We note that the reflection result is obtained using a non-standard target reduction theory instead of the conventional call-by-value theory. However, we show that one can obtain naturally a reduction correspondence using the conventional theory and an optimizing translation.

   Based on this study and Sabry and Wadler's own observation that Sabry and Felleisen's rigorously justified Fischer-style CBV CPS transformation cannot give rise to a reflection, we question if *any* optimizing CPS translation appearing in the literature can be a reflection using conventional reduction theories.

The rest of the paper is organized as follows. Section 2 reviews and assesses Sabry and Wadler's definitions of Galois connection and reflection. Section 3 studies the thunk-based simulation of call-by-name under call-by-value. Section 4

studies the Plotkin and Fischer-style call-by-name CPS translation. Section 5 assesses results concerning the call-by-value CPS translations studied by Sabry and Wadler. Section 6 presents related work. Section 7 concludes.

2 Relationships Between Calculi

In this section, we review the definitions of Galois connection and reflection given by Sabry and Wadler [23]. We also adopt their introductory definitions and notation.

We write $\longrightarrow$ for compatible single-step reduction; $\longrightarrow\!\!\!\!\!\rightarrow$ for the reflexive and transitive closure of reduction; $=$ for the reflexive, transitive, and symmetric closure of reduction; and $\equiv$ for syntactic identity up to renaming of bound variables.

Assume a source calculus S with a reduction relation $\longrightarrow_S$, and a target calculus T with reduction relation $\longrightarrow_T$. Reductions are directed in such a way that they naturally correspond to evaluation steps or optimizations. Let the maps $* : S \to T$ and $\sharp : T \to S$ correspond to compiling and decompiling, respectively. Finally, let M, N range over terms of S, and P, Q range over terms of T.

2.1 Galois connection

Sabry and Wadler introduce the notion of Galois connection as the foundation for relating source and target calculi (*via* compiling and decompiling). The original topological notion of Galois connection [8] must be adapted slightly since reduction is not a partial-order but a pre-order [23].

Definition 1 (Galois connection [Sabry and Wadler]**).** Maps $*$ and $\sharp$ form a *Galois connection* from S to T whenever

$$M \longrightarrow\!\!\!\!\!\rightarrow_S P^\sharp \text{ if and only if } M^* \longrightarrow\!\!\!\!\!\rightarrow_T P.$$

There is an alternative characterization of a Galois connection.

Proposition 2 (Sabry and Wadler). *Maps $*$ and $\sharp$ form a Galois connection from S to T if and only if the following four conditions hold.*

$$\begin{array}{l}(1)\ M \longrightarrow\!\!\!\!\!\rightarrow_S M^{*\sharp},\\ (2)\ P \longleftarrow\!\!\!\!\!\!\rightarrow_T P^{\sharp *},\\ (3)\ M \longrightarrow\!\!\!\!\!\rightarrow_S N \text{ implies } M^* \longrightarrow\!\!\!\!\!\rightarrow_T N^*,\\ (4)\ P \longrightarrow\!\!\!\!\!\rightarrow_T Q \text{ implies } P^\sharp \longrightarrow\!\!\!\!\!\rightarrow_S Q^\sharp.\end{array}$$

What are the implications of using a Galois connection as a criteria for judging compiling translations? Consider the following discussion [24, p. 5].

> Consider a source term M compiling to a target term M^*, and consider an optimization $M^* \longrightarrow\!\!\!\!\!\rightarrow_T P$ in the target.
>
> 1. A Galois connection guarantees the existence of a corresponding optimization $M \longrightarrow\!\!\!\!\!\rightarrow_S P^\sharp$ in the source.

2. Recompiling this yields a reduction $M^* \longrightarrow\!\!\!\!\!\rightarrow_T P^{\sharp *}$ in the target.

Now consider the requirement of a Galois connection given in component (2) of Proposition 2.

Observation 3. *A Galois connection requires that decompiling* ($\sharp$) *followed by compiling* ($*$) *yields a term* $P^{\sharp *}$ *that can be reduced to* P.

Thinking of reductions as optimizations, this means that $P^{\sharp *}$ must be *equally or less optimized* than P. That is, a Galois connection demands that decompiling followed by compiling puts you in a position that is the same or *poorer* (less optimized) than where you started.

2.2 Reflection

Sabry and Wadler recognize that component (2) of Proposition 2 represents an undesirable quality and they make the following observation [24, p. 5].

Observation 4 (Sabry and Wadler). *If this optimization (i.e.,* $P^{\sharp *}$*) is to be at least as good as the original* [term] P*, we require that* $P \longrightarrow\!\!\!\!\!\rightarrow_T P^{\sharp *}$.

At this point there is a choice to be made regarding correcting the undesirable component (2) of Proposition 2.

1. One can simply drop the insistence that compiling and decompiling form a Galois connection, and instead require these translations to satisfy the properties of Proposition 2 *with direction of reductions in component (2) reversed*, *i.e.*, require $P \longrightarrow\!\!\!\!\!\rightarrow_T P^{\sharp *}$.
2. One can keep the requirement of Galois connection, and also require that $P \equiv P^{\sharp *}$.

Sabry and Wadler take the latter option and refine the notion of Galois connection into a *reflection*.

Definition 5 (Reflection [Sabry and Wadler]**).** Maps $*$ and $\sharp$ form a *reflection* in S of T if they form a Galois connection and $P \equiv P^{\sharp *}$.

However, we believe that the first option is equally valid. Why should a situation where $P^{\sharp *}$ is more optimized than P be considered inappropriate for compiling? We return to this point after a more detailed assessment of reflections.

What are the implications of taking *reflection* as the criteria for judging compiling translations? The requirement that $P \equiv P^{\sharp *}$ has the following implications.

Observation 6 (Sabry and Wadler). *For a reflection,* $\sharp$ *is necessarily injective.*

Observation 7. *For a reflection, compiling must be surjective on the target language.*

More generally, a reflection guarantees that there is a kernel of the source language which is isomorphic to the target language. If S is the set of source terms, then $S^{*\sharp}$ is the relevant kernel. Sabry and Wadler show that Moggi's call-by-value monad translation, Plotkin's call-by-value CPS translation, and Girard's call-by-value translation to linear logic can be made into reflections. The framework of reflections gives an elegant description of the relationship between reduction theories in these cases.

However, if the target language is *isomorphic* to a kernel of the source language, one may question if the described compilation is truly interesting. The desire for an isomorphism seems to conflict with the typical situation in compiling where one is translating from a higher-level language to a lower-level language (with more implementation details and computational steps exposed). This is the case even when compiling to a relatively high-level intermediate language such as CPS or monadic-style terms (which we will later demonstrate).

The injectivity of decompiling requires that there exists a unique expressible computational state in the high-level language for each computational state in the target language. Many rigorously justified compiling translations appearing in the literature fail to have this property. For example, in each of the translations considered later in the paper, a single computational state in the source language may be realized by several intermediate computational states in the target language (involving instantiating continuations, opening thunks or closures, *etc.*). This type of relationship (which seems the norm for compiling) is ruled out in a reflection.

Sabry and Wadler note that in this type of situation, transformations that are not naturally reflections can be *made* into reflections by changing the source and target calculi. For example, when there are several intermediate states in the target language for each state in the source language, one may be able to obtain a reflection by changing the target calculus to include multiple-step reductions that skip over intermediate states and move directly between states isomorphic to source language states (this is the case with the Plotkin CBV CPS translation discussed in Section 5). This has the advantage of yielding a tighter relationship between (modified) source and target calculi. However, if either the source or target calculi has been modified, one must still relate the modified calculus to the original calculus, and one must reprove properties such as confluence, compatibility, *etc.* The multiple-step reductions required for establishing a reflection sometimes make these properties technically more complicated than in the conventional calculi (see Sabry and Wadler's treatment of Plotkin's CBV CPS translations [24]).

2.3 Characterizing the relation between source and target calculi

When characterizing the relation between source and target reduction theories, the four-component structure of Proposition 2 is quite useful. In describing the translations appearing in the remainder of the paper, it is convenient to use the following parameterized definition.

Definition 8. Maps $*$ and $\sharp$ form a $(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4)$-correspondence between S and T if and only if the following four conditions hold.

(1) $M \mathcal{R}_1 M^{*\sharp}$
(2) $P \mathcal{R}_2 P^{\sharp *}$,
(3) $M \longrightarrow\!\!\!\!\!\rightarrow_S N$ *implies* $M^* \mathcal{R}_3 N^*$,
(4) $P \longrightarrow\!\!\!\!\!\rightarrow_T Q$ *implies* $P^\sharp \mathcal{R}_4 Q^\sharp$.

For example, a $(=_S, =_T, =_T, =_S)$-correspondence is equivalent to the notion of equational correspondence defined by Sabry and Felleisen [22], a $(\twoheadrightarrow_S, \twoheadleftarrow_T, \twoheadrightarrow_T, \twoheadrightarrow_S)$-correspondence is a Galois connection, and a $(\twoheadrightarrow_S, \equiv, \twoheadrightarrow_T, \twoheadrightarrow_S)$-correspondence is a reflection. The appropriate subscript (*i.e.*, S or T) is determined by the particular component of the correspondence, and we will often drop the subscripts on relations when no ambiguity results.

We noted earlier that the refining of Galois connection to reflection is one solution to the problematic component (2) of Proposition 2. We believe that reversing the direction of reductions in component (2) is equally justifiable and perhaps even more natural. Following this latter path leads to what we call a *reduction correspondence.*

Definition 9. A reduction correspondence is a $(\twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$-correspondence.

Note that a reflection is intuitively the *intersection* of a Galois connection and a reduction correspondence.

Proposition 10. *If maps* $* : S \to T$ *and* $\sharp : T \to S$ *form a reflection, then they form a Galois connection and a reduction correspondence.*

3 Compiling CBN to CBV Using Thunks

To give a more concrete assessment of the relations of the previous section, we consider one of the oldest techniques in compiling: the implementation of call-by-name under call-by-value using thunks [15]. Here, we follow the presentation given in [13].

Figure 1 presents the source language Λ — the untyped λ-calculus under the theory of β-reduction. Figure 2 presents the target language Λ_τ — the untyped λ-calculus plus the suspension operators `delay` and `force` under the theory of β_{v}-reduction and thunk evaluation (τ-reduction). Intuitively, `delay` M suspends the computation of an expression M, and `force` M forces the evaluation of the suspension yielded by the evaluation of M. The notion of reduction $\beta_{\mathrm{v}}\tau$ is Church-Rosser [13].

Figure 3 presents the compiling translation from Λ to Λ_τ.

Observation 11. *Terms in the image of the compiling translation* $* : \Lambda \to \Lambda_\tau$ *are in* τ*-normal form.*

$$\begin{array}{lr} \text{terms} & M, N ::= V \mid x \mid M\,N \\ \text{values} & V ::= \lambda x\,.\,M \end{array}$$

$$(\beta) \qquad (\lambda x\,.\,M)\,N \longrightarrow M[x := N]$$

Fig. 1. The call-by-name language Λ

$$\begin{array}{lr} \text{terms} & M, N ::= V \mid M\,N \mid \texttt{force}\,M \\ \text{values} & V ::= x \mid \lambda x\,.\,M \mid \texttt{delay}\,M \end{array}$$

$$\begin{array}{lr} (\beta_{\mathrm{v}}) & (\lambda x\,.\,M)\,V \longrightarrow M[x := V] \\ (\tau) & \texttt{force}\,(\texttt{delay}\,M) \longrightarrow M \end{array}$$

Fig. 2. The call-by-value language with thunks Λ_τ

$$\begin{aligned} * &: \Lambda \to \Lambda_\tau \\ x^* &= \texttt{force}\,x \\ (\lambda x\,.\,M)^* &= \lambda x\,.\,M^* \\ (M\,N)^* &= M^*\,(\texttt{delay}\,N^*) \end{aligned}$$

Fig. 3. Compiling CBN to CBV using thunks

$$\text{terms} \quad P, Q ::= \texttt{force}\,x \mid \texttt{force}\,(\texttt{delay}\,P) \mid \lambda x\,.\,P \mid P\,(\texttt{delay}\,Q)$$

$$\begin{array}{lr} (\beta_{\mathrm{v}}) & (\lambda x\,.\,P)\,(\texttt{delay}\,Q) \longrightarrow P[x := \texttt{delay}\,Q] \\ (\tau) & \texttt{force}\,(\texttt{delay}\,P) \longrightarrow P \end{array}$$

Fig. 4. The compiled language of thunks $\Lambda_{\mathrm{thunks}}$

$$\begin{aligned} \sharp &: \Lambda_{\mathrm{thunks}} \to \Lambda \\ (\texttt{force}\,x)^\sharp &= x \\ (\texttt{force}\,(\texttt{delay}\,P))^\sharp &= P^\sharp \\ (\lambda x\,.\,P)^\sharp &= \lambda x\,.\,P^\sharp \\ (P\,(\texttt{delay}\,Q))^\sharp &= P^\sharp\,Q^\sharp \end{aligned}$$

Fig. 5. Decompiling from the thunk language

This follows from the fact that in the translation, `force` is only applied to identifiers.

Since the notion of reduction $\beta_{\mathrm{v}}\tau$ is Church-Rosser, it is sufficient to define the decompiling translation from Λ to Λ_τ on the set of terms in the image of the compiling translation closed under $\beta_{\mathrm{v}}\tau$ reduction. These terms can be described by the grammar in Figure 4. Figure 4 also presents the reductions of the call-by-value language Λ_τ specialized to $\Lambda_{\mathrm{thunks}}$. In analogy with the "administrative reductions" of CPS terms [21], one can view the τ-reduction of Figure 4 as an administrative reduction that manipulates suspensions (as opposed to the *source* reduction β_{v} of Figure 4 which corresponds to a reduction in the source language). Given this view, Observation 11 above implies that compiling generates terms that are in *administrative normal form*.

Figure 5 presents the decompiling translation. Decompiling simply removes the `delay` and `force` constructs.

Assessment: Is the translation a reflection? No. Having a reflection requires decompiling to be injective (Observation 6), and thunk decompiling is not injective. As in most cases in compiling, there are intermediate computational stages in the target language that are not uniquely expressible in the source language. For example, here are two target terms that decompile to the same source term:

$$(\texttt{force}\,(\texttt{delay}\,P))^\sharp \equiv P^\sharp.$$

This is to be expected since the source language does not make explicit the computation step where a suspension (*e.g.*, implemented as a closure) is forced (*e.g.*, by opening up the closure, restoring the saved environment, and evaluating).

From another point of view, having a reflection requires that compiling be surjective (Observation 7) on the language $\Lambda_{\mathrm{thunks}}$. Surjectivity fails since $\Lambda_{\mathrm{thunks}}$ terms with τ-redexes such as $\texttt{force}\,(\texttt{delay}\,P)$ are not in the image of the compiling translation (Observation 11 states that terms in the image of compiling are τ-normal).[1]

The reflection criteria is often violated when compiling fails to commute with substitution up to identity. In the thunk setting, commuting up to identity would mean that

$$M^*[x := \texttt{delay}\,N^*] \equiv M[x := N]^*.$$[2]

However, only the weaker property

$$M^*[x := \texttt{delay}\,N^*] \longrightarrow_\tau M[x := N]^*$$

holds, *i.e.*, $M^*[x := \texttt{delay}\,N^*]$ yields a term that lies outside the image of the compiling translation (which breaks the surjectivity requirement). Here is an example.

$$((\lambda x\,.\,x)\,N)^* \quad = \quad (\lambda x\,.\,\texttt{force}\,x)\,(\texttt{delay}\,N^*) \tag{3}$$

[1] However, such terms are reachable via $\beta_{\mathrm{v}}\tau$ from terms in the image of the translation.

[2] Note that the left-hand side above has the form of substitutions generated by β_{v}-reduction in the target language (see the β_{v}-reduction in Figure 4).

$$\longrightarrow_{\beta_{\mathrm{v}}} (\mathtt{force}\, x)[x := \mathtt{delay}\, N^*] \quad (4)$$
$$= \quad \mathtt{force}\, (\mathtt{delay}\, N^*) \quad (5)$$
$$\longrightarrow_{\tau} N^* \quad (6)$$

Line (5) shows that substitution produces a term $\mathtt{force}\,(\mathtt{delay}\, N^*)$ that is not in τ-normal form. Thus, (following Observation 11) this term is not in the image of the compiling translation. Therefore, compiling is not surjective, and thus the thunk translation cannot give rise to a reflection.

However, there is a tight relationship between the reduction theories of the source and target language — there is a reduction correspondence.

Theorem 12 (Thunking).
The translations $* : \Lambda \to \Lambda_{thunks}$ *and* $\sharp : \Lambda_{thunks} \to \Lambda$ *form a*

$$(\equiv, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)\text{-}correspondence.$$

This strengthens the result reported in [13] (an *equational correspondence*) to consider properties of reductions.

In summary, compiling with thunks is a very simple example where one cannot have a reflection because there is no source language kernel which is isomorphic to the target language (*e.g.*, $\Lambda_{\mathrm{thunks}}$). However, the reduction correspondence is a quite strong relationship: *(i)* components (3) and (4) guarantee that each reduction sequence in the source language is mirrored in the target language (and vice versa), *(ii)* components (1) and (2) state that compiling and decompiling act in a complementary manner: whether starting in the source language or target language, the translations never return a term that is poorer than the original. And in contrast to a reflection, the term may even be better.

4 Compiling CBN to CBV Using CPS

In this section, we consider compiling CBN to CBV using CBN CPS translations. Such translations have been used as the foundation for compiling CBN and lazy languages in several different settings [3,4,7,11,20].

4.1 Plotkin's CBN CPS Translation

We first consider Plotkin's CBN CPS translation [21] based on the presentation given in [13].

The source language is the call-by-name language of Figure 1. The target language is the call-by-value language of Figure 2, omitting the suspension constructs `delay` and `force` from the syntax, and τ-reduction from the calculus. We will refer to the modified calculus as Λ_{v}.

Figure 6 presents the compiling translation: Plotkin's call-by-name CPS transformation (where y and k are fresh variables). As with thunks, decompiling is defined on the language of terms in the image of the compiling translation closed under the reduction theory of the target language Λ_{v} (*i.e.*, β_{v}-reduction).

$$
\begin{array}{rlrl}
V^* &= \lambda k \,.\, k\, V^\dagger & (\lambda x \,.\, M)^\dagger &= \lambda x \,.\, M^* \\
x^* &= \lambda k \,.\, x\, k & & \\
(M\, N)^* &= \lambda k \,.\, M^*\, (\lambda y \,.\, y\, N^*\, k) & &
\end{array}
$$

Fig. 6. Compiling CBN to CBV using Plotkin's CPS translation

$$
\begin{array}{rl}
\text{root terms} & R, S ::= \lambda k \,.\, A \\
\text{computations} & C ::= x \mid R \mid (\lambda x \,.\, R)\, S \\
\text{answers} & A ::= K\, (\lambda x \,.\, R) \mid C\, K \\
\text{continuations} & K ::= k \mid \lambda y \,.\, y\, R\, K
\end{array}
$$

$$
\begin{array}{rr}
(\beta_{src}) & (\lambda x \,.\, R)\, S \longrightarrow R[x := S] \\
(\beta_{ans.1}) & (\lambda y \,.\, y\, R\, K)\, (\lambda x \,.\, S) \longrightarrow (\lambda x \,.\, S)\, R\, K \\
(\beta_{ans.2}) & (\lambda k \,.\, A)\, K \longrightarrow A[k := K]
\end{array}
$$

Fig. 7. The compiled language of Plotkin CBN CPS terms $\Lambda^P_{\text{cbn-cps}}$

$$
\begin{array}{rlrl}
(\lambda k \,.\, A)^\sharp &= A^\natural & (K\, (\lambda x \,.\, R))^\natural &= K^\Diamond[(\lambda x \,.\, R^\sharp)] \\
 & & (C\, K)^\natural &= K^\Diamond[C^\flat] \\
x^\flat &= x & & \\
R^\flat &= R^\sharp & k^\Diamond &= [\cdot] \\
((\lambda x \,.\, R)\, S)^\flat &= (\lambda x \,.\, R^\sharp)\, S^\sharp & (\lambda y \,.\, y\, R\, K)^\Diamond &= K^\Diamond[[\cdot]\, R^\sharp]
\end{array}
$$

Fig. 8. Decompiling from the Plotkin CBN CPS language

Figure 7 presents this language $\Lambda^P_{\text{cbn-cps}}$ and the reductions of Λ_{v} specialized to $\Lambda^P_{\text{cbn-cps}}$. For this language, we will assume three disjoint sets of identifiers.

$$
\begin{array}{l}
x \in \text{Computation-Identifiers} \\
y \in \text{Value-Identifiers} \\
k \in \text{Continuation-Identifiers} = \{k\}
\end{array}
$$

The first two are denumerably infinite, while the third need only contain exactly one identifier (a well-known property of CPS terms [5,6,22]).

Root terms expect a continuation and yield an answer. For every $M \in \Lambda$, $M^* = \lambda k \,.\, A$ for some answer A. *Answers* either throw a value to a continuation, or pass a continuation to a computation.

In the $\Lambda^P_{\text{cbn-cps}}$ reductions, β_{src}-reduction corresponds to β-reduction in the source calculus. The reductions $\beta_{ans.1}$ and $\beta_{ans.2}$ are administrative reductions — they simply manipulate continuations. Note that the syntactic categories of the language are closed under reduction.

Figure 8 presents the decompiling translation for $\Lambda^P_{\text{cbn-cps}}$. A key feature of this translation is that the translation of continuations $(\cdot)^\Diamond$ yields call-by-name evaluation contexts E_{n} which can be described by the following grammar.

$$E_{\text{n}} ::= [\cdot] \mid E_{\text{n}}\, N$$

Assessment: Hatcliff and Danvy's factoring of Plotkin's CBN CPS translation into the thunk translation and Plotkin's CBV CPS translation illustrates that the thunk translation and the CBN CPS translation are structurally similar [13]. This is further evidenced in the discussion below.

The Plotkin CBN CPS translation fails to yield a reflection for essentially the same reason as the thunking translation. To make the connection more apparent, note that a term of the form $\lambda k\,.\,A$ can be viewed as a "CPS-thunk" since it is a computation waiting for a continuation before evaluation can proceed. The reduction $\beta_{ans.2}$ of Figure 7 is analogous to τ-reduction; it can be viewed as forcing a CPS-thunk by supplying a continuation.

As with thunks, decompiling fails to be injective because the reduction of CPS-thunks is not directly expressible in the source language. The following example shows two terms that decompile to the same source term.

$$((\lambda k\,.\,k\,(\lambda x\,.\,R))\;K)^{\natural} \;\equiv\; (K\,(\lambda x\,.\,R))^{\natural}$$

In contrast to the thunks case, the CPS language $\Lambda^{P}_{\text{cbn-cps}}$ has the additional administrative reduction $\beta_{ans.1}$ which describes the naming of intermediate values. This reduction is also not reflected explicitly in the source language Λ.

As with thunks, the CBN CPS compiling translation also fails to be surjective because it does not commute with substitution up to identity. Instead, one has

$$M^{*}[x \;:=\; N^{*}] \;\longrightarrow\!\!\!\!\!\rightarrow_{\beta_{ans.2}}\; (M[x \;:=\; N])^{*}$$

as the following sequence illustrates:

$$x^{*}[x \;:=\; N^{*}] \;\;=\;\; (\lambda k\,.\,x\,k)[x \;:=\; N^{*}] \;\;=\;\; \lambda k\,.\,N^{*}\,k \;\;\longrightarrow_{\beta_{ans.2}}\;\; N^{*}$$

where the last step follows by a simple case analysis of N.

So what is exactly is the correspondence between Λ and $\Lambda^{P}_{\text{cbn-cps}}$? Following the thunk analogy, one might expect a $(\equiv, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$-correspondence (*i.e.*, a reduction correspondence). However, one has the slightly weaker property.

Theorem 13 (Plotkin CBN CPS). *The translations* $* : \Lambda \to \Lambda^{P}_{cbn\text{-}cps}$ *and* $\sharp : \Lambda^{P}_{cbn\text{-}cps} \to \Lambda$ *form a*

$$(\equiv, =, \twoheadrightarrow, \twoheadrightarrow)\text{-}\textit{correspondence.}$$

This strengthens the results of Plotkin [21] to include a notion of decompiling and an analysis of reduction properties (instead of only equational properties).

The following example illustrates why one has $=$ instead of $\twoheadrightarrow$ as the second component of the correspondence. Given the terms below which all correspond to the source term $x_1\,x_2$ (*i.e.*, $R_i{}^{\sharp} = x_1\,x_2$ for $i = 1, 2, 3$)

$$\begin{aligned}
R_1 &= \lambda k\,.\,(\lambda k\,.\,(\lambda k\,.\,x_1\,k)\,k)\,(\lambda y\,.\,y\,(\lambda k\,.\,x_2\,k)\,k)\\
R_2 &= \lambda k\,.\,(\lambda k\,.\,x_1\,k)\,(\lambda y\,.\,y\,(\lambda k\,.\,x_2\,k)\,k)\\
R_3 &= \lambda k\,.\,x_1\,(\lambda y\,.\,y\,(\lambda k\,.\,x_2\,k)\,k)
\end{aligned}$$

$$M^* = \lambda k\,.\,(M\ :\ k) \qquad x\ :\ K = x\,K$$
$$(\lambda x\,.\,M)\ :\ k = k\,(\lambda x\,.\,M^*)$$
$$(\lambda x\,.\,M)^\dagger = \lambda x\,.\,M^* \qquad (\lambda x\,.\,M)\ :\ (\lambda y\,.\,y\,P\,K) = (\lambda x\,.\,M^*)\,P\,K$$
$$V\,N\ :\ K = V^\dagger\,N^*\,K$$
$$M\,N\ :\ K = M\ :\ (\lambda y\,.\,y\,N^*\,K)$$

Fig. 9. Compiling CBN to CBV using an optimizing Plotkin-style CPS translation

one has $R_2 \equiv {R_1}^{\sharp *} \equiv {R_2}^{\sharp *} \equiv {R_3}^{\sharp *}$. Thus,

$$R_1 \longrightarrow_{\beta_{ans.2}} {R_1}^{\sharp *} \quad \text{and} \quad R_3 \longleftarrow_{\beta_{ans.2}} {R_3}^{\sharp *}.$$

Conceptually, the mismatch between Theorem 12 and Theorem 13 follows from the fact that thunk compiling produces terms in administrative normal form (see Observation 11), where as the Plotkin CBN CPS transformation does not. Specifically, $\beta_{ans.2}$ redexes exist in terms in the image of the translation.

Figure 9 presents an optimizing version of the translation in Figure 6. The translation uses a two argument function $(\cdot\ :\ \cdot)$ (inspired by Plotkin's colon translation [21]) that accumulates a continuation term in the second argument, and in essence performs adminstrative reductions "on the fly". A simple induction shows that terms in the image of the translation are in administrative normal form (*i.e.*, they do not contain $\beta_{ans.1}$ or $\beta_{ans.2}$ redexes).

Taking the optimizing translation as the definition of compiling and referring to the example above, we now have $R_3 \equiv {R_1}^{\sharp *} \equiv {R_2}^{\sharp *} \equiv {R_3}^{\sharp *}$. Thus,

$$R_1 \longrightarrow_{\beta_{ans.2}} {R_1}^{\sharp *} \quad \text{and} \quad R_2 \longrightarrow\!\!\!\!\!\rightarrow_{\beta_{ans.2}} {R_2}^{\sharp *} \quad \text{and} \quad R_3 \longrightarrow\!\!\!\!\!\rightarrow_{\beta_{ans.2}} {R_3}^{\sharp *}.$$

Even with the optimizing translation, one still does not have a reflection. Decompiling is still not injective (the definition of decompiling has not changed), and compiling is not surjective on $\Lambda^P_{\text{cbn-cps}}$ (it does not commute with substitution up to identity). However, we now have a reduction correspondence which matches the previous result for thunks.

Theorem 14 (Optimizing Plotkin CBN CPS).
The translations $* : \Lambda \to \Lambda^P_{cbn\text{-}cps}$ *of Figure 9 and* $\sharp : \Lambda^P_{cbn\text{-}cps} \to \Lambda$ *form a*

$$(\equiv, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)\text{-}\textit{correspondence}.$$

4.2 Fischer-style CBN CPS Translation

In Plotkin's CBN CPS translation, continuations are passed as the second argument to functions. This can been seen by the form of the continuation $(\lambda y\,.\,y\,R\,K)$ where R and K are the first and second arguments to a function that will bind to y. In the full version of this paper [2], we give an alternative to the Plotkin CBN CPS translation where continuations are passed as the first argument to function. We call this alternative a "Fischer-style" translation because the idea

of passing continuations as the first argument to functions originates with a CBV CPS translation given by Fischer [9]. Although one might imagine such a simple variation to be of no consequence, Sabry and Felleisen [22] have shown (in a call-by-value setting) that the Fischer-style allows more administrative reductions to be carried out at translation time by an optimizing translation. We carry this idea over to the call-by-name setting.

The Fischer-style CBN CPS translation fails to be a reflection for essentially the same reasons as the Plotkin CBN CPS translation: decompiling is not injective, and the translation does not commute with substitution up to identity. It gives rise to a $(\equiv, =, \twoheadrightarrow, =)$-correspondence.

Taking an optimizing translation to be the definition of compiling, one has a $(=, \twoheadrightarrow, \twoheadrightarrow, =)$-correspondence. We further show that that if one adds a single reduction to the source calculus, one actually has a $(\twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$-correspondence using the optimizing translation.

5 CBV CPS Translations

Fischer CBV CPS translation: Sabry and Felleisen gave an in-depth analysis of an optimizing Fischer CBV CPS transformation [22]. As noted earlier, they made the insightful observation that the Fischer-style translation allows more administrative reductions to be performed than the Plotkin CBV CPS transformation. A modified version of this transformation was subsequently used by Flanagan *et al.*[10] to show how the essence of compiling with continuations could be captured using an extended source calculus instead of actually introducing continuations.

Sabry and Wadler note that this translation cannot form a reflection (see [24] for a nice discussion). In terms of our previous discussions, it fails to be a reflection because it does not commute with substitution up to identity. Some reductions (and associated substitutions) produce terms (representing intermediate computational steps dealing with passing continuations) that lie outside the image of the translation (*i.e.*, surjectivity fails). Sabry and Wadler note that the proofs in the original presentation by Sabry and Felleisen [22] demontrate that the translation forms a $(\twoheadrightarrow, =, \twoheadrightarrow, \twoheadrightarrow)$ correspondence. They conclude their analysis of the translation by noting that "the Fischer translation [performs] too many administrative reductions to be a reflection" [24, p. 20]. They have recently shown that the translation gives rise to a Galois connection [25].

Plotkin CBV CPS translation: Sabry and Wadler show that an optimizing version of Plotkin's CBV CPS translation [21] forms a reflection if one takes Moggi's computational λ-calculus [19] as the source language. One may wonder how this CPS translation can form a reflection since all others that we have discussed cannot. The translation commutes with substitution up to identity, so this removes one obstacle. However β/β_{v} reduction still leads to terms which are not in the image of the translation. For example, consider the following compilation of a β_{v}-redex using Sabry and Wadler's optimizing translation.

$$((\lambda x\,.\,x)\,V)^* \quad = \quad \lambda k\,.\,(\lambda x\,.\,\lambda k\,.\,k\,x)\,V^\dagger\,k \tag{7}$$

$$\longrightarrow_{\beta_v} \lambda k\,.\,(\lambda k\,.\,k\,V^\dagger)\,k \tag{8}$$
$$\longrightarrow_{\beta_v} \lambda k\,.\,k\,V^\dagger \tag{9}$$

The term at line (8) necessarily lies outside the image of the optimizing translation because it contains an administrative redex. Therefore, one cannot have a reflection using the conventional β_v or β as the target calculi. Sabry and Wadler address this problem by using a non-standard calculus that glues together several β_v/β reductions. This follows their overall approach of modifying the associated calculi to obtain a reflection. Using this modified calculus one has the following reduction sequence.

$$((\lambda x\,.\,x)\,V)^* \;=\; \lambda k\,.\,(\lambda x\,.\,\lambda k\,.\,k\,x)\,V^\dagger\,k \tag{10}$$
$$\longrightarrow \lambda k\,.\,k\,V^\dagger \tag{11}$$

By combining the reductions into a single reduction, one moves past the offending term that lies outside the image of the translation. This fix cannot work with the other CPS translations because offending terms are created not only as the result of reduction, but as the result of *substitution* of terms which have leading λk's.

Even though one cannot have a reflection using the conventional β_v/β theory as the target calculus, in the full version of this paper we show that a reduction correspondence holds [2].

Based on the analysis of CPS translation throughout the paper, we question if *any* optimizing CPS translation can be a reflection with the conventional β_v/β calculus as the target calculus. Optimizing translations produce terms with no administrative redexes. When reducing CPS terms using β_v/β one must perform administrative reductions, that is, one cannot help but reach terms that lie outside the image of the translation. Since it seems that one must always create non-standard reduction theories to obtain reflections for CPS translations, the effectiveness of using reflection as a criteria for judging relationships between *existing standand calculi* when CPSing is not altogether clear.

6 Related Work

The most closely related results are those which use a compiling and decompiling translations to relate reduction or equational theories in source and target calculi. Of course, this includes the work of Sabry and Wadler which we have analyzed in detail [23,24]. This line of work can be traced back to Sabry and Felleisen's fundamental analysis of Fischer-style CBV CPS transformations, where they derive a equational theory that forms an equational correspondence with respect to Fischer-style CBV CPS terms under $\beta\eta$-equality. They showed that the derived theory forms an equational correspondence with Moggi's computational λ-calculus Λ_c.

Flanagan, Sabry, Duba, and Felleisen [10] showed that the properties of CPS terms that make them desirable for compiling could be captured using the equational theory derived by Sabry and Felleisen. That is, one could have the benefits of CPS with out resorting to the more complicated structure of CPS terms.

Hatcliff and Danvy incorporated many of these ideas in a general framework for reasoning about Plotkin-style CPS translations based on Moggi's computational meta-language [19]. They illustrated how CPS translations and associated meta-theory for various evaluation strategies such as call-by-name, call-by-value, and other mixed strategies (e.g., as might be applied to call-by-name strictness analyzed terms) could be derived using this framework. An equational correspondence between Moggi's meta-language and CPS terms plays a prominent rôle in establishing the general theory. In a work similar in spirit to Flanagan *et al.*, they demonstrated how Moggi's meta-language provides a foundation for partial evaluation of functional programs with computational effects [14]. Lawall and Thiemann have recently presented an alternate approach to the same goal by using the notion of reflection to justify the correctness of a partial evaluation for Moggi's computational λ-calculus Λ_c [18]. In an earlier work, Hatcliff and Danvy gave a detailed analysis of the relationship between thunks and Plotkin's CBN and CBV CPS translations [13]. In essence, they showed an equational correspondence between Λ and Λ_{thunks}, and demonstrated that Plotkin's CBN CPS translation could be factored through the thunk translation of Figure 3 and Plotkin's CBV CPS translation.

It was Danvy who first emphasized the importance of a *direct-style* (DS) or decompiling translation for CBV CPS terms [5]. Although he did not consider equational or reduction theories, he showed how CPS terms in a Scheme-like language could be mapped back to direct-style. Lawall and Danvy [17] later showed how the CPS and DS translations could be staged using an intermediate language similar to Moggi's computational λ-calculus. Although they did not consider equational or reduction theories, they paid special attention to the problem of preserving evaluation sequencing order in realistic languages like Scheme. It was this work that first introduced the notion of Galois connection to relate the compiling and decompiling CPS translations. Their notion of Galois connection is quite a bit different from Sabry and Wadler. The ordering used is not based on reduction (as is Sabry and Wadler's), but is induced directly from the translations and captures structural similarities between terms. The use of such intensional properties as the basis of the order seems to have some weaknesses. For example, Lawall and Danvy note that the order is not preserved by their translations that introduce and eliminate continuations.

7 Conclusion

The table below summarizes the relationships between the conventional source and target calculi induced by the translations we have discussed.

	conventional	*optimizing*
Thunks	$(\equiv, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$	
Plotkin CBN CPS	$(\equiv, =, \twoheadrightarrow, \twoheadrightarrow)$	$(\equiv, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$
Fischer CBN CPS	$(\equiv, =, \twoheadrightarrow, =)$	$(=, \twoheadrightarrow, \twoheadrightarrow, =)$
Plotkin CBV CPS	$(=, =, =, \twoheadrightarrow)$	$(\twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow, \twoheadrightarrow)$
Fischer CBV CPS	$(=, =, =, =)$	$(=, \twoheadrightarrow, \twoheadrightarrow, =)$

This work should be viewed as complementary to Sabry and Wadler's. The goal of both works is to establish technical tools and criteria for reasoning about and assessing how various compiling-oriented translations interact with reduction properties of program calculi. A difference is that Sabry and Wadler strive for a very tight relationship between source and target calculi reductions even if it means substantially modifying the source and target calculi. Our approach is to give a framework for describing the relationships that arise naturally between conventional calculi.

A benefit of Sabry and Wadler's approach is that the notions of Galois connection and reflection naturally lead to notions such as core source and target calculi that are isomorphic. Besides giving details of the properties of the translations from the table above, the full version of this paper illustrates how a similar notion of isomorphism can be captured in our framework by quotienting the conventional calculi [2].

A study of reduction theories such as ours and Sabry and Wadler's is not only of interest in compiling functional programs. For example, preserving reductions under CPS translation is important in techniques for inferring strong normalization from weak normalization [26]. There also appears to be strong connections to work on compilation of term-rewriting systems [16]. Both of these applications seem to involve translations that cannot form reflections.

Acknowledgements Thanks to Olivier Danvy, Sergey Kotov, Julia Lawall, Amr Sabry, Peter Thiemann, and Phil Wadler for helpful discussions.

References

1. Andrew W. Appel. *Compiling with Continuations.* Cambridge University Press, 1992.
2. Gilles Barthe, John Hatcliff, and Morten Heine Sørensen. Reflections on Reflections (full version). Forthcoming DIKU Technical Report. Available at `http://www.cs.okstate.edu/~hatclif`.
3. P. N. Benton. *Strictness Analysis of Lazy Functional Programs.* PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, England, 1995.
4. Geoffrey Burn and Daniel Le Métayer. Proving the correctness of compiler optimisations based on a global program analysis. Technical report Doc 92/20, Department of Computing, Imperial College of Science, Technology and Medicine, London, England, 1992.
5. Olivier Danvy. Back to direct style. *Science of Computer Programming*, 22(3):183–195, 1994. Special Issue on ESOP'92, the Fourth European Symposium on Programming, Rennes, February 1992.
6. Olivier Danvy and Andrzej Filinski. Representing control, a study of the CPS transformation. Mathematical Structures in Computer Science, Vol. 2, No. 4, pages 361–391. Cambridge University Press, December 1992.
7. Olivier Danvy and John Hatcliff. CPS transformation after strictness analysis. *ACM Letters on Programming Languages and Systems*, 1(3):195–212, 1993.
8. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
9. Michael J. Fischer. Lambda-calculus schemata. In Talcott [28]. An earlier version appeared in the *Proceedings of the ACM Conference on Proving Assertions about Programs*, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

10. Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In David W. Wall, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Languages Design and Implementation*, SIGPLAN Notices, Vol. 28, No 6, pages 237–247, Albuquerque, New Mexico, June 1993. ACM Press.
11. Pascal Fradet and Daniel Le Métayer. Compilation of functional languages by program transformation. *ACM Transactions on Programming Languages and Systems*, 13:21–51, 1991.
12. John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans Boehm, editor, *Proceedings of the Twenty-first Annual ACM Symposium on Principles of Programming Languages*, Portland, Oregon, January 1994. ACM Press.
13. John Hatcliff and Olivier Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 1995. (in press). The extended version of this paper appears as DIKU-Report 95/3.
14. John Hatcliff and Olivier Danvy. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science*, 1996. (in press). To appear in a special issue devoted to selected papers from the *Workshop on Logic, Domains, and Programming Languages.* Darmstadt, Germany. May, 1995.
15. P. Z. Ingerman. Thunks, a way of compiling procedure statements with some comments on procedure declarations. *Communications of the ACM*, 4(1):55–58, 1961.
16. J. Kamperman and H. Walters. Minimal term rewriting systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification*, volume 1130 of *Lecture Notes in Computer Science*, pages 274–290, 1996.
17. Julia L. Lawall and Olivier Danvy. Separating stages in the continuation-passing style transformation. In Susan L. Graham, editor, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 124–136, Charleston, South Carolina, January 1993. ACM Press.
18. Julia L. Lawall and Peter Thiemann. Sound specialization in the presence of computational effects. To appear in the *Proceedings of TACS'97.*
19. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
20. Chris Okasaki, Peter Lee, and David Tarditi. Call-by-need and continuation-passing style. In Talcott [28].
21. Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.
22. Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In Talcott [28], pages 289–360.
23. Amr Sabry and Philip Wadler. A reflection on call-by-value. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming*, April 1996.
24. Amr Sabry and Philip Wadler. A reflection on call-by-value. Technical Report CIS-TR-96-08, Department of Computing and Information Sciences, University of Oregon, Eugene, Oregon, April 1996.
25. Amr Sabry and Philip Wadler. Personal communication. June 4, 1997.
26. Morten Heine Sørensen. Strong normalization from weak normalization in typed λ-calculi. *Information and Computation*, 133(1):35–71, 1997.
27. 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.
28. Carolyn L. Talcott, editor. *Special issue on continuations*, LISP and Symbolic Computation, Vol. 6, Nos. 3/4. Kluwer Academic Publishers, 1993.