J. Functional Programming **1** (1): 1–000, January 1993

Operational Interpretations of an Extension of F_ω with Control Operators†

Robert Harper‡ and Mark Lillibridge§
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We study the operational semantics of an extension of Girard's System F_ω with two control operators: an *abort* operation that abandons the current control context, and a *callcc* operation that captures the current control context. Two classes of operational semantics are considered, each with a call-by-value and a call-by-name variant, differing in their treatment of polymorphic abstraction and instantiation. Under the *standard* semantics polymorphic abstractions are values and polymorphic instantiation is a significant computation step; under the *ML-like* semantics evaluation proceeds beneath polymorphic abstractions and polymorphic instantiation is computationally insignificant.

Compositional, type-preserving continuation-passing style (cps) transformation algorithms are given for the standard semantics, resulting in terms on which all four evaluation strategies coincide. This has as a corollary the soundness and termination of well-typed programs under the standard evaluation strategies. In contrast, such results are obtained for the call-by- value ML-like strategy only for a restricted sub-language in which constructor abstractions are limited to values. The ML-like call-by-name semantics is indistinguishable from the standard call-by-name semantics when attention is limited to complete programs.

Capsule Review

(To be provided by the editor)

Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah. Bah Blah Blaah.

† This is a revised and expanded version of "Explicit Polymorphism and CPS Conversion" presented at the Twenthieth Symposium on Principles of Programming Languages, Charleston, SC, January, 1993. (Harper and Lillibridge, 1993a).

‡ This work was sponsored by the Defense Advanced Research Projects Agency, CSTO, under the title "The Fox Project: Advanced Development of Systems Software", ARPA Order No. 8313, issued by ESD/AVS under Contract No. F19628–91–C–0168. Electronic mail address: `rwh@cs.cmu.edu`.

§ Supported by a National Science Foundation Graduate Fellowship. Electronic mail address: `mdl@cs.cmu.edu`.

1 Introduction

The use of type theory as a central organizing principle has led to significant advances in the design and implementation of programming languages. The influence of type theory is well exemplified by the various dialects of ML (Gordon *et al.*, 1979; Projet Formel, 1987; Leroy and Mauny, 1992; Harper and Mitchell, 1993; MacQueen, 1986; Milner *et al.*, 1990; Milner and Tofte, 1991), Hope (Burstall *et al.*, 1980), and Quest (Cardelli, 1989). These languages may be viewed as enrichments of the Girard-Reynolds polymorphic typed λ-calculus (Girard, 1972; Reynolds, 1974) with more expressive typing constructs (such as subsumption (Cardelli *et al.*, 1991) and intersection types (Pierce, 1993)) and with primitive operations for expressing control and store effects (Felleisen and Hieb, 1992; Harper *et al.*, 1993; Tofte, 1990). Taken together these extensions provide a highly expressive programming notation that captures a wide range of programming techniques.

Advances in language design are often accompanied by corresponding advances in compiler technology. Of particular relevance to this paper is the *continuation-passing style (cps) translation* introduced by Reynolds (1972) and Fischer (Fischer, 1993). The main idea of the cps translation is to make the control context of the evaluator available as a run-time value, thereby making the order of evaluation explicit and allowing for the extension of the language with non-local transfers of control. The translation has proved to be an important tool for compiler writers, as emphasized by Steele (1978), Kranz, *et al.* (1986), and Appel (1992), among others. (See Reynolds's survey (Reynolds, 1993) for a thorough account of the history of the use of continuations in programming language semantics and implementation.)

In this paper we study the properties of the cps translation for the extension of Girard's F_ω (Girard, 1972; Girard *et al.*, 1989) with two control operators, one which discards the current evaluation context and one which captures the current evaluation context (analogous to the `call/cc` primitive of Scheme (Clinger and Rees, 1991)). These constructs provide a basis for defining higher-level patterns of control such as co-routines (Haynes *et al.*, 1986) and threads (Cooper and Morrisett, 1990; Reppy, 1991). Several operational semantics for this extension of F_ω are considered. These may be divided into two broad categories, the *standard* semantics and the *ML-like* semantics, each of which admits a call-by-value (cbv) and call-by-name (cbn) variant. These four interpretations cover the main semantics for polymorphic functional languages that have been considered in the literature, including those for ML (Milner *et al.*, 1990), Haskell (Hudak and Wadler, 1990), and Quest (Cardelli, 1989). The "standard" semantics differ from their "ML-like" counterparts in the treatment of polymorphic abstractions. Under the standard interpretation polymorphic abstractions are values, and polymorphic instantiation is a non-trivial computation step. The ML-like semantics, on the other hand, evaluate beneath polymorphic abstractions and regard polymorphic instantiation as essentially trivial, mimicking the behavior of the untyped operational semantics of ML programs.

We study the typing and semantic properties of the cps translation for each of the operational interpretations of F_ω enriched with control operators. The main goal is to extend the results of Plotkin (Plotkin, 1975) (for the untyped case) and Meyer and Wand (Meyer and Wand, 1985) (for the simply typed case) to this extension of F_ω. To capture the "indifference" of the cps form to the cbv/cbn distinction we begin by isolating three *cps sub-languages* of pure F_ω. The standard cps sub-language consists of a set of terms of F_ω on which the cbv and cbn variants of the standard semantics coincide and which is closed under evaluation by both of those variants. Similarly, the ML-like cps sub-language consists of a set of terms on which the cbv and cbn variants of the ML-like semantics coincide and which is closed under evaluation by the ML-like variants. Finally, we isolate a "strict" cps sub-language on which both variants of both semantics coincide and which is closed under all four variants. We then consider the cps translation from F_ω enriched with control operators into a suitable cps sub-language corresponding to each variant of each operational semantics. The typing properties of the cps translations are established and used to derive termination and soundness results for F_ω with control operators. The correctness of the translations is established by extending the methods of Plotkin (Plotkin, 1975) and Griffin (Griffin, 1990).

This paper is an extension of an earlier study conducted by the authors (Harper and Lillibridge, 1993b) for the special case of ML under an untyped operational semantics. In particular the fundamental non-existence result for cps translations established there is extended here to the case of the cbv ML-like interpretation of F_ω enriched with control operators. On the other hand we establish the fundamental properties of the standard semantics, and show that no surprises such as those encountered for the ML-like interpretations arise. In view of these results it would appear that a standard, rather than an ML-like, interpretation of enrichments of F_ω is most appropriate.

2 The Language $\mathsf{F}_\omega^{\mathcal{C}}$

The syntax of $\mathsf{F}_\omega^{\mathcal{C}}$ is defined as follows:

Kinds	K	$::=$	$\Omega \mid K_1 \Rightarrow K_2$
Constructors	A, B	$::=$	$u \mid Ans \mid A_1 \rightarrow A_2 \mid \forall u{:}K.A \mid \lambda u{:}K.A \mid A_1\, A_2$
Terms	M	$::=$	$x \mid \lambda x{:}A.M \mid M_1\, M_2 \mid \Lambda u{:}K.M \mid M\{A\} \mid$
			$\mathcal{C}_A(M) \mid \mathcal{X}_A(M)$
Kind Assignments	Δ	$::=$	$\emptyset \mid \Delta, u{:}K$
Type Assignments	Γ	$::=$	$\emptyset \mid \Gamma, x{:}A$

The meta-variable u ranges over *constructor variables* and the meta-variable x ranges over *term variables*. The constructor Ans is an unspecified base type, representing the type of "answers", the final results of evaluation for complete programs. The primitives $\mathcal{X}$ and $\mathcal{C}$ are the control operators *abort* and *call-with-current-continuation (callcc)*.

The type system of $\mathsf{F}_\omega^{\mathcal{C}}$ consists of a set of rules for deriving judgements of the following forms:

$\rhd$	Δ	*well-formed kind assignment*
$\Delta \rhd$	Γ	*well-formed type assignment*
$\Delta \rhd$	$A : K$	*well-formed constructor*
$\Delta \rhd$	$A_1 = A_2 : K$	*equal constructors*
$\Delta; \Gamma \rhd$	$M : A$	*well-formed term*

The rules for deriving these judgements are largely standard; see the Appendix for a complete definition. The treatment of control operators is novel and merits further discussion. The typing rules governing the control primitives are as follows:

$$\frac{\Delta \rhd A : \Omega \qquad \Delta; \Gamma \rhd M : Ans}{\Delta; \Gamma \rhd \mathcal{X}_A(M) : A} \qquad (\text{T-ABORT})$$

$$\frac{\Delta; \Gamma \rhd M : (\forall u{:}\Omega. A \to u) \to A \qquad u \notin dom(\Delta)}{\Delta; \Gamma \rhd \mathcal{C}_A(M) : A} \qquad (\text{T-CALLCC})$$

Informally, $\mathcal{X}$ abandons the current evaluation context, and yields as final result the value of the given expression. Since the final answer computed by a program is to have type Ans, we require that the argument to abort have type Ans. As to $\mathcal{C}$ the informal interpretation is that the "current continuation" is passed to the argument of $\mathcal{C}$. The argument must therefore be of functional type, with domain the type of continuations that accept values of type A, where A is the type of the whole expression. For the sake of simplicity we take this type of A-accepting continuations to be $\forall u{:}\Omega. A \to u$, reflecting the fact that a continuation, once invoked, does not return to its call site. (See Harper, *et al.* (1993) for further discussion of this and related points. See also Griffin (1990) for a similar type system for control operations. Relative to Griffin's language our type Ans plays the role of logical falsehood, $\bot$, and $\forall u{:}\Omega. A \to u$ plays the role of classical negation, $\neg A$.)

The following technical lemma summarizes some useful properties of the type system:

Lemma 2.1

1. If $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd \Gamma$ then $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \rhd \Delta$
2. If $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd A : K$ then $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \rhd \Delta$
3. If $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd A_1 = A_2 : K$ then $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd A_1 : K$ and $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd A_2 : K$
4. If $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta; \Gamma \rhd M : A$ then $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd \Gamma$ and $\mathsf{F}^{\mathcal{C}}_{\omega} \vdash \Delta \rhd A : \Omega$

A *context* C is an expression of $\mathsf{F}^{\mathcal{C}}_{\omega}$ with a single *hole*, written $[]$:

$$\textit{Contexts} \quad C \quad ::= \quad [] \mid \lambda x{:}A.\, C \mid C\, M \mid M\, C \mid \Lambda u{:}K.\, C \mid C\{A\} \mid \mathcal{C}_A(C) \mid \mathcal{X}_A(C)$$

The hole in C may be *filled* by an expression M, written $C[M]$, by replacing the hole with M, incurring capture of free variables in M that are bound at the occurrence of the hole. For example, if $C = \Lambda u{:}\Omega.\lambda x{:}u.[]$, and $M = f\{u\}\, x$, then $C[M] = \Lambda u{:}\Omega.\lambda x{:}t.f\{u\}\, x$. The variables that are bound within the hole of a context are said to be *exposed (to capture)* by that context. We write $ETV(C)$ for the exposed constructor variables and $EV(C)$ for the exposed ordinary variables of a context.

Type checking in $\mathsf{F}_\omega^{\mathcal{C}}$ is compositional in the sense that if an expression is well-typed, then so are all constituent expressions.

Lemma 2.2 (Decomposition)
Suppose that $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \triangleright C[M] : A$ such that $EV(C) \cap dom(\Gamma) = \emptyset$ and $ETV(C) \cap dom(\Delta) = \emptyset$.† Then there exists Δ', Γ', and B such that:

- $\mathrm{dom}(\Delta') = ETV(C)$;
- $\mathrm{dom}(\Gamma') = EV(C)$;
- $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta, \Delta'; \Gamma, \Gamma' \triangleright M : B$

Proof
Routine induction on the structure of contexts. □

Furthermore, only the type of a constituent of a well-formed expression is relevant to typing. Consequently any constituent may be replaced by a term of the same type without effecting typability of the whole expression.

Lemma 2.3 (Replacement)
Suppose that $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \triangleright C[M] : A$, with $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta, \Delta'; \Gamma, \Gamma' \triangleright M : B$ in accordance with the decomposition lemma. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta, \Delta'', \Delta'; \Gamma, \Gamma'', \Gamma' \triangleright M' : B$ then $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta, \Delta''; \Gamma, \Gamma'' \triangleright C[M'] : A$.

Proof
Routine induction on typing derivations. □

3 Operational Semantics for $\mathsf{F}_\omega^{\mathcal{C}}$

We consider two main operational semantics for $\mathsf{F}_\omega^{\mathcal{C}}$ that differ in the treatment of polymorphic abstraction and application.

3.1 Notation

Following Plotkin (1975) and Felleisen (1992), we specify an operational semantics by defining the set of *values*, the set of *evaluation contexts*, and the *one step evaluation relation* for that semantics. One-step evaluation is a binary relation on programs that is defined by a set of rules of the form $E[R] \hookrightarrow M$, where E is an evaluation context, R is an expression, the *redex*, and M is determined as a function of E and R. In this case $E[R]$ is said to *evaluate to M in one step*. We define $\hookrightarrow^{0,1}$ to be the reflexive closure and $\hookrightarrow^{*}$ to be the reflexive, transitive closure of the $\hookrightarrow$ relation.

An $\mathsf{F}_\omega^{\mathcal{C}}$ *program* is a closed term of type Ans. We will arrange things so that a program P is either a value or can be represented in exactly one way as $E[R]$ where E is an evaluation context and R is a redex.

† The conditions on the exposed variables can always be satisfied by alpha-renaming $C[M]$ appropriately.

3.2 Standard Call-by-Value Semantics

The *standard call-by-value (std-cbv)* semantics is defined as follows:

$$\begin{array}{rcl} V & ::= & x \mid \lambda x{:}A.M \mid \Lambda u{:}K.M \\ E & ::= & [] \mid E\,M \mid V\,E \mid E\{A\} \end{array}$$

$$\begin{array}{rcll} E[(\lambda x{:}A.M)\,V] & \hookrightarrow_{std\text{-}cbv} & E[[V/x]M] & \\ E[(\Lambda u{:}K.M)\{A\}] & \hookrightarrow_{std\text{-}cbv} & E[[A/u]M] & \\ E[\mathcal{X}_A(M)] & \hookrightarrow_{std\text{-}cbv} & M & \\ E[\mathcal{C}_A(M)] & \hookrightarrow_{std\text{-}cbv} & E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))] & (u \notin FTV(A)) \end{array}$$

The first two evaluation rules specify β-(reduction) steps. When it is necessary to distinguish between them, we will use β_λ to refer to β-reductions of λ-applications and β_Λ to refer to β-reductions of Λ-applications. The third and fourth rules define the evaluation of the control operators. Note that the evaluation context E is "reified" as the polymorphic function $\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x])$. When applied to a target type B and an argument u of type A, this function aborts the evaluation context at the point of application and continues evaluation with the expression $E[u]$. Since the evaluation context of invocation is abandoned, the result type, B, is arbitrary.

Lemma 3.1 (Canonical Forms)

1. If V is a closed value of functional type, then $V = \lambda x{:}A.M$ for some type A and term M.
2. If V is a closed value of quantified type, then $V = \Lambda u{:}K.M$ for some kind K and term M.

Theorem 3.2 (Progress)
If M is a closed, well-typed expression of type A, then either M is a value, or else there exist a unique evaluation context E and unique redex R such that $M = E[R]$.

Proof
The proof proceeds by induction on the structure of typing derivations, using Lemma 2.2 and Lemma 3.1. □

Theorem 3.3 (Preservation)
If P is a program and $P \hookrightarrow_{std\text{-}cbv} Q$, then Q is a program.

Proof
If $P \hookrightarrow_{std\text{-}cbv} Q$, then $P = E[R]$ for some evaluation context E and redex R. By inspecting the definition of std-cbv evaluation contexts, we see that $EV(E) = \emptyset$ and $ETV(E) = \emptyset$. Hence, by Lemma 2.2 there exists a closed type B such that $\mathsf{F}_\omega^{\mathcal{C}} \vdash \emptyset;\emptyset \rhd R : B$. We proceed by cases on the form of R. If $R = \mathcal{X}_A(M)$, then $Q = M$ and $\mathsf{F}_\omega^{\mathcal{C}} \vdash \emptyset;\emptyset \rhd M : Ans$, as required. The remaining cases are handled similarly. □

The following corollary is analogous to Milner's type soundness theorem for ML (Milner, 1978):

Corollary 3.4 (Type Soundness)

If P is a program, then either P is a value, or there is a program Q such that $P \hookrightarrow_{\mathit{std\text{-}cbv}} Q$.

Theorem 3.5 (Termination for F_ω)
For every F_ω program P there exists a pure value V such that $P \hookrightarrow^*_{\mathit{std\text{-}cbv}} V$.

Proof
When restricted to F_ω programs, the std-cbv evaluation relation is a particular β-reduction strategy, and hence by the strong normalization property of F_ω (Girard, 1972) is terminating. The result must be a value by Lemma 3.1. $\Box$

Termination of $\mathsf{F}^{\mathcal{C}}_\omega$ programs under the std-cbv semantics will be established in Section 5. The following property of std-cbv evaluation will be important to that argument:

Lemma 3.6
Any infinite std-cbv evaluation sequence contains infinitely many β-reduction steps.

Proof
It is sufficient to show that it is impossible to construct an infinite evaluation sequence consisting solely of $\mathcal{X}$ and $\mathcal{C}$ steps. This can be done by showing that if $E[\mathcal{O}_A(M)] \hookrightarrow_{\mathit{std\text{-}cbv}} E'[\mathcal{O}'_{A'}(M')]$ where $\mathcal{O}$ and $\mathcal{O}' \in \{\mathcal{X},\ \mathcal{C}\}$ then M' is a proper subterm of M. $\Box$

3.3 Standard Call-by-Name Semantics

The *standard call-by-name (std-cbn)* semantics is defined as follows:

$$\begin{array}{lcl} V & ::= & \lambda x{:}A.M \mid \Lambda u{:}K.M \\ E & ::= & [] \mid E\,M \mid E\{A\} \end{array}$$

Note that variables are not values under the call-by-name interpretation.

$$\begin{array}{rll} E[(\lambda x{:}A.M_1)\,M_2] & \hookrightarrow_{\mathit{std\text{-}cbn}} & E[[M_2/x]M_1] \\ E[(\Lambda u{:}K.M)\{A\}] & \hookrightarrow_{\mathit{std\text{-}cbn}} & E[[A/u]M] \\ E[\mathcal{X}_A(M)] & \hookrightarrow_{\mathit{std\text{-}cbn}} & M \\ E[\mathcal{C}_A(M)] & \hookrightarrow_{\mathit{std\text{-}cbn}} & E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))] \qquad (u \notin FTV(A)) \end{array}$$

The canonical forms lemma and the progress and preservation theorems given above for the standard call-by-value semantics carry over to the standard call-by-name case.

Theorem 3.7 (Type Soundness)
If P is a program, then either P is a value, or there is a program Q such that $P \hookrightarrow_{\mathit{std\text{-}cbn}} Q$.

Theorem 3.8 (Termination for F_ω)
If P is an F_ω program, then there exists a value V such that $P \hookrightarrow^*_{\mathit{std\text{-}cbn}} V$.

Just as for the std-cbv case, an infinite std-cbn evaluation sequence must contain infinitely many β steps.

3.4 ML-like Call-by-Value Semantics

The *ML-like call-by-value (ml-cbv)* semantics is defined as follows:

$$\begin{array}{lll} V & ::= & x \mid \lambda x{:}A.M \mid \Lambda u{:}K.V \\ E & ::= & [] \mid E\,M \mid V\,E \mid \Lambda u{:}K.E \mid E\{A\} \end{array}$$

$$\begin{array}{rll} E[(\lambda x{:}A.M)\,V] & \hookrightarrow_{ml\text{-}cbv} & E[[V/x]M] \\ E[(\Lambda u{:}K.V)\{A\}] & \hookrightarrow_{ml\text{-}cbv} & E[[A/u]V] \\ E[\mathcal{X}_A(M)] & \hookrightarrow_{ml\text{-}cbv} & M \\ E[\mathcal{C}_A(M)] & \hookrightarrow_{ml\text{-}cbv} & E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))] \qquad (u \notin FTV(A)) \end{array}$$

Note that the hole in an evaluation context can occur within the scope of a constructor abstraction. Consequently, a constructor abstraction is a value only if its body is a value.

Lemma 3.9 (Canonical Forms)

1. If V is a closed value of functional type, then $V = \lambda x{:}A.M$ for some type A and term M.
2. If V is a closed value of quantified type, then $V = \Lambda u{:}K.V'$ for some kind K and value V'.

Theorem 3.10 (Progress)
If M is a well-typed, closed term of type A, then either M is a value, or there exist a unique evaluation context E and a unique redex R such that $M = E[R]$.

Proof
By induction on typing derivations, using Lemma 2.2 and Lemma 3.9. □

Theorem 3.11 (Preservation for F_ω)
If P is an F_ω program and $P \hookrightarrow_{ml\text{-}cbv} Q$, then Q is an F_ω program.

Proof
Similar to the proof for the std-cbv case. □

Theorem 3.12 (Termination for F_ω)
If P is an F_ω program, then there exists a value V such that $P \hookrightarrow^*_{ml\text{-}cbv} V$.

The preservation theorem cannot be extended to full $\mathsf{F}^{\mathcal{C}}_\omega$. It is instructive to see where the obvious proof attempt breaks down. Let $P = E'[\Lambda t{:}\Omega.\,\mathcal{C}_A(M)]$, and let $E = E'[\Lambda t{:}\Omega.[]]$. Then $P = E[\mathcal{C}_A(M)]$ is reducible, and we must show that the reduct, $E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))]$, is a program. By the decomposition lemma it suffices to show that $\mathsf{F}^{\mathcal{C}}_\omega \vdash \Delta', t{:}\Omega; \emptyset \rhd M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x])) : A$, where Δ' is derived from E'. For this it suffices to show that $\mathsf{F}^{\mathcal{C}}_\omega \vdash \Delta', t{:}\Omega, u{:}\Omega; x{:}A \rhd E[x] : Ans$. But notice that $E[x] = E'[\Lambda t{:}\Omega.x]$, which may not be well-formed: the type A ascribed to x may involve an occurrence of t that is captured by the inner Λ-abstraction. This may be turned into a counter-example to the preservation theorem by a simple adaptation of the argument given by the authors elsewhere (Harper and Lillibridge, 1993b).

If constructor abstractions are restricted so that $\Lambda u{:}K.M$ is well formed only if M is a value, the counterexample to preservation is avoided, and preservation can be proved. Let us call this restricted language $\mathsf{F}^{\mathcal{C}-}_\omega$.

Theorem 3.13 (Preservation for $\mathsf{F}_\omega^{\mathcal{C}-}$)
If P is a $\mathsf{F}_\omega^{\mathcal{C}-}$ program, and $P \hookrightarrow_{ml\text{-}cbv} Q$, then Q is a $\mathsf{F}_\omega^{\mathcal{C}-}$ program.

Proof
The ml-cbv and std-cbv semantics coincide on $\mathsf{F}_\omega^{\mathcal{C}-}$ terms. □

3.5 ML-like Call-by-Name Semantics

The *ML-like call-by-name (ml-cbn)* semantics is defined as follows:

$$\begin{array}{lcl} V & ::= & \lambda x{:}A.M \mid \Lambda u{:}K.V \\ E & ::= & [] \mid E\,M \mid \Lambda u{:}K.E \mid E\{A\} \end{array}$$

$$\begin{array}{rll} E[(\lambda x{:}A.M_1)\,M_2] & \hookrightarrow_{ml\text{-}cbn} & E[[M_2/x]M_1] \\ E[(\Lambda u{:}K.V)\{A\}] & \hookrightarrow_{ml\text{-}cbn} & E[[A/u]V] \\ E[\mathcal{X}_A(M)] & \hookrightarrow_{ml\text{-}cbn} & M \\ E[\mathcal{C}_A(M)] & \hookrightarrow_{ml\text{-}cbn} & E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))] \qquad (u \notin FTV(A)) \end{array}$$

Theorem 3.14 (Type Soundness for F_ω)
If P is a program, then either P is a value, or there is a program Q such that $P \hookrightarrow_{ml\text{-}cbn} Q$.

The extension of preservation to $\mathsf{F}_\omega^{\mathcal{C}}$ runs afoul of difficulties similar to those encountered in the ml-cbv case, but is nevertheless sound. The key observation is that in a call-by-name setting a polymorphic abstraction is evaluated only when it is applied to a type constructor argument. By insisting that the polymorphic instantiation occur prior to evaluation under the polymorphic abstraction, we avoid the problems with capture that arise in the ml-cbv case. Such a semantics is defined as follows:

$$\begin{array}{lcl} V & ::= & \lambda x{:}A.M \mid \Lambda u{:}K.V \\ E & ::= & []\{A_1\}\ldots\{A_n\} \mid (E\,M)\{A_1\}\ldots\{A_n\} \mid \Lambda u{:}K.E \end{array}$$

$$\begin{array}{rll} E[(\lambda x{:}A.M_1)\,M_2] & \hookrightarrow_{ml\text{-}cbn'} & E[[M_2/x]M_1] \\ E[(\Lambda u{:}K.M)\{A\}] & \hookrightarrow_{ml\text{-}cbn'} & E[[A/u]M] \\ E[\mathcal{X}_A(M)] & \hookrightarrow_{ml\text{-}cbn'} & M \\ E[\mathcal{C}_A(M)] & \hookrightarrow_{ml\text{-}cbn'} & E[M\,(\Lambda u{:}\Omega.\lambda x{:}A.\,\mathcal{X}_u(E[x]))]\ (u \notin FTV(A)) \end{array}$$

It is easy to see that this semantics and the standard call-by-name semantics coincide on complete programs (closed terms of basic type). Consequently we shall not give further consideration to the ml-cbn semantics.

3.6 Relation of ML-like Semantics to ML

The dynamic semantics of ML is ordinarily defined on untyped terms. To relate our ML-like semantics to the usual untyped semantics, we introduce a system of untyped terms along with a call-by-name and call-by-value semantics for them, then relate the ML-like semantics to the untyped semantics via *erasure* of type information.

The syntax of untyped terms is defined as follows:

$$m \quad ::= \quad x \mid \lambda x.m \mid m_1\, m_2 \mid \mathcal{C}(m) \mid \mathcal{X}(m)$$

The call-by-value evaluation semantics for untyped terms (u-cbv) is defined as follows:

$$\begin{array}{lcl} v & ::= & x \mid \lambda x.m \\ e & ::= & [] \mid e\, m \mid v\, e \end{array}$$

$$\begin{array}{lll} e[(\lambda x.m)\, v] & \hookrightarrow_{u\text{-}cbv} & e[[v/x]m] \\ e[\mathcal{C}(m)] & \hookrightarrow_{u\text{-}cbv} & e[m\, \lambda x.\, \mathcal{X}(e[x])] \\ e[\mathcal{X}(m)] & \hookrightarrow_{u\text{-}cbv} & m \end{array}$$

The call-by-name semantics for untyped terms (u-cbn) is defined similarly:

$$\begin{array}{lcl} v & ::= & \lambda x.m \\ e & ::= & [] \mid e\, m \end{array}$$

$$\begin{array}{lll} e[(\lambda x.m)\, m'] & \hookrightarrow_{u\text{-}cbn} & e[[m'/x]m] \\ e[\mathcal{C}(m)] & \hookrightarrow_{u\text{-}cbn} & e[m\, \lambda x.\, \mathcal{X}(e[x])] \\ e[\mathcal{X}(m)] & \hookrightarrow_{u\text{-}cbn} & m \end{array}$$

The ML-like semantics may be related to their untyped counterparts through the *erasure* of type information:

$$\begin{array}{rclrcl} x^\circ & = & x & & & \\ (\lambda x{:}A.M)^\circ & = & \lambda x.M^\circ & (M\,N)^\circ & = & M^\circ\, N^\circ \\ (\Lambda u{:}K.M)^\circ & = & M^\circ & (M\{A\})^\circ & = & M^\circ \\ (\mathcal{X}_A(M))^\circ & = & \mathcal{X}(M^\circ) & (\mathcal{C}_A(M))^\circ & = & \mathcal{C}(M^\circ) \end{array}$$

Erasure is extended to contexts by defining $[]^\circ = []$.

Lemma 3.15

1. $([M_2/x]M_1)^\circ = [M_2^\circ/x]M_1^\circ$, and $([A/u]M)^\circ = M^\circ$.
2. $C[M]^\circ = C^\circ[M^\circ]$.
3. If V is a ml-cbv (ml-cbn) value, then V° is a u-cbv (u-cbn) value.
4. If E is an ml-cbv (ml-cbn) evaluation context, then E° is an u-cbv (u-cbn) evaluation context.

Lemma 3.16

Let M_1 be a well-typed closed term.

1. If $M_1 \hookrightarrow_{ml\text{-}cbv} M_2$ or $M_1 \hookrightarrow_{ml\text{-}cbn} M_2$ by a β_Λ step, then $M_1^\circ = M_2^\circ$.
2. If $M_1 \hookrightarrow_{ml\text{-}cbv} M_2$ ($M_1 \hookrightarrow_{ml\text{-}cbn} M_2$) by other than a β_Λ step, then $M_1^\circ \hookrightarrow_{u\text{-}cbv} M_2^\circ$ ($M_1^\circ \hookrightarrow_{u\text{-}cbn} M_2^\circ$).
3. There exists a term M_2 such that $M_1 \hookrightarrow^*_{ml\text{-}cbv} M_2$ ($M_1 \hookrightarrow^*_{ml\text{-}cbn} M_2$) by a sequence of zero or more β_Λ steps such that M_2 is β_Λ irreducible.

Proof

1. Erasure eliminates β_Λ redices.
2. Erasure preserves β_λ, $\mathcal{C}$, and $\mathcal{X}$ steps.

3. Each β_Λ step reduces the number of constructor abstractions.

$\square$

With these facts in mind we may now relate the ML-like semantics to their untyped counterparts:

Theorem 3.17 (Simulation)
Let M_1 be a well-typed closed term.

1. If $M_1 \hookrightarrow_{\mathit{ml\text{-}cbv}} M_2$, then $M_1^\circ \hookrightarrow^{0,1}_{\mathit{u\text{-}cbv}} M_2^\circ$, and similarly for ml-cbn and u-cbn.
2. If $M_1^\circ \hookrightarrow_{\mathit{u\text{-}cbv}} m_2$, then $\exists M_2$ such that $M_1 \hookrightarrow^*_{\mathit{ml\text{-}cbv}} M_2$ and $M_2^\circ = m_2$, and similarly for ml-cbn and u-cbn.

Proof
We consider two illustrative cases. Suppose that $M_1 = E[(\lambda x{:}A.M)\,V] \hookrightarrow_{\mathit{ml\text{-}cbv}} E[[V/x]M] = M_2$. Then

$$\begin{array}{ccl} M_1^\circ & = & E^\circ[(\lambda x.M^\circ)\,V^\circ] \\ & \hookrightarrow_{\mathit{u\text{-}cbv}} & E^\circ[[V^\circ/x]M^\circ] \\ & = & E^\circ[([V/x]M)^\circ] \\ & = & (E[[V/x]M])^\circ \\ & = & M_2^\circ \end{array}$$

Suppose that $M_1^\circ \hookrightarrow_{\mathit{u\text{-}cbv}} m_2$. By Lemma 3.16 there exists M_3 such that M_3 is not reducible by a β_Λ step and $M_1 \hookrightarrow^*_{\mathit{ml\text{-}cbv}} M_3$ by a sequence of β_Λ steps. Consequently, $M_1^\circ = M_3^\circ$, and hence $M_3^\circ \hookrightarrow_{\mathit{u\text{-}cbv}} m_2$. This means M_3 must be ml-cbv reducible by other than a β_Λ step since the erasure of an ml-cbv value is irreducible under u-cbv. Hence $M_3 \hookrightarrow_{\mathit{ml\text{-}cbv}} M_2$ for some M_2 and $M_3^\circ \hookrightarrow_{\mathit{u\text{-}cbv}} M_2^\circ$ by Lemma 3.16. Since the u-cbv evaluation relation is a partial function, we have that $M_2^\circ = m_2$. $\square$

The ml-cbn and ml-cbn$'$ semantics coincide under erasure on programs.

Theorem 3.18 (Equivalence)
If P_1 and P_2 are programs such that $P_1^\circ = P_2^\circ$ then

1. If $P_1 \hookrightarrow_{\mathit{ml\text{-}cbn}} Q_1$ then $\exists Q_2$ such that $P_2 \hookrightarrow^*_{\mathit{ml\text{-}cbn'}} Q_2$ and $Q_1^\circ = Q_2^\circ$.
2. If $P_2 \hookrightarrow_{\mathit{ml\text{-}cbn'}} Q_2$ then $\exists Q_1$ such that $P_1 \hookrightarrow^*_{\mathit{ml\text{-}cbn}} Q_1$ and $Q_1^\circ = Q_2^\circ$.

Proof
First, prove a version of Theorem 3.17 with ml-cbn$'$ in place of ml-cbn. The result then follows from the simulation theorems. $\square$

4 Continuation-Passing Style

The *cps sub-language* of F_ω for a given semantics is a set of pure F_ω terms on which the call-by-name and call-by-value interpretations of that semantics coincide and which is closed under evaluation under those interpretations. Such terms are said to be *indifferent* to the by-name/by-value distinction (Plotkin, 1975). In this section we define the standard and ML-like cps sub-languages of F_ω. In addition we isolate a third cps-sublanguage, called *strict cps form*, on which all four operational interpretations of F_ω coincide.

4.1 Standard Cps Form

An analogue of untyped cps form, which we will call *standard cps form* (std-cps), may be defined for the standard semantics. The grammar for this subset of F_ω is as follows:

$$\begin{array}{lcl} W & ::= & x \mid \lambda x{:}A.N \mid \Lambda u{:}K.N \\ N & ::= & W \mid N\,W \mid N\{A\} \end{array}$$

The variable W ranges over *standard cps values* and the variable N ranges over *standard cps terms*.

Lemma 4.1

1. The std-cps sub-language is closed under std-cbv and std-cbn evaluation.
2. Evaluation of std-cps programs terminates with a std-cps value under both variants of the standard semantics.

Theorem 4.2 (Indifference)
The standard call-by-name and call-by-value semantics coincide on std-cps terms.

Proof
For terms in standard cps form the std-cbv and std-cbn semantics coincide with the following operational semantics:

$$\begin{array}{lcl} V & ::= & W \\ E & ::= & [] \mid E\,V \mid E\{A\} \end{array} \qquad \begin{array}{rcl} E[(\lambda x{:}A.N)\,V] & \hookrightarrow & E[[V/x]N] \\ E[(\Lambda u{:}K.N)\{A\}] & \hookrightarrow & E[[A/u]N] \end{array}$$

□

4.2 ML-cps Form

The ml-cbv and ml-cbn semantics do not coincide on standard cps terms. To see this, consider the following standard cps term:

$$(\lambda x{:}(\forall u{:}K.A).\,x)\,(\Lambda u{:}K.(\lambda y{:}A.y)c)$$

Under ml-cbv the innermost redex will be reduced first, whereas under ml-cbn the outermost will be reduced first. An analogue of untyped cps form for the ML-like semantics, which we call *ml-cps form*, is defined as follows:

$$\begin{array}{lcl} W & ::= & x \mid \lambda x{:}A.N \mid \Lambda u{:}K.W \\ N & ::= & W \mid N\,W \mid \Lambda u{:}K.N \mid N\{A\} \end{array}$$

It is easy to see that every ml-cps term is a standard cps term, and that every ml-cps value is a standard cps value. Note that if N is an ml-cps term, then N° is an untyped cps term, and if W is an ml-cps value, then W° is an untyped cps value, which was not the case for the standard cps form.

Lemma 4.3

1. The ml-cps sub-language is closed under both ml-cbv and ml-cbn evaluation.
2. Evaluation of ml-cps programs terminates with an ml-cps value under both ml-cbv and ml-cbn evaluation.

Theorem 4.4 (Indifference)

1. The ml-cbv and ml-cbn semantics coincide on ml-cps terms.
2. The std-cbv and std-cbn semantics coincide on ml-cps terms.

Proof

When restricted to terms in ml-cps form, the ml-cbv and ml-cbn semantics coincide with the following operational semantics:

$$\begin{array}{lll}
V & ::= & W \\
E & ::= & [\,] \mid E\,V \mid \Lambda u{:}K.E \mid E\{A\}
\end{array}
\qquad
\begin{array}{rcl}
E[(\lambda x{:}A.N)\,V] & \hookrightarrow & E[[V/x]N] \\
E[(\Lambda u{:}K.V)\{A\}] & \hookrightarrow & E[[A/u]V]
\end{array}$$

□

4.3 *Strict Cps Form*

The standard and ML-like semantics do not coincide on terms in ml-cps form. Consider the ml-cps term $\Lambda u{:}K.((\lambda x{:}A.x)c)$. This term is a value under std-cbv and std-cbn, but is not a value under either the ml-cbv or ml-cbn semantics. By further restricting ml-cps to avoid constructor abstractions over non-values, we obtain a subset of ml-cps called *strict cps form (s-cps)*, on which all four interpretations coincide:

$$\begin{array}{lll}
W & ::= & x \mid \lambda x{:}A.N \mid \Lambda u{:}K.W \\
N & ::= & W \mid N\,W \mid N\{A\}
\end{array}$$

Lemma 4.5

1. The strict cps sub-language is closed under both variants of the standard and ML-like semantics.
2. Evaluation of strict cps programs under either variant of either semantics terminates with a strict cps value.

Theorem 4.6 (Indifference)

Both variants of the standard and ML-like strategies all coincide on terms in strict cps form.

Proof

When restricted to terms in strict cps form, all four operational semantics coincide with the following semantics:

$$\begin{array}{lll}
V & ::= & W \\
E & ::= & [\,] \mid E\,V \mid E\{A\}
\end{array}
\qquad
\begin{array}{rcl}
E[(\lambda x{:}A.N)\,V] & \hookrightarrow & E[[V/x]N] \\
E[(\Lambda u{:}K.V)\{A\}] & \hookrightarrow & E[[A/u]V]
\end{array}$$

□

5 Conversion to Continuation-Passing Style

In this section we define the continuation-passing translation of $\mathsf{F}_\omega^{\mathcal{C}}$ into pure F_ω. The main idea of the translation is to explicitly represent the evaluation context as an expression of F_ω. The translation for a given semantics yields terms in the cps form for that semantics. Moreover, the control operators are eliminated in favor of direct manipulation of continuations.

Both forms of the standard semantics admit translations into strict cps form (and hence into standard cps form) that enjoy suitable generalizations of the Meyer-Wand typing properties (Meyer and Wand, 1985). In view of the unsoundness of the ML-like call-by-value semantics for $\mathsf{F}_\omega^{\mathcal{C}}$ we are unable to give a similar translation for this case, but rather only for the restricted language $\mathsf{F}_\omega^{\mathcal{C}-}$ introduced in Section 3.

5.1 Transformation of Constructors

In order to state the typing properties of the cps translation we must give a corresponding translation of types and constructors. These translations differ only in the treatment of function types (call-by-name and call-by-value variants) and of quantified types (standard and ML-like variants).

Definition 5.1

$$|A| \quad = \quad (A^*{\rightarrow}Ans){\rightarrow}Ans$$

$$\begin{array}{rclrcl} u^* & = & u & (\lambda u{:}K.A)^* & = & \lambda u{:}K.A^* \\ Ans^* & = & Ans & (A_1\,A_2)^* & = & A_1^*A_2^* \end{array}$$

$$\begin{array}{rclrcl} \multicolumn{3}{c}{\textit{Function types, call-by-value:}} & \multicolumn{3}{c}{\textit{Quantified types, standard:}} \\ (A_1{\rightarrow}A_2)^* & = & A_1^*{\rightarrow}|A_2| & (\forall u{:}K.A)^* & = & \forall u{:}K.|A| \\ \multicolumn{3}{c}{\textit{Function types, call-by-name:}} & \multicolumn{3}{c}{\textit{Quantified types, ML-like:}} \\ (A_1{\rightarrow}A_2)^* & = & |A_1|{\rightarrow}|A_2| & (\forall u{:}K.A)^* & = & \forall u{:}K.A^* \end{array}$$

The constructor transforms are extended to type assignments Γ by defining $\Gamma^*(x) = A^*$ and $|\Gamma|(x) = |A|$ whenever $\Gamma(x) = A$.

Lemma 5.2 (Compositional Translation)
The following equations hold for both variants of both semantics:

1. $([A_1/u]A_2)^* = [A_1^*/u]A_2^*$.
2. $|[A_1/u]A_2| = [A_1^*/u]\,|A_2|$.

The constructor transformations preserve kinds and equality:

Theorem 5.3 (Kind Preservation)
For both variants of both semantics:

1. If $\mathsf{F}_\omega \vdash \Delta \triangleright A : K$, then $\mathsf{F}_\omega \vdash \Delta \triangleright A^* : K$.
2. If $\mathsf{F}_\omega \vdash \Delta \triangleright A_1 = A_2 : K$, then $\mathsf{F}_\omega \vdash \Delta \triangleright A_1^* = A_2^* : K$.
3. If $\mathsf{F}_\omega \vdash \Delta \triangleright A : \Omega$, then $\mathsf{F}_\omega \vdash \Delta \triangleright |A| : \Omega$.
4. If $\mathsf{F}_\omega \vdash \Delta \triangleright A_1 = A_2 : \Omega$, then $\mathsf{F}_\omega \vdash \Delta \triangleright |A_1| = |A_2| : \Omega$.

5.2 Transformation of Terms

A cps translation is given by a translation for values, $(-)^*$, and a translation for general terms, $|-|$. These translations are defined by induction on the structure of typing derivations, rather than "raw" terms. This is largely a technical convenience since both the source and target languages are explicitly-typed, and we are only interested in the properties of well-typed terms. In defining a translation on typing derivations we must take account of *coherence*: since a given term may have several typing derivations, it is important that all translations are equivalent. Although this can be far from obvious in many cases (Breazu-Tannen, *et al.* 1991; Curien and Ghelli, 1990), in the present setting coherence is readily established. The only non-syntax-directed rule in the system is the rule of type equality, and uses of this rule affect only the type labels attached to variables. But since the operational semantics is insensitive to these labels, all translations are easily seen to be equivalent.

In order to simplify the presentation of the cps translations we adopt the following conventions. New variables introduced by the transform are assumed to be chosen so as to avoid capture. In cases where more than one clause of the transform applies (this only occurs in the optimized versions), the first one listed is to be chosen. When defining the transforms we suppress mention of the sub-derivations whenever possible in the interest of brevity.

5.2.1 Standard Call-by-Value

The cps transform for the standard call-by-value semantics is given in figure 1.

Theorem 5.4 (Typing)

1. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \rhd M : A$, then $|M|$ is a strict cps value such that $\mathsf{F}_\omega \vdash \Delta; \Gamma^* \rhd |M| : |A|$.
2. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \rhd V : A$, then V^* is a strict cps value such that $\mathsf{F}_\omega \vdash \Delta; \Gamma^* \rhd V^* : A^*$.

The correctness of the std-cbv cps transform for $\mathsf{F}_\omega^{\mathcal{C}}$ may be established by adapting methods introduced by Plotkin (1975) and Griffin (1990). The main idea is to consider an "optimized" cps translation in which most administrative redices (Plotkin, 1975) are eliminated during translation. The optimized translation is relativized to an explicitly-given continuation which is a representation of the current evaluation context.

The optimized std-cbv cps transform is given in figure 2. It satisfies essentially the same typing properties as the unoptimized version.

Theorem 5.5 (Typing)

1. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \rhd M : A$ and Y is a strict cps value such that $\mathsf{F}_\omega \vdash \Delta; \Gamma \rhd Y : A^* \to Ans$, then $|M|_Y$ is a strict cps term such that $\mathsf{F}_\omega \vdash \Delta; \Gamma^* \rhd |M|_Y : Ans$.
2. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta; \Gamma \rhd V : A$, then $V^\dagger$ is a strict cps value such that $\mathsf{F}_\omega \vdash \Delta; \Gamma^* \rhd V^\dagger : A^*$.

$$
\begin{array}{rcl}
|\Delta;\Gamma \rhd V : A| & = & \lambda k{:}A^*{\rightarrow}Ans.\, k\, V^* \\[1ex]
|\Delta;\Gamma \rhd M_1\, M_2 : A| & = & \lambda k{:}A^*{\rightarrow}Ans.\, |M_1|\,(\lambda x_1{:}(A_2{\rightarrow}A)^*. \\
 & & \quad |M_2|\,(\lambda x_2{:}A_2^*.x_1\, x_2\, k)), \\
\multicolumn{3}{c}{\textit{where } \Delta;\Gamma \rhd M_1 : A_2{\rightarrow}A \textit{ and } \Delta;\Gamma \rhd M_2 : A_2} \\[1ex]
|\Delta;\Gamma \rhd M\,\{A_1\} : [A_1/u]A_2| & = & \lambda k{:}([A_1/u]A_2)^*{\rightarrow}Ans. \\
 & & \quad |M|\,(\lambda x{:}(\forall u{:}K_1.A_2)^*.x\{A_1^*\}\, k) \\[1ex]
|\Delta;\Gamma \rhd \mathcal{X}_A(M) : A| & = & \lambda k{:}Ans^*{\rightarrow}Ans.\, |M|\,(\lambda m{:}A^*.m) \\
|\Delta;\Gamma \rhd \mathcal{C}_A(M) : A| & = & \lambda k{:}A^*{\rightarrow}Ans.\, |M|\, Y, \textit{ where} \\
Y & = & \lambda m{:}((\forall u{:}\Omega.A{\rightarrow}u){\rightarrow}A)^*.\, m\, Y'\, k, \textit{ and} \\
Y' & = & \Lambda u{:}\Omega.\lambda l{:}(A{\rightarrow}u)^*{\rightarrow}Ans. \\
 & & \quad l\,(\lambda x{:}A^*.\lambda k'{:}u^*{\rightarrow}Ans.k\, x) \\[1ex]
|\Delta;\Gamma \rhd M : A'| & = & |M|, \\
\multicolumn{3}{c}{\textit{where } \Delta;\Gamma \rhd M : A \textit{ and } \Delta \rhd A = A' : \Omega} \\[1ex]
(\Delta;\Gamma \rhd x : A)^* & = & x \\
(\Delta;\Gamma \rhd \lambda x{:}A.M : A{\rightarrow}A')^* & = & \lambda x{:}A^*.\, |M| \\
(\Delta;\Gamma \rhd \Lambda u{:}K.M : \forall u{:}K.A)^* & = & \Lambda u{:}K.\, |M| \\[1ex]
(\Delta;\Gamma \rhd V : A')^* & = & V^*, \textit{ where } \Delta;\Gamma \rhd V : A \textit{ and } \Delta \rhd A = A' : \Omega
\end{array}
$$

Fig. 1. The standard call-by-value transform

The optimized transform is related to the unoptimized transform through the notion of β_v *reduction* defined as follows:

$$
\begin{array}{rcl}
C[(\lambda x{:}A.M)\, V] & \rightarrow_{\beta_v} & C[[V/x]M] \\
C[(\Lambda u{:}K.M)\{A\}] & \rightarrow_{\beta_v} & C[[A/u]M]
\end{array}
$$

Notice that β_v reduction may occur in any context, rather than just an evaluation context.

Theorem 5.6 (Optimization)
$|M|\, Y \rightarrow^*_{\beta_v} |M|_Y$ and $V^* \rightarrow^*_{\beta_v} V^\dagger$.

The optimized transform is extended to evaluation contexts by considering the hole to be a non-value and defining $|\Delta;\Gamma \rhd [] : B|_Y = Y$. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta;\Gamma \rhd E : A$ (regarding the "hole" in E as having some type B) and $\mathsf{F}_\omega \vdash \Delta;\Gamma \rhd Y : A^*{\rightarrow}Ans$, then $|E|_Y$ is a strict cbv value such that $\mathsf{F}_\omega \vdash \Delta;\Gamma \rhd |E|_Y : B^*{\rightarrow}Ans$.

It is possible to regard any evaluation context as the composition of a series of *frames*:

$$F \quad ::= \quad []\, M \mid V\, [] \mid []\{A\}$$

Thus we may think of the evaluation context $(V[])\{A\}$ as the composition of frames $([]\{A\}) \circ (V[])$.

Lemma 5.7
Suppose that X is an expression that is not a std-cbv value. Then:

$$
\begin{array}{rcl}
|\Delta;\Gamma \triangleright V : A|_Y & = & Y\,V^\dagger \\[1ex]
|\Delta;\Gamma \triangleright V_1\,V_2 : A|_Y & = & V_1^\dagger\,V_2^\dagger\,Y \\
|\Delta;\Gamma \triangleright V_1\,M_2 : A|_Y & = & |M_2|_{Y'},\ \textit{where}\ \Delta;\Gamma \triangleright M_2 : A_2,\ \textit{and} \\
Y' & = & \lambda m_2{:}A_2^*.\,V_1^\dagger\,m_2\,Y \\
|\Delta;\Gamma \triangleright M_1\,M_2 : A|_Y & = & |M_1|_{Y'},\ \textit{where}\ \Delta;\Gamma \triangleright M_2 : A_2,\ \textit{and} \\
Y' & = & \lambda m_1{:}(A_2{\to}A)^*.\,|M_2|_{Y''},\ \textit{and} \\
Y'' & = & \lambda m_2{:}A_2^*.\,m_1\,m_2\,Y \\[1ex]
|\Delta;\Gamma \triangleright V\,\{A_1\} : [A_1/u]A_2|_Y & = & V^\dagger\,\{A_1^*\}\,Y \\
|\Delta;\Gamma \triangleright M\,\{A_1\} : [A_1/u]A_2|_Y & = & |M|_{Y'},\ \textit{where} \\
Y' & = & \lambda m{:}(\forall u{:}K.\,A_2)^*.\,m\,\{A_1^*\}\,Y \\[1ex]
|\Delta;\Gamma \triangleright \mathcal{X}_A(M) : A|_Y & = & |M|_{\lambda x:Ans.\,x} \\[1ex]
|\Delta;\Gamma \triangleright \mathcal{C}_A(M) : A|_Y & = & |M|_{Y'},\ \textit{where} \\
Y' & = & \lambda m{:}((\forall u{:}\Omega.\,A{\to}u){\to}A)^*. \\
 & & \quad (\lambda n{:}(\forall u{:}\Omega.\,A{\to}u)^*.\,m\,n\,Y)\,Y'',\ \textit{and} \\
Y'' & = & \Lambda u{:}\Omega.\,\lambda l{:}(A{\to}u)^*{\to}Ans. \\
 & & \quad l\,(\lambda x{:}A^*.\,\lambda k'{:}u^*{\to}Ans.\,Y\,x) \\[1ex]
|\Delta;\Gamma \triangleright M : A'|_Y & = & |M|_Y, \\
\multicolumn{3}{c}{\textit{where}\ \Delta;\Gamma \triangleright M : A\ \textit{and}\ \Delta \triangleright A = A' : \Omega} \\[1ex]
(\Delta;\Gamma \triangleright x : A)^\dagger & = & x \\
(\Delta;\Gamma \triangleright \lambda x{:}A_1.\,M : A_1{\to}A_2)^\dagger & = & \lambda x{:}A_1^*.\,\lambda k{:}A_2^*{\to}Ans.\,|M|_k \\
(\Delta;\Gamma \triangleright \Lambda u{:}K.\,M : \forall u{:}K.\,A)^\dagger & = & \Lambda u{:}K.\,\lambda k{:}A^*{\to}Ans.\,|M|_k \\[1ex]
(\Delta;\Gamma \triangleright V : A')^\dagger & = & V^\dagger,\quad \textit{where}\ \Delta;\Gamma \triangleright V : A\ \textit{and}\ \Delta \triangleright A = A' : \Omega
\end{array}
$$

Fig. 2. The optimized standard call-by-value transform

1. $|[]|_Y = Y$.
2. $E[X]$ is not a std-cbv value.
3. $|F[X]|_Y = |X|_{|F|_Y}$.

Theorem 5.8 (Decomposition)

1. If M is not a std-cbv value, then $|E[M]|_Y = |M|_{|E|_Y}$.
2. If V is a std-cbv value, then $|V|_{|E|_Y} \to^*_{\beta_v} |E[V]|_Y$.

Proof

By induction on E as a sequence of frames. Two illustrative cases are as follows:

$$
\begin{array}{rcl}
|E[M]|_Y & = & |F \circ E'[M]|_Y \\
 & = & |F[E'[M]]|_Y \\
 & = & |E'[M]|_{|F|_Y} \\
 & = & |M|_{|E'|_{|F|_Y}} \\
 & = & |M|_{|F\circ E'|_Y} \\
 & = & |M|_{|E|_Y}
\end{array}
$$

$$
\begin{array}{rcl}
|V|_{|E|_Y} & = & |V|_{|V_1\,[]|_Y} \\
& = & |V_1\,[]|_Y\, V^\dagger \\
& = & |[]|_{\lambda m_2{:}A_2^*.V_1^\dagger\, m_2\, Y}\, V^\dagger \\
& = & (\lambda m_2{:}A_2^*.V_1^\dagger\, m_2\, Y)\, V^\dagger \\
& \rightarrow_{\beta_v} & V_1^\dagger\, V^\dagger\, Y \\
& = & |V_1\, V|_Y \\
& = & |(V_1[])[V]|_Y \\
& = & |E[V]|_Y
\end{array}
$$

□

It follows that $|M|_{|E|_Y} \rightarrow^*_{\beta_v} |E[M]|_Y$ for all terms M.

Theorem 5.9 (Simulation)

If P is a program and $P \hookrightarrow^*_{\mathit{std\text{-}cbv}} Q$, then $|P|_{\lambda x:Ans.x} \rightarrow^*_{\beta_v} |Q|_{\lambda x:Ans.x}$. Moreover, each β-step on P induces at least one β_v-step on the converted form.

Proof

The main steps are to show that if $P \hookrightarrow_{\mathit{std\text{-}cbv}} Q$ by a β-step, then $|P|_Y \rightarrow^+_{\beta_v} |Q|_Y$, and if $P \hookrightarrow_{\mathit{std\text{-}cbv}} Q$ by either an $\mathcal{X}$ or $\mathcal{C}$ step, then $|P|_{\lambda x:Ans.x} \rightarrow^*_{\beta_v} |Q|_{\lambda x:Ans.x}$. □

Theorem 5.10 (Termination)

For any program P,

1. There exists a unique std-cbv value V such that $P \hookrightarrow^*_{\mathit{std\text{-}cbv}} V$.
2. If $P \hookrightarrow^*_{\mathit{std\text{-}cbv}} V$ then $|P|\,(\lambda x{:}Ans.x) \rightarrow^*_{\beta_v} V'$ where V' is such that $V^* \rightarrow^*_{\beta_v} V'$.

Proof

Part (1) is a consequence of simulation and the strong normalization property of F_ω. As previously remarked, any infinite std-cbv reduction sequence must contain infinitely many β-steps. Part (2) follows from the simulation and optimization theorems, together with the observation that $|V|_{\lambda x:Ans.x} = V^\dagger$. □

5.2.2 Standard Call-by-Name

The standard call-by-name semantics also admits a conversion into cps sharing essentially the same properties as are enjoyed by the standard call-by-value cps transformation. We have only to switch to the call-by-name variant of the constructor transform and modify the term transform by replacing the variable, application, and $\mathcal{C}$ clauses by the following clauses. Recall that under the call-by-name interpretation variables are not considered to be values.

Definition 5.11

$$
\begin{array}{rcl}
|\Delta;\Gamma \rhd x : A| & = & x \\[1em]
|\Delta;\Gamma \rhd M_1\, M_2 : A| & = & \lambda k{:}A^*{\to}Ans.\, |M_1|\,(\lambda x_1{:}(A_1{\to}A_2)^*.\, x_1\, |M_2|\, k) \\
\multicolumn{3}{c}{\textit{where } \Delta;\Gamma \rhd M_1 : A_2{\to}A \textit{ and } \Delta;\Gamma \rhd M_2 : A_2} \\[1em]
|\Delta;\Gamma \rhd \mathcal{C}_A(M) : A| & = & \lambda k{:}A^*{\to}Ans. \\
 & & \quad |M|\,(\lambda m{:}((\forall u{:}\Omega.\, A{\to}u){\to}A)^*.\, m\, Y\, k),\ \textit{where} \\
Y & = & \lambda l{:}(\forall u{:}\Omega.\, A{\to}u)^*{\to}Ans.\, l\, Y',\ \textit{and} \\
Y' & = & \Lambda u{:}\Omega.\, \lambda l{:}(A{\to}u)^*{\to}Ans. \\
 & & \quad l\,(\lambda x{:}|A|.\, \lambda k'{:}u^*{\to}Ans.\, x\, k) \\[1em]
(\Delta;\Gamma \rhd \lambda x{:}A.\, M : A{\to}A')^* & = & \lambda x{:}|A|.\, |M|
\end{array}
$$

Theorem 5.12 (Typing)

1. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta;\Gamma \rhd M : A$, then there exists a strict cps value $|M|$ such that $\mathsf{F}_\omega \vdash \Delta; |\Gamma| \rhd |M| : |A|$.
2. If $\mathsf{F}_\omega^{\mathcal{C}} \vdash \Delta;\Gamma \rhd V : A$, then there exists a strict cps value V^* such that $\mathsf{F}_\omega \vdash \Delta; |\Gamma| \rhd V^* : A^*$.

Proof
Analogous to the standard call-by-value case. $\square$

Theorem 5.13 (Simulation and Termination)
Let P be a program.

1. If $P \hookrightarrow^*_{\textit{std-cbn}} V$ then $|\,P|\,(\lambda x{:}Ans.x) \to^*_{\beta_v} V'$ where V' is such that $V^* \to^*_{\beta_v} V'$.
2. There exists a unique std-cbn value V such that $P \hookrightarrow^*_{\textit{std-cbn}} V$.

Proof
Similar to the call-by-value case. The necessary optimized transform is given in figure 3. $\square$

5.2.3 ML-Like Call-by-Value

The constructor transforms for the standard semantics are based on the idea that constructor applications are "serious" computations (in the sense of Reynolds (1972)). For the restricted language $\mathsf{F}_\omega^{\mathcal{C}-}$ the body of a polymorphic abstraction must be a value that is immediately passed to its continuation, and hence constructor application is fundamentally a trivial computation step. We are thus led to consider an alternative cps translation that more accurately reflects the computational behavior of $\mathsf{F}_\omega^{\mathcal{C}-}$ terms.

The definition of the alternative ml-cbv cps transform is the same as for the std-cbv cps transform, with the following differences. We employ the ML-like definition of the $(-)^*$ transform on constructors given in Definition 5.1, and take the following clauses for constructor abstraction and application and for $\mathcal{C}$:

$$
\begin{array}{rcl}
|\Delta;\Gamma \rhd V : A|_Y & = & Y\,V^{\dagger} \\[1ex]
|\Delta;\Gamma \rhd x : A|_Y & = & x\,Y \\[1ex]
|\Delta;\Gamma \rhd V_1\,M_2 : A|_Y & = & V_1^{\dagger}\,(\lambda k{:}A_2^*{\rightarrow}Ans.\,|M_2|_k)\,Y \\
 & \multicolumn{2}{l}{\textit{where } \Delta;\Gamma \rhd M_2 : A_2} \\
|\Delta;\Gamma \rhd M_1\,M_2 : A|_Y & = & |M_1|_{Y'},\ \textit{where } \Delta;\Gamma \rhd M_2 : A_2,\ \textit{and} \\
Y' & = & \lambda m{:}(A_2{\rightarrow}A)^*.\,m\,(\lambda k{:}A_2^*{\rightarrow}Ans.\,|M_2|_k)\,Y \\[1ex]
|\Delta;\Gamma \rhd V\,\{A_1\} : [A_1/u]A_2|_Y & = & V^{\dagger}\,\{A_1^*\}\,Y \\
|\Delta;\Gamma \rhd M\,\{A_1\} : [A_1/u]A_2|_Y & = & |M|_{Y'},\ \textit{where} \\
Y' & = & \lambda m{:}(\forall u{:}K.\,A_2)^*.\,m\,\{A_1^*\}\,Y \\[1ex]
|\Delta;\Gamma \rhd \mathcal{X}_A(M) : A|_Y & = & |M|_{\lambda x:Ans.\,x} \\[1ex]
|\Delta;\Gamma \rhd \mathcal{C}_A(M) : A|_Y & = & |M|_{Y'},\ \textit{where} \\
Y' & = & \lambda m{:}((\forall u{:}\Omega.\,A{\rightarrow}u){\rightarrow}A)^*.\,m\,Y''\,Y,\ \textit{and} \\
Y'' & = & \lambda l{:}(\forall u{:}\Omega.\,A{\rightarrow}u)^*{\rightarrow}Ans.\,l\,Y''',\ \textit{and} \\
Y''' & = & \Lambda u{:}\Omega.\,\lambda l{:}(A{\rightarrow}u)^*{\rightarrow}Ans. \\
 & & \quad l\,(\lambda x{:}|A|.\,\lambda k'{:}u^*{\rightarrow}Ans.\,x\,Y) \\[1ex]
|\Delta;\Gamma \rhd M : A'|_Y & = & |M|_Y, \\
 & \multicolumn{2}{l}{\textit{where } \Delta;\Gamma \rhd M : A \textit{ and } \Delta \rhd A = A' : \Omega} \\[1ex]
(\Delta;\Gamma \rhd \lambda x{:}A_1.\,M : A_1{\rightarrow}A_2)^{\dagger} & = & \lambda x{:}|A_1|.\,\lambda k{:}A_2^*{\rightarrow}Ans.\,|M|_k \\
(\Delta;\Gamma \rhd \Lambda u{:}K.\,M : \forall u{:}K.\,A)^{\dagger} & = & \Lambda u{:}K.\,\lambda k{:}A^*{\rightarrow}Ans.\,|M|_k \\[1ex]
(\Delta;\Gamma \rhd V : A')^{\dagger} & = & V^{\dagger},\ \textit{where } \Delta;\Gamma \rhd V : A \textit{ and } \Delta \rhd A = A' : \Omega
\end{array}
$$

Fig. 3. The optimized standard call-by-name transform

Definition 5.14

$$
\begin{array}{rcl}
|\Delta;\Gamma \rhd M\,\{A_1\} : [A_1/u]A_2| & = & \lambda k{:}([A_1/u]A_2)^*{\rightarrow}Ans. \\
 & & \quad |M|\,(\lambda m{:}(\forall u{:}K.A_2)^*.\,k\,(m\,\{A_1^*\})) \\[1ex]
|\Delta;\Gamma \rhd \mathcal{C}_A(M) : A| & = & \lambda k{:}A^*{\rightarrow}Ans.\,|M|\,(\lambda m{:}((\forall u{:}\Omega.\,A{\rightarrow}u){\rightarrow}A)^*. \\
 & & \quad m\,(\Lambda u{:}\Omega.\,\lambda x{:}A^*.\,\lambda k'{:}u^*{\rightarrow}Ans.\,k\,x)\,k) \\[1ex]
(\Delta;\Gamma \rhd \Lambda u{:}K.\,V : \forall u{:}K.\,A)^* & = & \Lambda u{:}K.\,V^*
\end{array}
$$

This transformation does not yield terms in strict, or even ML-like, cps form. In particular, terms of the form $k\,(x\,\{A\})$ arise in the transformation, violating the condition that arguments to functions are restricted to values. By regarding constructor applications as trivial computations (tantamount to values), we may regard the translation as yielding terms in *quasi-cps form*, which is defined as follows.

$$
\begin{array}{rcl}
W & ::= & x \mid \lambda x{:}A.N \mid \Lambda u{:}K.W \mid W\{A\} \\
N & ::= & W \mid N\,W \mid \Lambda u{:}K.N \mid N\{A\}
\end{array}
$$

The set of terms in quasi-cps form is closed under ml-cbv and ml-cbn evaluation.

However, ml-cbv and ml-cbn do not coincide on this subset; the term $(\lambda x{:}A.x)$ $((\Lambda u{:}K.W)\{A\})$ may be further evaluated under ml-cbv, but not under ml-cbn evaluation. However, the two semantics coincide under erasure:

Theorem 5.15
Let P_1 and P_2 be quasi-cps programs such that $P_1^\circ = P_2^\circ$. If $P_1 \hookrightarrow^*_{\mathit{ml\text{-}cbv}} Q_1$ and $P_2 \hookrightarrow^*_{\mathit{ml\text{-}cbn}} Q_2$, then there exists Q_1' and Q_2' such that $(Q_1')^\circ = (Q_2')^\circ$, $Q_1 \hookrightarrow^*_{\mathit{ml\text{-}cbv}} Q_1'$, and $Q_2 \hookrightarrow^*_{\mathit{ml\text{-}cbn}} Q_2'$.

Proof
The erasure of quasi-cps form gives untyped cps form. The result follows from the relationship between the ML-like semantics and the untyped semantics (see Theorem 3.17) and the fact that the untyped semantics coincide on untyped cps form. □

Theorem 5.16 (Typing)
1. If $\mathsf{F}_\omega^{\mathcal{C}-} \vdash \Delta;\Gamma \rhd M : A$, then there exists a quasi-cps value $|M|$ such that $\mathsf{F}_\omega^{-} \vdash \Delta;\Gamma^* \rhd |M| : |A|$.
2. If $\mathsf{F}_\omega^{\mathcal{C}-} \vdash \Delta;\Gamma \rhd V : A$, then there exists a quasi-cps value V^* such that $\mathsf{F}_\omega^{-} \vdash \Delta;\Gamma^* \rhd V^* : A^*$.

This transform is essentially a typed version of the untyped call-by-value cps transform.

Theorem 5.17
If $\mathsf{F}_\omega^{\mathcal{C}-} \vdash \Delta;\Gamma \rhd M : A$, then $|M|^\circ \to_\eta^* |M^\circ|_{ucbv}$.

Proof
By induction on typing derivations. One illustrative case is as follows:

$$\begin{array}{rcl} |M\,\{A\}|^\circ & = & \lambda k.|M|^\circ(\lambda m.(k\{A^*\}\,m)^\circ) \\ & = & \lambda k.|M|^\circ(\lambda m.k\,m) \\ & \to_\eta & \lambda k.|M|^\circ\,k \\ & \to_\eta & |M|^\circ \\ & = & |M^\circ|_{ucbv} \end{array}$$

□

6 Summary

We have described four different operational interpretations for $\mathsf{F}_\omega^{\mathcal{C}}$. Under the standard semantics, polymorphic abstractions are values and polymorphic instantiation is a significant computation step. Under the ML-like semantics, which are intended to model first erasing type information then evaluating using an untyped semantics, evaluation proceeds beneath polymorphic abstractions and polymorphic instantiation is essentially insignificant. We have analyzed these two semantics, considering a call-by-value and call-by-name variant for each, by means of the technique of cps transformation.

The standard semantics — both call-by-value and call-by-name variants — validate subject reduction, are terminating, and admit faithful, type-preserving transformations into continuation-passing style. We conclude that the standard semantics are semantically unproblematic, at least from the point of view of compilation and typing. These semantics have the significant advantage of being extensible to a more sophisticated set of primitive operations, in particular, those that make non-trivial use of type information at run time.

On the other hand, the ML-like call-by-value semantics is problematic — $\mathsf{F}_\omega^{\mathcal{C}}$, when evaluated under this semantics, fails to be sound. Restriction to the fragment $\mathsf{F}_\omega^{\mathcal{C}-}$ in which constructor abstractions are limited to values restores soundness at the cost of losing ml-cbv's uniqueness. (Std-cbv and ml-cbv coincide on this fragment.) We have presented an alternate cps transform for this fragment which treats constructor application as a trivial computation unlike the normal std-cbv cps transform.

7 Acknowledgements

We are grateful to Olivier Danvy, Andrzej Filinski, Timothy Griffin, Benjamin Pierce, Philip Wadler, and the referees for their comments and suggestions.

References

Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991.

Rod Burstall, David MacQueen, and Donald Sannella. HOPE: An experimental applicative language. In *Proceedings of the 1980 LISP Conference*, pages 136–143, Stanford, California, 1980. Stanford University.

Luca Cardelli. Typeful programming. Technical Report 45, DEC Systems Research Center, 1989.

Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of System F with subtyping. Research Report 80, Digital Systems Research Center, Palo Alto, California, December 1991.

William Clinger and Jonathan Rees. Revised^4 Report on the Algorithmic Language Scheme. *LISP Pointers*, 5(3):1–55, 1991.

Eric C. Cooper and J. Gregory Morrisett. Adding threads to Standard ML. Technical Report CMU-CS-90-186, School of Computer Science, Carnegie Mellon University, December 1990.

Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption. Technical Report LIENS–90–10, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, February 1990.

Bruce Duba, Robert Harper, and David MacQueen. Typing first-class continuations in ML. In *Eighteenth ACM Symposium on Principles of Programming Languages*, January 1991.

Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 10(2):235–271, 1992.

Michael J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, November 1993.

Projet Formel. CAML: The reference manual. Technical report, INRIA–ENS, June 1987.

Jean-Yves Girard. *Interprétation Fonctionnelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieure*. PhD thesis, Université Paris VII, 1972.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

Timothy Griffin. A formulae-as-types notion of control. In *Seventeenth ACM Symposium on Principles of Programming Languages*, San Francisco, CA, January 1990. ACM, ACM.

Robert Harper, Bruce Duba, and David MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3(4):465–484, October 1993. (See also (Duba *et al.*, 1992).).

Robert Harper and Mark Lillibridge. Explicit polymorphism and CPS conversion. In *Twentieth ACM Symposium on Principles of Programming Languages*, pages 206–219, Charleston, SC, January 1993. ACM, ACM.

Robert Harper and Mark Lillibridge. Polymorphic type assignment and CPS conversion. In Olivier Danvy and Carolyn Talcott, editors, *Proceedings of the ACM SIGPLAN Workshop on Continuations CW92*, pages 13–22, Stanford, CA 94305, June 1992. Department of Computer Science, Stanford University. Published as technical report STAN–CS–92–1426.

Robert Harper and Mark Lillibridge. Polymorphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6(4):361–380, November 1993. (See also (Harper and Lillibridge, 1992).).

Robert Harper and John C. Mitchell. On the type structure of Standard ML. *ACM Transactions on Programming Languages and Systems*, 15(2):211–252, April 1993.

Robert Harper and Greg Morrisett. Compiling with non-parametric polymorphism. Technical Report CMU–CS–94–122, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1994. (Also published as Fox Memorandum CMU–CS–FOX–94–03).

Christopher T. Haynes, Daniel P. Friedman, and Mitchell Wand. Obtaining coroutines from continuations. *Journal of Computer Languages*, 11:143–153, 1986.

Paul Hudak and Philip Wadler. Report on the Programming Language Haskell, Version 1.0. Research Report YALEU/DCS/RR–777, Yale University, April 1990.

D. Kranz, R. Kelsey, J. Rees, P. Hudak, J.Philbin, and N. Adams. Orbit: An optimizing compiler for Scheme. In *Proc. SIGPLAN Symposium on Compiler Construction*, pages 219–233. ACM SIGPLAN, 1986.

Xavier Leroy. Unboxed objects and polymorphic typing. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Albuquerque*, pages 177–188. ACM Press, January 1992.

Xavier Leroy and Michel Mauny. The Caml Light system, version 0.5 — documentation and user's guide. Technical Report L-5, INRIA, 1992.

David MacQueen. Using dependent types to express modular structure. In *Thirteenth ACM Symposium on Principles of Programming Languages*, 1986.

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

Robin Milner. A theory of type polymorphism in programming languages. *Journal of Computer and System Sciences*, 17:348–375, 1978.

Robin Milner and Mads Tofte. *Commentary on Standard ML*. MIT Press, 1991.

Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

John Mitchell and Robert Harper. The essence of ML. In *Fifteenth ACM Symposium on*

Principles of Programming Languages, San Diego, California, January 1988.

Benjamin C. Pierce. Intersection types and bounded polymorphism. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, pages 346–360, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664. (To appear in *Mathematical Structures in Computer Science*, 1995.)

Gordon Plotkin. Call-by-name, call-by-value, and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

John H. Reppy. CML: A higher-order concurrent language. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 293–305, June 1991.

John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Conference Record of the 25th National ACM Conference*, pages 717–740, Boston, August 1972. ACM.

John C. Reynolds. Towards a theory of type structure. In *Colloq. sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1974.

John C. Reynolds. Preliminary design of the programming language Forsythe. Technical Report CMU–CS–88–159, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1988.

John C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3/4):233–248, November 1993.

Guy L. Steele, Jr. RABBIT: A compiler for SCHEME. Technical Report Memo 474, MIT AI Laboratory, 1978.

Mads Tofte. Type inference for polymorphic references. *Information and Computation*, 89:1–34, November 1990.

A Rules for $\mathsf{F}_\omega^{\mathcal{C}}$

Definition A.1 (Constructor Context Formation Rules)

$$\triangleright\, \emptyset \tag{C-EMPTY}$$

$$\frac{\triangleright\, \Delta \quad u \notin \mathrm{dom}(\Delta)}{\triangleright\, \Delta, u{:}K} \tag{C-EXTEND}$$

Definition A.2 (Term Context Formation Rules)

$$\frac{\triangleright\, \Delta}{\Delta \triangleright \emptyset} \tag{T-EMPTY}$$

$$\frac{\Delta \triangleright \Gamma \quad \Delta \triangleright A : \Omega \quad x \notin \mathrm{dom}(\Gamma)}{\Delta \triangleright \Gamma, x{:}A} \tag{T-EXTEND}$$

Definition A.3 (Constructor Formation Rules)

$$\frac{\triangleright\, \Delta}{\Delta \triangleright u : \Delta(u)} \tag{C-VAR}$$

$$\frac{\Delta \triangleright A_1 : \Omega \quad \Delta \triangleright A_2 : \Omega}{\Delta \triangleright A_1 \to A_2 : \Omega} \tag{C-ARR}$$

$$\frac{\Delta, u{:}K \triangleright A : \Omega}{\Delta \triangleright \forall u{:}K.A : \Omega} \tag{C-ALL}$$

$$\frac{\Delta, u{:}K_1 \triangleright A : K_2}{\Delta \triangleright \lambda u{:}K_1.A : K_1 \Rightarrow K_2} \qquad \text{(C-ABS)}$$

$$\frac{\Delta \triangleright A_1 : K_2 \Rightarrow K \quad \Delta \triangleright A_2 : K_2}{\Delta \triangleright A_1\, A_2 : K} \qquad \text{(C-APP)}$$

Definition A.4 (Constructor Equality Rules)

$$\frac{\Delta \triangleright A : K}{\Delta \triangleright A = A : K} \qquad \text{(REFL)}$$

$$\frac{\Delta \triangleright A_1 = A_2 : K}{\Delta \triangleright A_2 = A_1 : K} \qquad \text{(SYMM)}$$

$$\frac{\Delta \triangleright A_1 = A_2 : K \quad \Delta \triangleright A_2 = A_3 : K}{\Delta \triangleright A_1 = A_3 : K} \qquad \text{(TRANS)}$$

$$\frac{\Delta \triangleright A_1 = A_1' : \Omega \quad \Delta \triangleright A_2 = A_2' : \Omega}{\Delta \triangleright A_1 \to A_2 = A_1' \to A_2' : \Omega} \qquad \text{(C-ARR-EQ)}$$

$$\frac{\Delta, u{:}K \triangleright A = A' : \Omega}{\Delta \triangleright \forall u{:}K.A = \forall u{:}K.A' : \Omega} \qquad \text{(C-ALL-EQ)}$$

$$\frac{\Delta, u{:}K_1 \triangleright A = A' : K_2}{\Delta \triangleright \lambda u{:}K_1.A = \lambda u{:}K_1.A' : K_1 \Rightarrow K_2} \qquad \text{(C-ABS-EQ)}$$

$$\frac{\Delta \triangleright A_1 = A_1' : K_2 \Rightarrow K \quad \Delta \triangleright A_2 = A_2' : K_2}{\Delta \triangleright A_1\, A_2 = A_1'\, A_2' : K} \qquad \text{(C-APP-EQ)}$$

$$\frac{\Delta, u{:}K_1 \triangleright A_2 : K_2 \quad \Delta \triangleright A_1 : K_1}{\Delta \triangleright (\lambda u{:}K_1.A_2)\, A_1 = [A_1/u]A_2 : K_2} \qquad \text{(C-BETA)}$$

$$\frac{\Delta \triangleright A : K_1 \Rightarrow K_2 \quad u \notin \mathrm{dom}(\Delta)}{\Delta \triangleright \lambda u{:}K_1.A\, u = A : K_1 \Rightarrow K_2} \qquad \text{(C-ETA)}$$

Definition A.5 (Term Formation Rules)

$$\frac{\Delta \triangleright \Gamma}{\Delta; \Gamma \triangleright x : \Gamma(x)} \qquad \text{(T-VAR)}$$

$$\frac{\Delta; \Gamma, x{:}A_1 \triangleright M : A_2}{\Delta; \Gamma \triangleright \lambda x{:}A.M : A_1 \to A_2} \qquad \text{(T-ABS)}$$

$$\frac{\Delta; \Gamma \triangleright M_1 : A_2 \to A \quad \Delta; \Gamma \triangleright M_2 : A_2}{\Delta; \Gamma \triangleright M_1\, M_2 : A} \qquad \text{(T-APP)}$$

$$\frac{\Delta, u{:}K; \Gamma \triangleright M : A \quad \Delta \triangleright \Gamma}{\Delta; \Gamma \triangleright \Lambda u{:}K.M : \forall u{:}K.A} \qquad \text{(T-CABS)}$$

$$\frac{\Delta; \Gamma \triangleright M : \forall u{:}K.A' \quad \Delta \triangleright A : K}{\Delta; \Gamma \triangleright M\{A\} : [A/u]A'} \qquad \text{(T-CAPP)}$$

$$\frac{\Delta \rhd A : \Omega \qquad \Delta ; \Gamma \rhd M : Ans}{\Delta ; \Gamma \rhd \mathcal{X}_A(M) : A} \qquad \text{(T-ABORT)}$$

$$\frac{\Delta ; \Gamma \rhd M : (\forall u{:}\Omega . A \to u) \to A \qquad u \notin dom(\Delta)}{\Delta ; \Gamma \rhd \mathcal{C}_A(M) : A} \qquad \text{(T-CALLCC)}$$

$$\frac{\Delta ; \Gamma \rhd M : A \quad \Delta \rhd A = A' : \Omega}{\Delta ; \Gamma \rhd M : A'} \qquad \text{(T-EQ)}$$