Proving termination of evaluation for System F with control operators

Małgorzata Biernacka
Institute of Computer Science
University of Wrocław
mabi@cs.uni.wroc.pl

Dariusz Biernacki
Institute of Computer Science
University of Wrocław
dabi@cs.uni.wroc.pl

Sergueï Lenglet
Université de Lorraine, LORIA
serguei.lenglet@univ-lorraine.fr

Marek Materzok
Institute of Computer Science
University of Wrocław
marek.materzok@cs.uni.wroc.pl

We present new proofs of termination of evaluation in reduction semantics (i.e., a small-step operational semantics with explicit representation of evaluation contexts) for System F with control operators. We introduce a modified version of Girard's proof method based on reducibility candidates, where the reducibility predicates are defined on values and on evaluation contexts as prescribed by the reduction semantics format. We address both abortive control operators (*callcc*) and delimited-control operators (*shift* and *reset*) for which we introduce novel polymorphic type systems, and we consider both the call-by-value and call-by-name evaluation strategies.

1 Introduction

Termination of reductions is one of the crucial properties of typed λ-calculi. When considering a λ-calculus as a deterministic programming language, one is usually interested in termination of reductions according to a given evaluation strategy, such as call by value or call by name, rather than in more general normalization properties. A convenient format to specify such strategies is reduction semantics, i.e.,a form of operational semantics with explicit representation of evaluation (reduction) contexts [14], where the evaluation contexts represent continuations [10]. Reduction semantics is particularly convenient for expressing non-local control effects and has been most successfully used to express the semantics of control operators such as *callcc* [14], or *shift* and *reset* [5].

For simply-typed languages with control operators, it is common to prove termination of evaluation (and of normalization in general) by translating, in a reduction-preserving way, terms in the source language to a target language for which the normalization property has been established before [16, 27]. Such indirect proofs in general can be cumbersome and, as argued by Ikeda and Nakazawa [19], they can be error-prone.

In a previous work [3, 4], it has been shown that a context-based variant of Tait's proof method based on reducibility predicates [28, 29] allows for direct and concise proofs of termination of evaluation in reduction semantics for the simply-typed λ-calculus with control operators, be they abortive or delimited. Unlike translation-based proofs, the context-based proof method directly takes advantage of the format of the reduction semantics, where the key role is played by evaluation contexts. So, for instance, in order to prove termination of evaluation for the simply-typed λ-calculus under call by value using the context-based method, one defines mutually inductively reducibility predicates on values (normal forms) as well as on evaluation contexts. The termination result then follows by induction on well-typed terms, where the reasoning is driven by the control flow of a typical evaluator in continuation-passing style [3, 4].

Submitted to:
COS 2013

© M. Biernacka, D. Biernacki, S. Lenglet & M. Materzok
This work is licensed under the
Creative Commons Attribution License.

In this article, we show that the context-based method can be generalized from reducibility predicates to reducibility candidates, and therefore it provides simple proofs of termination of evaluation for System F with control operators. Just as for the simply-typed λ-calculi, normalization for polymorphic λ-calculi with control operators has been mainly established indirectly, via translations to strongly normalizing calculi: Harper and Lillibridge reduced termination of call-by-value evaluation for F_ω with *abort* and *callcc* to normalization in F_ω [18] by a CPS translation, Parigot reduced strong normalization of the second-order $\lambda\mu$-calculus to strong normalization of the simply-typed λ-calculus [25], Danos et al. reduced strong normalization of the second-order classical logic to strong normalization of linear logic [8], and Kameyama and Asai reduced strong normalization of System F with *shift* and *reset* under the standard semantics to strong normalization of System F [20].

On the other hand, Parigot directly proved strong normalization of the second-order $\lambda\mu$-calculus using another variant of the reducibility candidates [25]—we discuss this result in Section 2.3. Later on, further adaptations of Tait-Girard's method have been proposed and applied to various flavors of second-order logic [9, 21, 22, 24]. In particular, following Girard, the techniques of orthogonality have been used as a framework in which the concepts of the original reducibility method can be phrased. The use of orthogonality induces the notion of context which is understood as a sequence of terms—roughly corresponding to call-by-name contexts. Reducibility candidates are then defined in terms of TT-closed sets. In contrast to this approach, we consider concrete evaluation strategies and our contexts come directly from the reduction semantics (either call-by-name or call-by-value, and the contexts can be layered in the delimited-control case), and in particular our reducibility candidates only contain values and are not TT-closed. Another related work is the proof of strong normalization for Moggi's computational calculus given by Lindley and Stark [23] who have introduced the operation of TT-lifting in order to interpret computational types. This operation seems to correspond to our definition of reducibility for one layer of (reduction) contexts (see Section 2.3). However, in strong normalization, the notion of context is to be understood as a means of syntactically splitting a term rather than "the remaining computation." In particular, we do not analyze the reducibility of terms forming reduction contexts.

The calculi we consider are System F with *callcc* under call by value and call by name as well as System F with *shift* and *reset* under call by value and call by name, in each case with the standard semantics, where, unlike in the ML-like semantics, evaluation does not proceed under polymorphic abstraction [18]. The type system for *callcc* is inspired by that of Harper and Lillibridge [18], whereas the type systems for *shift* and *reset* (one for each evaluation strategy) are new and they generalize Asai and Kameyama's type system [2] in that they allow for polymorphic abstractions over arbitrary expressions, not only pure ones. It is worth noting that, as in the simply-typed case [3, 4], the context-based proofs we present in this article have the structure of an evaluator in continuation-passing style.

We would like to stress that the semantics we consider in this article are not instances of abstract machines working on explicit decompositions of terms, where the process of decomposition is built in the transitions of the system. Instead, we rely on the higher-level reduction semantics approach where the operations of decomposition and recomposition of terms are left implicit. Consequently, the type systems we consider are in the form of natural deduction rather than in sequent calculus [7].

The rest of this article is organized as follows. In Section 2, we present System F with abortive control operators and we prove termination of evaluation under the call-by-value and call-by-name evaluation strategies for this system. We also relate this result to Parigot's work [25]. In Section 3, we present System F with delimited-control operators and we prove termination of evaluation under call by value and call by name. We also relate our type systems to Asai and Kameyama's [2]. In Section 4, we conclude.

2 System F with abortive control operators

In this section, we present a context-based proof of termination for call-by-value evaluation in System F extended with the control operator *callcc*. We use a variant of Girard's method of reducibility candidates, where in particular we define reducibility predicates for reduction contexts.

2.1 Syntax and Semantics

We consider the explicitly typed System F under the call-by-value reduction strategy, which we extend with the binder version of the *callcc* operator (denoted $\mathscr{K}$) and with a construct to apply captured continuations $\hookleftarrow$, similar to the *throw* construct of SML/NJ [17]. We call this language λ_v^F.
The syntax of terms, term types, and call-by-value contexts of λ_v^F is defined as follows:

$$\begin{array}{rrcl}
\text{Terms:} & t & ::= & x \mid \lambda x^S.t \mid t\,t \mid \Lambda X.t \mid t\{S\} \mid \mathscr{K}k.t \mid k \hookleftarrow t \mid \ulcorner E \urcorner \hookleftarrow t \\
\text{Term types:} & S & ::= & X \mid S \to S \mid \forall X.S \\
\text{CBV contexts:} & E & ::= & [\,] \mid (\lambda x^S.t)\,E \mid E\,t \mid E\,\{S\} \mid \ulcorner E \urcorner \hookleftarrow E \\
\text{Values:} & v & ::= & \lambda x^S.t \mid \Lambda X.t
\end{array}$$

We let x range over term variables, k range over continuation variables, and X range over type variables, and we assume the three sets of variables are pairwise disjoint. We use capital letters starting from S to denote term types. The term $\Lambda X.t$ quantifies over the type variable X, while the term $t\{S\}$ instantiates such quantification with type S. The term $\mathscr{K}k.t$ denotes the *callcc* operator that binds a captured context (representing a continuation) to the variable k and makes it available in its body t. In turn, constructs of the form $k \hookleftarrow t$ and $\ulcorner E \urcorner \hookleftarrow t$ denote the operation of *throwing* the term t to a continuation variable k and to the captured context $\ulcorner E \urcorner$, respectively. The use of $\ulcorner \cdot \urcorner$ indicates that a context is reified as a term, as opposed to its role as the representation of the "rest of the program."

Expressions of the form $\ulcorner E \urcorner \hookleftarrow t$ are not allowed in source programs (because we do not let programmers handle contexts explicitly), but they may occur during evaluation. In the sequel, it will be useful to distinguish the subset of *plain terms*, i.e., terms without any subterm of the form $\ulcorner E \urcorner \hookleftarrow t$.

An abstraction $\lambda x^S.t$ (resp., $\Lambda X.t$, $\mathscr{K}k.t$) binds x (resp., X, k) in t, and a type $\forall X.S$ binds X in S. We write $ftv(S)$ for the set of free type variables occurring in type S, defined in the usual way. The definitions of free term variables, free type variables, and free continuation variables of a term are also standard. A term is *closed* if it does not have any free variable of any kind. We identify terms and types up to α-conversion of their bound variables.

The syntax of reduction contexts encodes the reduction strategy, here—call by value. The contexts can be seen as "terms with a hole", and are represented inside-out. Informally, $[\,]$ denotes the empty context, $(\lambda x^S.t)\,E$ represents $E[(\lambda x^S.t)\,[\,]]$ with the hole indicated by $[\,]$, $E\,t$ represents $E[[\,]\,t]$, $E\,\{S\}$ represents $E[[\,]\{S\}]$, and $\ulcorner E_0 \urcorner \hookleftarrow E$ represents $E[\ulcorner E_0 \urcorner \hookleftarrow [\,]]$. A reduction context is closed if and only if all its components (terms, types, or contexts) are closed. We make the meaning of contexts precise by defining a function *plug* which maps a term and a context to the term which is obtained by putting the term in the hole of the context:

$$\begin{array}{rcl}
plug\,(t,[\,]) & = & t \\
plug\,(t_0,(\lambda x^S.t)\,E) & = & plug\,((\lambda x^S.t)\,t_0,E) \\
plug\,(t_0,E\,t_1) & = & plug\,(t_0\,t_1,E) \\
plug\,(t,E\,\{S\}) & = & plug\,(t\{S\},E) \\
plug\,(t,\ulcorner E_0 \urcorner \hookleftarrow E) & = & plug\,(\ulcorner E_0 \urcorner \hookleftarrow t,E)
\end{array}$$

We write $E[t]$ for the result of plugging t in the context E (i.e., the result of $plug\,(t,E)$).

A program p is a closed plain term. When $p = E[t]$, we say that p decomposes into term t in the context E. In general, a program can be decomposed into a term in a context in more than one way. For example, the program $(\lambda x^S.t_0)\,t_1$ can be represented by term t_1 in context $(\lambda x^S.t_0)\,[\,]$, or by term $(\lambda x^S.t_0)$ in context $[\,]\,t_1$, or by term $(\lambda x^S.t_0)\,t_1$ in context $[\,]$.

The one-step reduction relation in the call-by-value strategy is defined on programs by the following rules:

$$\begin{array}{rcll} E[(\lambda x^S.t)\,v] & \to_{\mathrm{v}} & E[t\{v/x\}] & (\beta_v) \\ E[(\Lambda X.t)\{S\}] & \to_{\mathrm{v}} & E[t\{S/X\}] & (\beta_T) \\ E[\mathscr{K}\,k.t] & \to_{\mathrm{v}} & E[t\{\ulcorner E\urcorner/k\}] & (callcc) \\ E_1[\ulcorner E_0\urcorner \hookleftarrow v] & \to_{\mathrm{v}} & E_0[v] & (throw_v) \end{array}$$

where $t\{v/x\}$ (resp., $t\{S/X\}$, $t\{\ulcorner E\urcorner/k\}$) is the usual capture-avoiding substitution of value v (resp., of type S, of context $\ulcorner E\urcorner$) for variable x (resp., for X, for k) in t. The rules (β_v) and (β_T) are standard in System F; in addition, we introduce the rule $(callcc)$, where the current context E is captured by the *callcc* operator and bound to the continuation variable k, and the rule $(throw_v)$, where a previously captured context E_0 is restored as the current context, and the context E_1 is discarded (the latter fact shows the abortive character of the *callcc* operator). The plugged terms on the left-hand side of the arrow in the above rules are called redexes, and are ranged over by r. Note that the reduction relation is not compatible, i.e., it only applies to entire programs (due to the context capture in the rule $(callcc)$).

We define the call-by-value evaluation relation as the reflexive and transitive closure of the relation $\to_{\mathrm{v}}$. The expected result of evaluation is a value.

The reduction relation $\to_{\mathrm{v}}$ is deterministic; this property is ensured by the unique-decomposition lemma. We could state this lemma in a general version for all terms, but in order to consider only well-behaved programs and simplify the statement of the lemma, we choose to postpone it to the next section where we define well-typed programs.

2.2 Type System

We define a type system for λ_v^F that is an extension of the type system for the lambda calculus introduced by Biernacka and Biernacki [3], where types are assigned to terms as well as to contexts. The syntax of context types is $\neg S$. Roughly, the type $\neg S$ of a context E indicates that any well-typed term of type S can be plugged in E. The answer type of a context need not be specified, and it is often taken to be $\bot$ to reflect the fact that continuations never return.[1] The answer type of a closed evaluation context E of type $\neg S$ can be determined by typing the expression $E[x]$ for a fresh variable x of type S.

We let Γ range over type environments for term variables (i.e., lists of pairs of the form $x:S$), and we let Δ range over type environments for continuation variables (i.e., lists of pairs of the form $k:C$). For $\Gamma = x_1:S_1,\ldots,x_n:S_n$ and $\Delta = k_1:\neg S_1,\ldots,k_n:\neg S_n$, we define $ftv\,(\Gamma) := \cup_{i\in\{1,\ldots,n\}} ftv\,(S_i)$ and $ftv\,(\Delta) := \cup_{i\in\{1,\ldots,n\}} ftv\,(S_i)$. The typing rules for terms and contexts are shown in Figure 1.

We can now state the unique-decomposition lemma that ensures the determinism of the reduction relation $\to_{\mathrm{v}}$, and progress of reduction:

Lemma 1 (Unique decomposition)**.** *For all well-typed programs p, p either is a value, or it decomposes uniquely into a context E and a redex r, i.e., $p = E[r]$.*

[1] This decision has more serious implications when a type system is studied from a logical perspective via the Curry-Howard isomorphism (see for example [1]). However, we do not take this viewpoint in this article.

Typing terms ($S ::= X \mid S \to T \mid \forall X.S$):

$$\frac{}{\Gamma, x : S; \Delta \vdash x : S} \qquad \frac{\Gamma, x : S; \Delta \vdash t : T}{\Gamma; \Delta \vdash \lambda x^S.t : S \to T} \qquad \frac{\Gamma; \Delta \vdash t_0 : S \to T \qquad \Gamma; \Delta \vdash t_1 : S}{\Gamma; \Delta \vdash t_0\, t_1 : T}$$

$$\frac{\Gamma; \Delta \vdash t : S \qquad X \notin ftv(\Gamma)}{\Gamma; \Delta \vdash \Lambda X.t : \forall X.S} \qquad \frac{\Gamma; \Delta \vdash t : \forall X.S}{\Gamma; \Delta \vdash t\{T\} : S\{T/X\}} \qquad \frac{\Gamma; \Delta, k : \neg S \vdash t : S}{\Gamma; \Delta \vdash \mathscr{K} k.t : S}$$

$$\frac{\Gamma; \Delta, k : \neg S \vdash t : S}{\Gamma; \Delta, k : \neg S \vdash k \hookleftarrow t : T} \qquad \frac{\Gamma; \Delta \vdash E : \neg S \qquad \Gamma; \Delta \vdash t : S}{\Gamma; \Delta \vdash \ulcorner E \urcorner \hookleftarrow t : T}$$

Typing contexts ($C ::= \neg S$):

$$\frac{}{\Gamma; \Delta \vdash [\,] : \neg S} \qquad \frac{\Gamma; \Delta \vdash \lambda x^S.t : S \to T \qquad \Gamma; \Delta \vdash E : \neg T}{\Gamma; \Delta \vdash (\lambda x^S.t)\, E : \neg S} \qquad \frac{\Gamma; \Delta \vdash t : S \qquad \Gamma; \Delta \vdash E : \neg T}{\Gamma; \Delta \vdash E\, t : \neg(S \to T)}$$

$$\frac{\Gamma; \Delta \vdash E : \neg(S\{T/X\})}{\Gamma; \Delta \vdash E\,\{T\} : \neg(\forall X.S)} \qquad \frac{\Gamma; \Delta \vdash E_0 : \neg S \qquad \Gamma; \Delta \vdash E_1 : \neg T}{\Gamma; \Delta \vdash \ulcorner E_0 \urcorner \hookleftarrow E_1 : \neg S}$$

Figure 1: Typing rules for System F with abortive control operators

As for the subject reduction property, it is a more subtle issue. In the reduction rule $(throw_v)$, the current evaluation context E_1 is replaced by another context E_0, where the answer types of the two contexts do not have to be related in any way. Therefore, as observed before [3, 17, 30], in general subject reduction does not hold for languages with reduction and typing rules for continuation invocation similar to the ones presented in this work. However, if we assume that all reified contexts in a given term have the same answer type as the type of the term itself—which is the case for all terms in the reduction sequence starting in a plain term—subject reduction will be recovered [3, 17, 30]. Such an assumption can be made implicit [17] or explicit in a refined type system that controls the answer types of evaluation contexts [3, 30]. It can be shown that from subject reduction of such a refined type system strong type soundness for plain terms in the original type system follows [3, 30].

2.3 Termination

We now prove termination of the call-by-value evaluation for λ_v^F, using a context variant of Girard's method of reducibility candidates [15]. Our definition of a reducibility candidate is simpler than in Girard's proof of strong normalization for system F, because we are interested only in the termination of the call-by-value evaluation, not in strong normalization. Moreover, we exploit the structure of the call-by-value continuation-passing style [16, 18, 26] that underlies the semantics of the language we consider, and therefore the central role in the proof is played by predicates on evaluation contexts (representing continuations) and on values, and not on arbitrary terms.

First, we define the normalization predicate $\mathscr{N}(p)$ as follows:

$$\mathscr{N}(p) := \exists v.\, p \to_{\mathrm{v}}^* v,$$

i.e., a program p normalizes ($\mathscr{N}(p)$ holds) if it reduces in several steps to a value.

Definition 1 (Reducibility candidate)**.** *A* reducibility candidate $\mathscr{R}$ *of type* S *is any set of closed values of type* S*.*

We write $\mathscr{RC}(S)$ for the set of reducibility candidates of type S. For each reducibility candidate $\mathscr{R}$ of type S, we define the associated predicate $\mathscr{C}_{\mathscr{R}}$ on closed contexts of type $\neg S$ as follows:

$$\mathscr{C}_{\mathscr{R}}(E) := \forall v.\, v \in \mathscr{R} \to \mathscr{N}(E[v])$$

As in the original proof, we introduce the notion of parametric reducibility candidates. However, we base our definition on the CPS interpretation of terms rather than on the direct-style interpretation. Let S be a type, $ftv(S) \subseteq \vec{X}$ [2] , $\vec{T}$ be a sequence of types of the same size as $\vec{X}$, and $\vec{\mathscr{R}}$ be such that $\mathscr{R}_i \in \mathscr{RC}(T_i)$. We define the parametric reducibility candidate $RED_S[\vec{\mathscr{R}}/\vec{X}]$ by induction on S:

$$\begin{array}{rll} v \in RED_{X_i}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & v \in \mathscr{R}_i \\ v_0 \in RED_{S_1 \to S_2}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall v_1.v_1 \in RED_{S_1}[\vec{\mathscr{R}}/\vec{X}] \to \forall E.\mathscr{C}_{RED_{S_2}[\vec{\mathscr{R}}/\vec{X}]}(E) \to \mathscr{N}(E[v_0\, v_1]) \\ v \in RED_{\forall c.S}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall U.\forall \mathscr{S}.\mathscr{S} \in \mathscr{RC}(U) \to \forall E.\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(E) \to \mathscr{N}(E[v\{U\}]) \end{array}$$

It is easy to see that $RED_S[\vec{\mathscr{R}}/\vec{X}] \in \mathscr{RC}(S\{\vec{T}/\vec{X}\})$. To prove the main result, we need a substitution lemma.

Lemma 2. *We have* $RED_{S\{T/c\}}[\vec{\mathscr{R}}/\vec{X}] = RED_S[\vec{\mathscr{R}}/\vec{X}, RED_T[\vec{\mathscr{R}}/\vec{X}]/c]$.

Proof. By induction on S. □

We are now ready to state the main lemma:

Lemma 3. *Let* t *be a plain term such that* $\Gamma;\Delta \vdash t : S$, $\Gamma = x_1 : T_1, \ldots, x_n : T_n$, *and* $\Delta = k_1 : \neg U_1, \ldots, k_m : \neg U_m$. *Let* $\{X_1, \ldots, X_p\} = ftv(S) \cup ftv(\Gamma) \cup ftv(\Delta)$. *Let* $\vec{V}$ *be a sequence of types of length* p, *and* $\vec{\mathscr{R}}$ *be reducibility candidates such that* $\mathscr{R}_i \in \mathscr{RC}(V_i)$ *for all* $i = 1, \ldots, p$. *Let* $\vec{v}$ *be closed values such that* $\cdot;\cdot \vdash v_i : T_i\{\vec{V}/\vec{X}\}$ *and* $v_i \in RED_{T_i}[\vec{\mathscr{R}}/\vec{X}]$ *for all* $i = 1, \ldots, n$. *Let* $\vec{E}$ *be closed contexts such that* $\cdot;\cdot \vdash E_i : \neg U_i$ *and* $\mathscr{C}_{RED_{U_i}[\vec{\mathscr{R}}/\vec{X}]}(E_i)$ *for all* $i = 1, \ldots, m$, *and let* E *be such that* $\cdot;\cdot \vdash E : \neg S$ *and* $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}]}(E)$. *Then* $\mathscr{N}(E[t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}])$ *holds.*

Proof. By induction on t.

- In the case $t = x_i$, we have $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = v_i$, as well as $T_i = S$. Because $v_i \in RED_{T_i}[\vec{\mathscr{R}}/\vec{X}]$, by definition of $\mathscr{C}_{RED_{T_i}[\vec{\mathscr{R}}/\vec{X}]}(E)$, we have the required result.
- In the case $t = \lambda x^{S_1}.s$, we have $S = S_1 \to S_2$. Let $s' = s\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$ and $S_1' = S_1\{\vec{V}/\vec{X}\}$; then $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = \lambda x^{S_1'}.s'$. We now prove that $\lambda x^{S_1'}.s' \in RED_S[\vec{\mathscr{R}}/\vec{X}]$; from that we can deduce the required result by the definition of $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}]}(E)$. Let v be such that $v \in RED_{S_1}[\vec{\mathscr{R}}/\vec{X}]$, and let E' be such that $\mathscr{C}_{RED_{S_2}[\vec{\mathscr{R}}/\vec{X}]}(E')$. We have $E'[(\lambda x^{S_1'}.s')\, v] \to_{\mathrm{v}} E'[s'\{v/x\}]$. By the induction hypothesis, we have $\mathscr{N}(E'[s'\{v/x\}])$, therefore $\mathscr{N}(E'[(\lambda x^{S_1'}.s')\, v])$ holds. Consequently, $\lambda x^{S_1'}.s' \in RED_S[\vec{\mathscr{R}}/\vec{X}]$ as required.
- In the case $t = t_0\, t_1$, we have $\Gamma;\Delta \vdash t_0 : S' \to S$ and $\Gamma;\Delta \vdash t_1 : S'$ for some S'. Let $t_0' = t_0\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$, $t_1' = t_1\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$; then $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = t_0'\, t_1'$. We then have $E[t_0'\, t_1'] = E\, t_1'[t_0']$, and to conclude, we would like to apply the induction hypothesis to t_0. To this

[2] Henceforth, for any metavariable m, we write $\vec{m}$ to range over sequences of entities denoted by m.

end, we have to prove that $\mathscr{C}_{RED_{S'\to S}[\vec{\mathscr{R}}/\vec{X}]}(E\,t_1')$ holds. Let v_0 be such that $v_0 \in RED_{S'\to S}[\vec{\mathscr{R}}/\vec{X}]$. We want to prove that $\mathscr{N}(E\,t_1'[v_0])$ holds, which is equivalent to proving that $\mathscr{N}((v_0\,E)[t_1'])$ holds. Again, we want to prove this fact by using the induction hypothesis on t_1, but to do this, we first have to prove that $\mathscr{C}_{RED_{S'}[\vec{\mathscr{R}}/\vec{X}]}(v_0\,E)$ holds. Let v_1 be such that $v_1 \in RED_{S'}[\vec{\mathscr{R}}/\vec{X}]$. Since we have $v_0 \in RED_{S'\to S}[\vec{\mathscr{R}}/\vec{X}]$ and $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}]}(E)$, therefore $\mathscr{N}(E[v_0\,v_1])$ holds, i.e.,we have $\mathscr{N}((v_0\,E)[v_1])$. Consequently, $\mathscr{C}_{RED_{S'}[\vec{\mathscr{R}}/\vec{X}]}(v_0\,E)$ holds. Therefore, we can use the induction hypothesis on t_1 to deduce that $\mathscr{N}((v_0\,E)[t_1'])$ holds. As a result, we therefore have $\mathscr{C}_{RED_{S'\to S}[\vec{\mathscr{R}}/\vec{X}]}(E\,t_1')$ and we can prove the required fact by using the induction hypothesis on t_0.

- In the case $t = \Lambda c.s$, $S = \forall c.S'$ for some S'. Let $s' = s\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$ and $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = \Lambda c.s'$. We now prove that $\Lambda c.s' \in RED_S[\vec{\mathscr{R}}/\vec{X}]$; the required result then holds by the definition of $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}]}(E)$. Let V' be a type and let $\mathscr{R}' \in \mathscr{R}\mathscr{C}(V')$. Let E' be such that $\mathscr{C}_{RED_{S'}[\vec{\mathscr{R}}/\vec{X},\mathscr{R}'/c]}(E')$ holds. We have $E'[(\Lambda c.s')\{V'\}] \to_{\mathrm{v}} E'[s'\{V'/c\}]$. By the induction hypothesis, we have that $\mathscr{N}(E'[s'\{V'/c\}])$ holds, therefore we obtain that $\mathscr{N}(E'[(\Lambda c.s')\{V'\}])$ holds as required.
- In the case $t = t_0\{V'\}$, we have $\Gamma;\Delta \vdash t_0 : \forall c.S'$ with $S = S'\{V'/c\}$ for some S'. Let $t_0' = t_0\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$ and $V'' = V'\{\vec{V}/\vec{X}\}$; we have $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = t_0'\{V''\}$, $E[t_0'\{V''\}] = E\,\{V''\}[t_0']$. To conclude, we want to apply the induction hypothesis to t_0, but first we have to prove that $\mathscr{C}_{RED_{\forall c.S'}[\vec{\mathscr{R}}/\vec{X}]}(E\,\{V''\})$ holds. Let v be such that $v \in RED_{\forall c.S'}[\vec{\mathscr{R}}/\vec{X}]$. By Lemma 2, $RED_{S'\{V'/c\}}[\vec{\mathscr{R}}/\vec{X}] = RED_{S'}[\vec{\mathscr{R}}/\vec{X}, RED_{V'}[\vec{\mathscr{R}}/\vec{X}]/c]$ and hence $\mathscr{C}_{RED_{S'}[\vec{\mathscr{R}}/\vec{X}, RED_{V'}[\vec{\mathscr{R}}/\vec{X}]/c]}(E)$ holds. Besides, we have that $RED_{V'}[\vec{\mathscr{R}}/\vec{X}] \in \mathscr{R}\mathscr{C}(V'')$ holds, so by the definition of $RED_{\forall c.S'}[\vec{\mathscr{R}}/\vec{X}]$ we obtain $\mathscr{N}(E[v\{V''\}])$, i.e.,$\mathscr{N}(E\,\{V''\}[v])$ holds. Therefore, $\mathscr{C}_{RED_{\forall c.S'}[\vec{\mathscr{R}}/\vec{X}]}(E\,\{V''\})$ holds, hence we have the required result by using the induction hypothesis on s.
- Suppose $t = \mathscr{K}k.s$. Let $s' = s\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$. We then have the equality $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = \mathscr{K}k.s'$. Since $E[\mathscr{K}k.s'] \to_{\mathrm{v}} E[s'\{\ulcorner E\urcorner/k\}]$, we obtain $\mathscr{N}(E[s'\{\ulcorner E\urcorner/k\}])$ by the induction hypothesis, and therefore $\mathscr{N}(E[\mathscr{K}k.s'])$ holds.
- Suppose $t = k_i \hookleftarrow s$. Let $s' = s\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\}$. We then have the equality $t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \ulcorner\vec{E}\urcorner/\vec{k}\} = \ulcorner E_i\urcorner \hookleftarrow s'$. The program $E[\ulcorner E_i\urcorner \hookleftarrow s']$ is equivalent to $(\ulcorner E_i\urcorner \hookleftarrow E)[s']$, so we need to prove that $\mathscr{N}((\ulcorner E_i\urcorner \hookleftarrow E)[s'])$ holds, applying the induction hypothesis to s. To this end, we first prove that $\mathscr{C}_{RED_{U_i}[\vec{\mathscr{R}}/\vec{X}]}(\ulcorner E_i\urcorner \hookleftarrow E)$ holds. Let v be such that $v \in RED_{U_i}[\vec{\mathscr{R}}/\vec{X}]$. The program $(\ulcorner E_i\urcorner \hookleftarrow E)[v]$ is equivalent to $E[\ulcorner E_i\urcorner \hookleftarrow v]$. We have $E[\ulcorner E_i\urcorner \hookleftarrow v] \to_{\mathrm{v}} E_i[v]$, and since $\mathscr{C}_{RED_{U_i}[\vec{\mathscr{R}}/\vec{X}]}(E_i)$ holds, we obtain $\mathscr{N}(E_i[v])$. Consequently, $\mathscr{N}((\ulcorner E_i\urcorner \hookleftarrow E)[v])$ holds.

□

Theorem 1. *If p is a well-typed program, then $\mathscr{N}(p)$ holds.*

Proof. We have $\mathscr{C}_{\mathscr{R}}([\,])$ for any $\mathscr{R}$, therefore we can use the previous lemma. □

The proof of Theorem 1 is constructive and its computational content is a call-by-value evaluator for plain terms in the continuation-passing style that is an instance of normalization by evaluation [3, 4].

2.4 Call by name

The proof method can be adapted to the call-by-name strategy, again by using a corresponding continuation-passing style interpretation of terms [16, 18, 26]. In this case, the syntax of reduction contexts

becomes:

$$\text{CBN contexts:}\quad E \;::=\; [\,] \mid E\,t \mid E\,\{S\}$$

and the reduction rules are modified in that a lambda abstraction and a throwing operation can be applied to an arbitrary term instead of only to a value, in the rules (β_n) and $(throw_n)$ below:

$$\begin{array}{rcll} E[(\lambda x^S.t_0)\,t_1] & \to_{\mathrm{n}} & E[t_0\{t_1/x\}] & (\beta_n) \\ E[(\Lambda X.t)\{S\}] & \to_{\mathrm{n}} & E[t\{S/X\}] & (\beta_T) \\ E[\mathscr{K}\,k.t] & \to_{\mathrm{n}} & E[t\{\ulcorner E\urcorner/k\}] & (callcc) \\ E_1[\ulcorner E_0\urcorner \hookleftarrow t] & \to_{\mathrm{n}} & E_0[t] & (throw_n) \end{array}$$

The type system is as before, except there are fewer rules for typing contexts. In the remainder of this section we briefly point out the main differences in the proof of termination of evaluation between the call-by-value and the call-by-name strategies.

A reducibility candidate of type S in call by name is a set of values of type S (where values are as in call by value), and the definition of the associated predicate $\mathscr{C}_{\mathscr{R}}$ is the same as in call by value, except that the predicate $\mathscr{N}(\cdot)$ is defined using the call-by-name reduction relation $\to_{\mathrm{n}}$.

However, parametric reducibility candidates are defined in a substantially different way, reflecting the call-by-name strategy. Let S be a type, $ftv(S) \subseteq \vec{X}$, $\vec{T}$ be a sequence of types of the same size as $\vec{X}$, and $\vec{\mathscr{R}}$ be reducibility candidates such that each $\mathscr{R}_i$ is of type T_i. We define the parametric reducibility candidate $RED_S[\vec{\mathscr{R}}/\vec{X}]$ of type $S\{\vec{T}/\vec{X}\}$ as follows:

$$\begin{array}{rcl} v \in RED_{X_i}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & v \in \mathscr{R}_i \\ v \in RED_{S_1\to S_2}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall t.\, \mathscr{Q}_{RED_{S_1}[\vec{\mathscr{R}}/\vec{X}]}(t) \to \mathscr{Q}_{RED_{S_2}[\vec{\mathscr{R}}/\vec{X}]}(v\,t) \\ v \in RED_{\forall c.S}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall U.\forall \mathscr{S}.\mathscr{S} \in \mathscr{RC}(U) \to \forall E.\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(E) \to \mathscr{N}(E[v\{U\}]) \end{array}$$

with

$$\mathscr{Q}_{\mathscr{R}}(t) = \forall E.\mathscr{C}_{\mathscr{R}}(E) \to \mathscr{N}(E[t])$$

The predicate $\mathscr{Q}$ used in the second clause of this definition reflects the fact that a term given as argument to a function is not yet a value (whose reducibility is immediate), but it can be seen as a delayed computation that may be forced later, by putting it in a context.

The main lemma is now formulated as follows:

Lemma 4. *Let t be a plain term such that $\Gamma;\Delta \vdash t : S$, $\Gamma = x_1:T_1, \ldots, x_n : T_n$, and $\Delta = k_1:\neg U_1, \ldots, k_m : \neg U_m$. Let $\{X_1,\ldots,X_p\} = ftv(S) \cup ftv(\Gamma) \cup ftv(\Delta)$. Let $\vec{V}$ be a sequence of types of length p, and let $\vec{\mathscr{R}}$ be reducibility candidates such that $\mathscr{R}_i \in \mathscr{RC}(V_i)$ for all $i = 1,\ldots,p$. Let $\vec{t}$ be closed terms such that $\cdot;\cdot \vdash t_i : T_i\{\vec{V}/\vec{X}\}$ and $\mathscr{Q}_{RED_{T_i}[\vec{\mathscr{R}}/\vec{X}]}(t_i)$ for all $i = 1,\ldots,n$. Next, let $\vec{E}$ be closed contexts such that $\cdot;\cdot \vdash E_i : \neg U_i$ and $\mathscr{C}_{RED_{U_i}[\vec{\mathscr{R}}/\vec{X}]}(E_i)$ for all $i = 1,\ldots,m$, and let E be such that $\cdot;\cdot \vdash E : \neg S$ and $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}]}(E)$. Then $\mathscr{N}(E[t\{\vec{V}/\vec{X},\vec{t}/\vec{x},\vec{E}/\vec{k}\}])$ holds.*

The termination of call-by-name evaluation for well-typed programs follows from Lemma 4. As before, the computational content of the proof takes the form of an evaluator, only this time in the call-by-name continuation-passing style.

In [25], Parigot gives a proof of strong normalization for the second-order (i.e., with the types of System F) $\lambda\mu$-calculus using a variant of Girard's method of reducibility candidates. Parigot's proof and ours share some similarities, even though the results are quite different in nature (strong normalization

vs. termination of a particular strategy) and Parigot's proof is more general in that it can be applied to the implicitly typed as well as to the explicitly typed language. The key point in Parigot's proof is the following reducibility candidates characterization result: for all reducibility candidates $\mathscr{R}$, there exists a set $\mathscr{S}$ of (possibly empty) finite sequences of strongly normalizing terms such that we have $t \in \mathscr{R}$ iff for all $\vec{s} \in \mathscr{S}$, $t\,\vec{s}$ is strongly normalizing. The greatest such set $\mathscr{S}$ is denoted by $\mathscr{R}^{\perp}$. The characterization result is then used to prove a lemma similar to Lemma 3.

The finite sequences of terms can be seen as (call-by-name) contexts in our setting. Moreover, we notice that $\vec{s} \in \mathscr{R}^{\perp}$ iff for all $t \in \mathscr{R}$, $t\,\vec{s}$ is strongly normalizing; this resembles the definition of the reducibility predicates on contexts $\mathscr{C}_{\mathscr{R}}$ in our proof. However, the terms in a sequence $\vec{s}$ have to be strongly normalizing in Parigot's proof, while we do not have a similar requirement on contexts. This fact will have consequences for program extraction; the program extracted from our proof would be an evaluator in CPS style where contexts (continuations) are passed around without being deconstructed (as in [3]).

3 System F with delimited control operators

In this section, we prove termination of the call-by-value evaluation in an extension of the explicitly typed System F with the delimited control operators *shift* and *reset* of Danvy and Filinski [12]. While the abortive control operators such as *callcc* model jumps, *shift* and *reset* allow for delimited-control capture and continuation composition.

3.1 Syntax and semantics

We extend the explicitly typed System F with the operators *shift* $\mathscr{S}$, *reset* $\langle\cdot\rangle$, and *throw* $\hookleftarrow$. We call the language $\lambda^{F}_{\mathscr{S},v}$. The syntax of terms, term types, contexts, and metacontexts of $\lambda^{F}_{\mathscr{S},v}$ is given as follows:

$$\begin{array}{rrcl}
\text{Terms:} & t & ::= & x \mid \lambda x^{S}.t \mid t\,t \mid \Lambda X.t \mid t\{S\} \mid \mathscr{S}k.t \mid \langle t\rangle \mid k \hookleftarrow t \mid \ulcorner E \urcorner \hookleftarrow t \\
\text{Term types:} & S & ::= & X \mid S_S \to_S S \mid \forall X.S^{S,S} \\
\text{CBV contexts:} & E & ::= & [\,] \mid (\lambda x^{S}.t)\,E \mid E\,t \mid E\,\{S\} \mid \ulcorner E \urcorner \hookleftarrow E \\
\text{Metacontexts:} & F & ::= & \bullet \mid E \# F
\end{array}$$

The new term constructs are the *shift* operator $\mathscr{S}k.t$ binding the continuation variable k in t, and a term delimited by *reset*, denoted $\langle t\rangle$. The remaining term constructs are as before. The (non-standard) syntax of types is discussed in Section 3.2. The syntax of reduction contexts is the same as in Section 2.1, and terms are plugged in contexts using a function *plug*, defined in a similar manner as before.

The new syntactic category is that of metacontexts. A metacontext can be understood as a stack of contexts: $\bullet$ is the empty metacontext and the metacontext $E\#F$ is obtained by pushing the context E on top of F with each context in the stack separated from the rest by a delimiter. The meaning of metacontexts is formalized through a function $plug_m$, defined below:

$$\begin{array}{rcl}
plug_m\,(t,\bullet) & = & t \\
plug_m\,(t,E\#F) & = & plug_m\,(\langle plug\,(t,E)\rangle,F)
\end{array}$$

The result of $plug_m\,(t,F)$ is denoted by $F[t]$.

Programs are closed plain terms delimited by a *reset*. A program p is subject to decompositions into a term t, a context E and a metacontext F such that $p = F[\langle E[t]\rangle]$.

The call-by-value reduction relation on programs in $\lambda^{F}_{\mathscr{S},v}$ is defined by the following rules:

$$\begin{array}{rcll}
F[\langle E[(\lambda x^S.t)\, v]\rangle] & \to_{\mathrm{v}} & F[\langle E[t\{v/x\}]\rangle] & (\beta_v) \\
F[\langle E[(\Lambda X.t)\{S\}]\rangle] & \to_{\mathrm{v}} & F[\langle E[t\{S/X\}]\rangle] & (\beta_T) \\
F[\langle E[\mathscr{S}k.t]\rangle] & \to_{\mathrm{v}} & F[\langle t\{\ulcorner E\urcorner/k\}\rangle] & (\mathit{shift}) \\
F[\langle E[\ulcorner E'\urcorner \hookleftarrow v]\rangle] & \to_{\mathrm{v}} & F\,\#\,E[\langle E'[v]\rangle] & (\mathit{throw}_v) \\
F[\langle E[\langle v\rangle]\rangle] & \to_{\mathrm{v}} & F[\langle E[v]\rangle] & (\mathit{reset})
\end{array}$$

where values are defined as before. The first two reduction rules are standard (and insensitive to the surrounding context and metacontext). The rule (shift) states that reducing $\mathscr{S}k.t$ consists in capturing the context E and substituting it for the continuation variable k in the body t (the current context is then set to be empty). When a captured context E' is applied to a value (in the rule (throw_v)), it is reinstated as the current context, and the then-current context E is pushed on the metacontext F. Finally, the last rule states that when a value is enclosed in a *reset*, it means that the *reset* can be discarded since no further captures can occur inside it.

The evaluation relation $\to^*_{\mathrm{v}}$ is defined as before, where the expected result of evaluation is a program value of the form $\langle v\rangle$.

3.2 Type system

We add System F types to the type system of Biernacka and Biernacki [4], which is a slight modification of the classical Danvy and Filinski's type system for *shift* and *reset* [11]. The type system is presented in Figure 2. In a type $\forall X.S^{T,U}$, the quantifier binds the occurrences of X in S, T, and U. We define the set of free type variables $\mathit{ftv}(S)$ of a term type S accordingly, and we define $\mathit{ftv}(S \rhd T) := \mathit{ftv}(S) \cup \mathit{ftv}(T)$.

In this system, contexts are assigned types of the form $S \rhd T$, where S is the type of the hole and T is the answer type, and metacontexts are assigned types of the form $\neg S$, where S is the type of the hole. A typing judgment $\Gamma;\Delta \,|\, T \vdash t : S \,|\, U$ roughly means that under the assumptions Γ and Δ, the term t can be plugged into a context of type $S \rhd T$ and a metacontext of type $\neg U$ (in general, the evaluation of t may use the surrounding context of type $S \rhd T$ to produce a value of type U, with $T \neq U$). Because both abstractions $\lambda x^S.t$ and $\Lambda X.t$ denote "frozen" computations—waiting for a term and a type, resp., to activate them—the arrow type and the $\forall$-type contain additional type annotations. Roughly, the type $S_U \to_V T$ is assigned to a function that can be applied to an argument of type S within a context of type $T \rhd U$ and a metacontext of type $\neg V$. Similarly, the type $\forall X.S^{T,U}$ is assigned to a term that can be applied to a type V within a context of type $S\{V/X\} \rhd T\{V/X\}$ and a metacontext of type $\neg U\{V/X\}$. It can be shown that closed well-typed terms either are values or decompose uniquely into a redex, a context and a metacontext, and that the reduction rules preserve types.

The type system of Figure 2 is more liberal than the one defined for $\lambda_2^{s/r,Std}$, a language defined by Asai and Kameyama in [2], which is similar to $\lambda^{F}_{\mathscr{S},v}$. In [2], polymorphic abstraction types do not contain any additional type annotations, and can only be assigned to abstractions $\Lambda X.t$ where t is a *pure* term, i.e., a term such that $\Gamma;\Delta \,|\, T \vdash t : S \,|\, T$ is derivable for any T. Pure terms are terms free from control effects, such as x, $\lambda x^S.t$, $\langle t\rangle$, or $\Lambda X.t$. In contrast, we allow arbitrary abstractions of the form $\Lambda X.t$, at the cost of additional type annotations in the polymorphic abstraction types. As pointed out by Asai and Kameyama, restricting $\forall$-introduction to pure terms is not mandatory in a calculus with standard call-by-value evaluation, such as $\lambda_2^{s/r,Std}$ and our calculus. However, such restriction becomes necessary

Typing terms ($S ::= X \mid S_U \to_V T \mid \forall X.S^{T,U}$):

$$\frac{}{\Gamma, x : S; \Delta \,|\, T \vdash x : S \,|\, T} \qquad \frac{\Gamma, x : S; \Delta \,|\, U \vdash t : T \,|\, V}{\Gamma; \Delta \,|\, W \vdash \lambda x^S.t : S_U \to_V T \,|\, W}$$

$$\frac{\Gamma; \Delta \,|\, X \vdash t_0 : S_U \to_W T \,|\, V \qquad \Gamma; \Delta \,|\, W \vdash t_1 : S \,|\, X}{\Gamma; \Delta \,|\, U \vdash t_0\, t_1 : T \,|\, V} \qquad \frac{\Gamma; \Delta \,|\, T \vdash t : S \,|\, U \qquad X \notin ftv(\Gamma) \cup ftv(\Delta)}{\Gamma; \Delta \,|\, V \vdash \Lambda X.t : \forall X.S^{T,U} \,|\, V}$$

$$\frac{\Gamma; \Delta \,|\, U\{V/X\} \vdash t : \forall X.S^{T,U} \,|\, W}{\Gamma; \Delta \,|\, T\{V/X\} \vdash t\{V\} : S\{V/X\} \,|\, W} \qquad \frac{\Gamma; \Delta \,|\, U \vdash t : U \,|\, S}{\Gamma; \Delta \,|\, T \vdash \langle t \rangle : S \,|\, T} \qquad \frac{\Gamma; \Delta, k : S \rhd T \,|\, V \vdash t : V \,|\, U}{\Gamma; \Delta \,|\, T \vdash \mathscr{S} k.t : S \,|\, U}$$

$$\frac{\Gamma; \Delta, k : S \rhd T \,|\, U \vdash t : S \,|\, V}{\Gamma; \Delta, k : S \rhd T \,|\, U \vdash k \hookleftarrow t : T \,|\, V} \qquad \frac{\Gamma; \Delta \vdash E : S \rhd T \qquad \Gamma; \Delta \,|\, U \vdash t : S \,|\, V}{\Gamma; \Delta \,|\, U \vdash \ulcorner E \urcorner \hookleftarrow t : T \,|\, V}$$

Typing contexts ($C ::= S \rhd T$):

$$\frac{}{\Gamma; \Delta \vdash [\,] : S \rhd S} \qquad \frac{\Gamma; \Delta \vdash E : T \rhd U \qquad \Gamma; \Delta \,|\, V \vdash t : S \,|\, W}{\Gamma; \Delta \vdash E\, t : (S_U \to_V T) \rhd W}$$

$$\frac{\Gamma; \Delta \,|\, W \vdash \lambda x^S.t : S_U \to_V T \,|\, W \qquad \Gamma; \Delta \vdash E : T \rhd U}{\Gamma; \Delta \vdash (\lambda x^S.t)\, E : S \rhd V} \qquad \frac{\Gamma; \Delta \vdash E : S\{V/X\} \rhd T\{V/X\}}{\Gamma; \Delta \vdash E\,\{V\} : \forall X.S^{T,U} \rhd U\{V/X\}}$$

$$\frac{\Gamma; \Delta \vdash E' : S \rhd T \qquad \Gamma; \Delta \vdash E : T \rhd U}{\Gamma; \Delta \vdash \ulcorner E' \urcorner \hookleftarrow E : S \rhd U}$$

Typing metacontexts ($D ::= \neg S$):

$$\frac{}{\Gamma; \Delta \vdash \bullet : \neg S} \qquad \frac{\Gamma; \Delta \vdash E : S \rhd T \qquad \Gamma; \Delta \vdash F : \neg T}{\Gamma; \Delta \vdash E \# F : \neg S}$$

Figure 2: Typing rules for System F with delimited control operators under call by value

for the calculus $\lambda_2^{s/r,ML}$ of [2] with ML-like call-by-value evaluation (where reduction is allowed under Λ) for subject reduction to hold.

3.3 Termination

The proof of termination is very similar to that of Section 2.3, and here we only point out the main differences. This time our development is based on the layered continuation-passing style for *shift* and *reset*, where terms are passed two layers of continuations [12].

We define the normalization predicate $\mathscr{N}(p)$ as follows:

$$\mathscr{N}(p) := \exists v.\, p \to_{\mathrm{v}}^* \langle v \rangle$$

A *reducibility candidate* $\mathscr{R}$ of type S is a set of closed values of type S. We write $\mathscr{RC}(S)$ for the set of reducibility candidates of type S. Let $\mathscr{R}$, $\mathscr{S}$ be reducibility candidates of types S and T, respectively. We define the predicate $\mathscr{C}_{\mathscr{R},\mathscr{S}}(E)$ on closed contexts of type $S \rhd T$ and the predicate $\mathscr{M}_{\mathscr{R}}(F)$ on closed

metacontexts of type $\neg S$ as follows:

$$\begin{array}{rcl} \mathscr{C}_{\mathscr{R},\mathscr{S}}(E) & := & \forall v.\, v \in \mathscr{R} \to \forall F.\, \mathscr{M}_{\mathscr{S}}(F) \to \mathscr{N}(F[\langle E[v]\rangle]) \\ \mathscr{M}_{\mathscr{R}}(F) & := & \forall v.\, v \in \mathscr{R} \to \mathscr{N}(F[\langle v\rangle]) \end{array}$$

Let S be a type, $ftv(S) \subseteq \vec{X}$, $\vec{T}$ be a sequence of types of the same size as $\vec{X}$, and $\vec{\mathscr{R}}$ be reducibility candidates such that each $\mathscr{R}_i$ is of type T_i. We now define the parametric reducibility candidate $RED_S[\vec{\mathscr{R}}/\vec{X}]$ of type $S\{\vec{T}/\vec{X}\}$ as follows:

$$\begin{array}{rcl} v \in RED_{X_i}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & v \in \mathscr{R}_i \\ v_0 \in RED_{S_U \to_V T}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall v_1.v_1 \in RED_S[\vec{\mathscr{R}}/\vec{X}] \to \forall E.\mathscr{C}_{RED_T[\vec{\mathscr{R}}/\vec{X}],RED_U[\vec{\mathscr{R}}/\vec{X}]}(E) \to \\ & & \quad \forall F.\mathscr{M}_{RED_V[\vec{\mathscr{R}}/\vec{X}]}(F) \to \mathscr{N}(F[\langle E[v_0\, v_1]\rangle]) \\ v \in RED_{\forall c.S^{T,U}}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall V.\forall \mathscr{S}.\mathscr{S} \in \mathscr{RC}(V) \to \forall E.\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c],RED_T[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(E) \to \\ & & \quad \forall F.\mathscr{M}_{RED_U[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(F) \to \mathscr{N}(F[\langle E[v\{V\}]\rangle]) \end{array}$$

Using a substitution lemma similar to Lemma 2, we can prove the following result, from which the termination theorem follows for closed plain terms:

Lemma 5. *Let t be a plain term such that $\Gamma;\Delta \,|\, T \vdash t : S \,|\, U$, $\Gamma = x_1 : T_1, \ldots, x_n : T_n$, $\Delta = k_1 : C_1, \ldots, k_m : C_m$. Let $\{X_1, \ldots, X_p\} = ftv(S) \cup ftv(T) \cup ftv(U) \cup ftv(\Gamma) \cup ftv(\Delta)$. Let $\vec{V}$ be a sequence of types of length p, and let $\vec{\mathscr{R}}$ be reducibility candidates such that $\mathscr{R}_i \in \mathscr{RC}(V_i)$. Let $\vec{v}$ be closed values such that $\cdot;\cdot \,|\, W \vdash v_i : T_i\{\vec{V}/\vec{X}\} \,|\, W$ and $v_i \in RED_{T_i}[\vec{\mathscr{R}}/\vec{X}]$ for each i. Let $\vec{E}$ be closed contexts such that $\cdot;\cdot \vdash E_i : C_i$, $C_i = W_i^1 \rhd W_i^2$, and $\mathscr{C}_{RED_{W_i^1}[\vec{\mathscr{R}}/\vec{X}],RED_{W_i^2}[\vec{\mathscr{R}}/\vec{X}]}(E_i)$ for each i. Let E be such that $\cdot;\cdot \vdash E : S \rhd T$ and $\mathscr{C}_{RED_S[\vec{\mathscr{R}}/\vec{X}],RED_T[\vec{\mathscr{R}}/\vec{X}]}(E)$. Let F be such that $\cdot;\cdot \vdash F : \neg U$ and $\mathscr{M}_{RED_U[\vec{\mathscr{R}}/\vec{X}]}(F)$. Then $\mathscr{N}(F[\langle E[t\{\vec{V}/\vec{X}, \vec{v}/\vec{x}, \vec{E}/\vec{k}\}]\rangle])$ holds.*

Theorem 2. *If p is a well-typed program, then $\mathscr{N}(p)$ holds.*

Proof. We have $\mathscr{C}_{\mathscr{R},\mathscr{R}}([\,])$ for any $\mathscr{R}$ and $\mathscr{M}_{\mathscr{R}}(\bullet)$ for any $\mathscr{R}$, so we can use the previous lemma. □

Again, the computational content of this proof takes the form of an evaluator in CPS, this time with two layers of continuations [4].

3.4 Call by name

The developments of the previous section can be easily adapted to the call-by-name strategy. Following [4], we modify the syntax of types to express the fact that functions accept not values, but suspended computations expecting a continuation:

$$\begin{array}{rcl} \text{Terms:} \quad t & ::= & x \mid \lambda x^{S^{S,S}}.t \mid \ldots \\ \text{Term types:} \quad S & ::= & X \mid S^{S,S}{}_S \to_S S \mid \forall X.S^{S,S} \end{array}$$

The syntax of reduction contexts is modified:

$$\text{CBN contexts:} \quad E ::= [\,] \mid E\,t \mid E\,\{S\}$$

The reduction rules are modified as in Section 2.4:

$$\begin{array}{rcll} F[\langle E[(\lambda x^{S^{T,U}}.t_0)\,t_1]\rangle] & \to_{\mathrm{n}} & F[\langle E[t_0\{t_1/x\}]\rangle] & (\beta_n) \\ F[\langle E[(\Lambda X.t)\{S\}]\rangle] & \to_{\mathrm{n}} & F[\langle E[t\{S/X\}]\rangle] & (\beta_T) \\ F[\langle E[\mathscr{S}k.t]\rangle] & \to_{\mathrm{n}} & F[\langle t\{\ulcorner E\urcorner/k\}\rangle] & (\mathit{shift}) \\ F[\langle E[\ulcorner E'\urcorner \hookleftarrow t]\rangle] & \to_{\mathrm{n}} & F\,\#\,E[\langle E'[t]\rangle] & (\mathit{throw}_n) \\ F[\langle E[\langle v\rangle]\rangle] & \to_{\mathrm{n}} & F[\langle E[v]\rangle] & (\mathit{reset}) \end{array}$$

The typing rules for abstractions, function applications and throwing are changed, as in [4]. The modified rules for abstractions and function applications are as follows:

$$\frac{\Gamma, x : S^{T,U};\Delta \,|\, W \vdash t : V \,|\, X}{\Gamma;\Delta \,|\, Y \vdash \lambda x^{S^{T,U}}.t : S^{T,U}{}_W \to_X V \,|\, Y} \qquad \frac{\Gamma;\Delta \,|\, X \vdash t_0 : S^{T,U}{}_W \to_X V \,|\, Y \qquad \Gamma;\Delta \,|\, T \vdash t_1 : S \,|\, U}{\Gamma;\Delta \,|\, W \vdash t_0\,t_1 : V \,|\, Y}$$

We also modify the rules for typing contexts in a straightforward way.

We define reducibility candidates and the predicates $\mathscr{C}_{\mathscr{R},\mathscr{S}}$ and $\mathscr{M}_{\mathscr{T}}$ as in call by value, the predicate $\mathscr{N}(\cdot)$ is defined using $\to_{\mathrm{n}}$, the call by name reduction relation. The case for function types in the definition of the parametric reducibility candidates is as in Section 2.4. The definition of the $\mathscr{Q}^{\mathscr{S},\mathscr{T}}_{\mathscr{R}}$ is adapted to the language with *shift* and *reset*.

$$\begin{array}{rcl} v \in \mathit{RED}_{X_i}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & v \in \mathscr{R}_i \\ v \in \mathit{RED}_{S^{T,U}{}_W \to_X V}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall t.\mathscr{Q}^{\mathit{RED}_T[\vec{\mathscr{R}}/\vec{X}],\mathit{RED}_U[\vec{\mathscr{R}}/\vec{X}]}_{\mathit{RED}_S[\vec{\mathscr{R}}/\vec{X}]}(t) \to \mathscr{Q}^{\mathit{RED}_W[\vec{\mathscr{R}}/\vec{X}],\mathit{RED}_X[\vec{\mathscr{R}}/\vec{X}]}_{\mathit{RED}_V[\vec{\mathscr{R}}/\vec{X}]}(v\,t) \\ v \in \mathit{RED}_{\forall c.S^{T,U}}[\vec{\mathscr{R}}/\vec{X}] & \text{iff} & \forall V.\forall \mathscr{S}.\mathscr{S} \in \mathscr{RC}(V) \to \forall E.\mathscr{C}_{\mathit{RED}_S[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c],\mathit{RED}_T[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(E) \to \\ & & \quad \forall F.\mathscr{M}_{\mathit{RED}_U[\vec{\mathscr{R}}/\vec{X},\mathscr{S}/c]}(F) \to \mathscr{N}(F[\langle E[v\{V\}]\rangle]) \end{array}$$

with

$$\mathscr{Q}^{\mathscr{S},\mathscr{T}}_{\mathscr{R}}(t) = \forall E.\mathscr{C}_{\mathscr{R},\mathscr{S}}(E) \to \forall F.\mathscr{M}_{\mathscr{T}}(F) \to \mathscr{N}(F[\langle E[t]\rangle])$$

The main lemma is as follows:

Lemma 6. *Let t be a plain term such that $\Gamma;\Delta \,|\, T \vdash t : S \,|\, U$, $\Gamma = x_1 : S_1^{T_1,U_1}, \ldots, x_n : S_n^{T_n,U_n}$, $\Delta = k_1 : C_1, \ldots, k_m : C_m$. Let $\{X_1,\ldots,X_p\} = \mathit{ftv}(S) \cup \mathit{ftv}(T) \cup \mathit{ftv}(U) \cup \mathit{ftv}(\Gamma) \cup \mathit{ftv}(\Delta)$. Let $\vec{V}$ be a sequence of types of length p, and let $\vec{\mathscr{R}}$ be reducibility candidates such that $\mathscr{R}_i \in \mathscr{RC}(V_i)$. Let $\vec{r}$ be closed terms such that $\cdot;\cdot \,|\, T_i\{\vec{V}/\vec{X}\} \vdash r_i : S_i\{\vec{V}/\vec{X}\} \,|\, U_i\{\vec{V}/\vec{X}\}$ and $\mathscr{Q}^{\mathit{RED}_{T_i}[\vec{\mathscr{R}}/\vec{X}],\mathit{RED}_{U_i}[\vec{\mathscr{R}}/\vec{X}]}_{\mathit{RED}_{S_i}[\vec{\mathscr{R}}/\vec{X}]}(r_i)$ for each i. Let $\vec{E}$ be closed contexts such that $\cdot;\cdot \vdash E_i : C_i$, $C_i = W_i^1 \rhd W_i^2$, and $\mathscr{C}_{\mathit{RED}_{W_i^1}[\vec{\mathscr{R}}/\vec{X}],\mathit{RED}_{W_i^2}[\vec{\mathscr{R}}/\vec{X}]}(E_i)$ for each i. Let E be such that $\cdot;\cdot \vdash E : S \rhd T$ and $\mathscr{C}_{\mathit{RED}_S[\vec{\mathscr{R}}/\vec{X}],\mathit{RED}_T[\vec{\mathscr{R}}/\vec{X}]}(E)$. Let F be such that $\cdot;\cdot \vdash F : \neg U$ and $\mathscr{M}_{\mathit{RED}_U[\vec{\mathscr{R}}/\vec{X}]}(F)$. Then $\mathscr{N}(F[\langle E[t\{\vec{V}/\vec{X},\vec{r}/\vec{x},\vec{E}/\vec{k}\}]\rangle])$ holds.*

4 Conclusion and perspectives

We have shown that the context-based proof method developed by the first two authors for the simply-typed lambda calculus with control operators, be they abortive or delimited, scales to much more expressive type systems based on System F. The presented proofs are rather simple and elegant, and they do not require a journey through an optimized CPS translation in order to show termination of evaluation

for such calculi. Furthermore, if formalized in a logical framework equipped with program extraction mechanism, they can lead to executable specifications of programming languages with control operators and polymorphism—which is left as future work.

The proof method we have proposed is tailored towards characterization of termination in a wide range of context-sensitive reduction semantics that account for arbitrary reduction strategies or advanced control operators. For example, the proof of termination for delimited-control operators of Section 3 can be straightforwardly generalized to a polymorphic version of the CPS hierarchy of Danvy and Filinski [6, 12]. Similarly, we expect that one could consider a call-by-need version of System F, e.g., based on [13] and readily apply the context-based method to it. Such results do not seem to be immediately obtainable in other frameworks, e.g., using orthogonality techniques.

Acknowledgments: We thank the anonymous reviewers for detailed and insightful comments on several versions of this article. This work has been partially supported by Polish NCN grant number DEC-011/03/B/ST6/00348.

References

[1] Z. M. Ariola, H. Herbelin, and A. Sabry. A proof-theoretic foundation of abortive continuations. *Higher-Order and Symbolic Computation*, 20(4):403–429, 2007. DOI: 10.1007/s10990-007-9007-z.

[2] K. Asai and Y. Kameyama. Polymorphic delimited continuations. In Z. Shao, editor, *Proceedings of the Fifth Asian Symposium on Programming Languages and Systems, APLAS'07*, number 4807 in LNCS, pages 239–254, Singapore, Dec. 2007. DOI: 10.1007/978-3-540-76637-7_16.

[3] M. Biernacka and D. Biernacki. A context-based approach to proving termination of evaluation. In *Proceedings of the 25th Annual Conference on Mathematical Foundations of Programming Semantics*, Oxford, UK, Apr. 2009. DOI: 10.1016/j.entcs.2009.07.090.

[4] M. Biernacka and D. Biernacki. Context-based proofs of termination for typed delimited-control operators. In F. J. López-Fraguas, editor, *Proceedings of the 11th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'09)*, Coimbra, Portugal, Sept. 2009. DOI: 10.1145/1599410.1599446.

[5] M. Biernacka, D. Biernacki, and O. Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science*, 1(2:5):1–39, Nov. 2005. DOI: 10.2168/LMCS-1(2:5)2005.

[6] M. Biernacka, D. Biernacki, and S. Lenglet. Typing control operators in the CPS hierarchy. In M. Hanus, editor, *Proceedings of the 13th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'11)*, Odense, Denmark, July 2011. DOI: 10.1145/2003476.2003498.

[7] P.-L. Curien and H. Herbelin. The duality of computation. In P. Wadler, editor, *Proceedings of the 2000 ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, SIGPLAN Notices, Vol. 35, No. 9, pages 233–243, Montréal, Canada, Sept. 2000. ACM Press. DOI: 10.1145/351240.351262.

[8] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *Journal of Symbolic Logic*, 62(3):755–807, 1997. DOI: 10.2307/2275572.

[9] V. Danos and J.-L. Krivine. Disjunctive tautologies as synchronisation schemes. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic, 14th Annual Conference of the EACSL, Fischbachau, Germany, August 21-26, 2000, Proceedings*, volume 1862 of *Lecture Notes in Computer Science*, pages 292–301. Springer, 2000. DOI: 10.1007/3-540-44622-2_19.

[10] O. Danvy. Defunctionalized interpreters for programming languages. In P. Thiemann, editor, *Proceedings of the 2008 ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, SIGPLAN Notices, Vol. 43, No. 9, Victoria, British Columbia, Sept. 2008. DOI: 10.1145/1411203.1411206.

[11] O. Danvy and A. Filinski. A functional abstraction of typed contexts. DIKU Rapport 89/12, DIKU, Computer Science Department, University of Copenhagen, Copenhagen, Denmark, July 1989.

[12] O. Danvy and A. Filinski. Abstracting control. In M. Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. DOI: 10.1145/91556.91622.

[13] O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defunctionalized interpreters for call-by-need evaluation. In M. Blume and G. Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, number 6009 in LNCS, pages 240–256, Sendai, Japan, Apr. 2010. DOI: 10.1007/978-3-642-12251-4_18.

[14] M. Felleisen and D. P. Friedman. Control operators, the SECD machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.

[15] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[16] T. G. Griffin. A formulae-as-types notion of control. In P. Hudak, editor, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, Jan. 1990. DOI: 10.1145/96709.96714.

[17] R. Harper, B. F. Duba, and D. MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3(4):465–484, Oct. 1993. DOI: 10.1017/S095679680000085X.

[18] R. Harper and M. Lillibridge. Operational interpretations of an extension of F_ω with control operators. *Journal of Functional Programming*, 6(3):393–418, 1996. DOI: 10.1017/S0956796800001775.

[19] S. Ikeda and K. Nakazawa. Strong normalization proofs by CPS-translations. *Information Processing Letters*, 99(4):163–170, 2006. DOI: 10.1016/j.ipl.2006.03.009.

[20] Y. Kameyama and K. Asai. Strong normalization of polymorphic calculus for delimited continuations. In *Proceedings of the Austrian-Japanese Workshop on Symbolic Computation in Software Science (SCSS 2008), RISC-Linz Report Series No. 08-08*, pages 96–108, Hagenberg, Austria, July 2008.

[21] J.-L. Krivine. Realizability in classical logic. *Interactive models of computation and program behaviour*, 27:197–229, 2009.

[22] S. Lengrand and A. Miquel. Classical F_ω, orthogonality and symmetric candidates. *Annals of Pure Applied Logic*, 153(1-3):3–20, 2008. DOI: 10.1016/j.apal.2008.01.005.

[23] S. Lindley and I. Stark. Reducibility and TT-lifting for computation types:. In P. Urzyczyn, editor, *Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2005. DOI: 10.1007/11417170_20.

[24] P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In P. Panangaden, editor, *Proceedings of the Twentieth Annual IEEE Symposium on Logic in Computer Science*, pages 82–91, Chicago, IL, June 2005. IEEE Computer Society Press. DOI: 10.1109/LICS.2005.42.

[25] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997. DOI: 10.2307/2275652.

[26] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975. DOI: 10.1016/0304-3975(75)90017-1.

[27] H. Schwichtenberg. Proofs, lambda terms and control operators. In H. Schwichtenberg, editor, *Logic of Computation, volume 157 of Series F: Computer and Systems Sciences, NATO ASI Series, Marktoberdorf, Germany, July 25 - August 6, 1995*, pages 309–347, 1997. DOI: 10.1007/978-3-642-59048-1_9.

[28] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967. DOI: 10.2307/2271658.

[29] A. S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer-Verlag, 1973. DOI: 10.1007/BFb0066739.

[30] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115:38–94, 1994. DOI: 10.1006/inco.1994.1093.