ROPAS
Research On Program Analysis System
National Creative Research Initiative Center
Korea Advanced Institute of Science and Technology

ROPAS MEMO
2000-8
October 4, 2000

Proving Exception Stackability and Linearity in an Ordered Logical Framework

Jeff Polakow
Department of Computer Science
Carnegie Mellon University
`jpolakow@cs.cmu.edu`

Kwangkeun Yi
Department of Computer Science
KAIST
`kwang@cs.kaist.ac.kr`

October 4, 2000

Abstract

We formally prove the stackability and linearity of exception handlers of ML-style semantics using a novel proof technique via an ordered logical framework (OLF). We first transform exceptions into continuation-passing-style (CPS) terms and formalize the exception properties as a judgement on the CPS terms. Then, rather than directly proving that the properties hold for terms, we prove our theorem for the representations of the CPS terms and transform in OLF. We rely upon the correctness of our representations to transfer the results back to the actual CPS terms and transform.

Our work can be seen as two-fold: we present a theoretical justification of using the stack mechanism to implement exceptions of ML-like semantics; and we demonstrate the value of an ordered logical framework as a conceptual tool in the theoretical study of programming languages.

1 Introduction

Exception handling facilities in modern languages like ML [MTHM97, LDG$^+$00] or Java allow the programmer to define, raise and handle exceptional conditions. Exceptional conditions are brought (by a raise expression) to the attention of another expression where the raised exceptions may be handled. Exceptions are not necessarily limited to dealing with errors. The programmer can use exceptions as a "control diverter" to escape from any control structure to a point where the corresponding exception is handled. Also, using exceptions, the programmer can tailor an operation's results to particular purposes in a wider variety of contexts than would otherwise be the case.

In this article we formally prove a folklore property of exceptions: exception handlers are used at most once (linearity) in a stack-like-manner (stackability) (i.e., installing an exception handler and handling an exception respectively amounts to "push" and "pop."). Furthermore we show that the ordering properties investigated in [DDP99, DP95] for results of the conventional continuation-passing-style (CPS) transformation [DF92, Plo75, Ste78]— stackability of both continuation identifiers and continuation parameters— also hold for results of an extended CPS transform which replaces exception-raise and -handle expressions by function (continuation) calls and constructions in higher-order programs[KYD98, App97].

We prove the two properties as follows:

[0]This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology.

1. In order to expose the semantics of exceptions in the program text, we encode exceptions in source programs with continuations by using the extended CPS transformation.
2. We then formalize the properties of interest as a judgement on CPS terms.
3. We then prove that all terms resulting from the transformation satisfy our judgement.

We carry out the main portion of our proof (pt. 3 above) in a novel fashion— via an ordered logical framework (OLF) [PP00], a new formalism which is particularly well-suited for our purpose. Rather than directly proving that the properties hold for terms (which would require a rather tedious logical-relations style argument), we directly prove our theorem for representations of the CPS terms and transform in OLF. By working in OLF we can take advantage of known properties of OLF terms (e.g. substitution properties) which simplify our task. We then rely upon the correctness of our representations to transfer the results back to the actual CPS terms and transformation.

Our work can be seen as a theoretical justification of existing compilers that use the stack mechanism to implement exceptions. Our work also demonstrates the value of a (ordered) logical framework as a conceptual tool in the theoretical study of programming languages. We believe that working inside OLF greatly simplifies our proof. Of course such simplification comes at the cost of gaining familiarity with OLF. However, we feel the trade-off is advantageous. Logical frameworks have generally proven themselves to be useful tools for studying programming languages [Pfe96]; and we believe OLF, though still a new formalism, will likewise prove itself useful.

1.1 Overview

Section 2 introduces the ordered logical framework in which we will represent our terms and transform. In section 3.2 we define direct-style terms with exception raise and handle expressions, CPS terms, and the CPS transformation for encoding exception-raise and -handle expressions by continuations. We also define judgements on CPS terms for stackability (and linearity) of the exception handling mechanism. In section 4 we give OLF representations for direct-style terms and for CPS terms satisfying the stackability judgements. In section 5 we show the representation of the CPS transformation. This representation takes represented direct-style terms to represented CPS terms. The correctness of this representation completes our proof since represented CPS terms correspond to actual CPS terms satisfying the stackability judgements. Finally, we give a conclusion with some related and future work in section 7.

2 Ordered Logical Framework

OLF is a logical framework in the tradition of LF [HHP93] and its linear extension LLF [CP99]. Thus OLF is essentially a dependent type theory[1] for which type checking is decidable and canonical forms exist. Since OLF has come under study quite recently, the remainder of this section provides the necessary background information to follow our proof.

OLF should be thought of as ordered linear types [PP99a] extended with dependent types. Thus, we will first review Ordered Linear Logic, the logic corresponding to ordered linear types.

[1]Types can depend upon terms.

2.1 Ordered Linear Logic

Ordered linear logic (OLL) is a conservative extension of intuitionistic linear logic with ordered hypotheses. We begin with a review of the fragment of OLL which we will use. For a description of the full system see [PP99b, PP99a].

Types	A	$::=$	a	atomic types
		$\mid$	$A_0 \rightarrow A_1$	intuitionistic implication
		$\mid$	$A_0 \twoheadrightarrow A_1$	ordered right implication
		$\mid$	$A_0 \mathbin{\&} A_1$	additive conjunction
		$\mid$	$\top$	additive truth

Objects	M	$::=$	c	constants
		$\mid$	$x \mid z$	variables
		$\mid$	$\lambda x{:}A.\ M \mid M_0\, M_1$	intuitionistic functions $(A \rightarrow B)$
		$\mid$	$\overset{>}{\lambda} z{:}A.\ M \mid M_0 \overset{>}{\ } M_1$	right ordered functions $(A \twoheadrightarrow B)$
		$\mid$	$\langle M\ ,\ N\rangle \mid \pi_1\, M \mid \pi_2\, M$	additive pairs $(A \mathbin{\&} B)$
		$\mid$	$\langle\rangle$	additive unit $(\top)$

We build typing rules for OLL objects from the following judgement

$$\Gamma; \Omega \vdash M : A$$

where Γ is a list of unrestricted hypotheses, Ω is a list of ordered hypotheses, and a signature containing constant declarations is left implicit. The inference rules will be structured to allow copying, discarding, and exchanging of unrestricted hypotheses. However, ordered hypotheses will not enjoy those structural properties— they must be used exactly once in their relative order.

Here are the rules for unrestricted functions.

$$\frac{}{(\Gamma_1, x{:}A, \Gamma_2); \cdot \vdash x : A}\ \textbf{ivar} \qquad \frac{(\Gamma, x{:}A); \Omega \vdash M : B}{\Gamma; \Omega \vdash \lambda x{:}A.\ M : A \rightarrow B} \rightarrow I$$

$$\frac{\Gamma; \Omega \vdash M : A \rightarrow B \qquad \Gamma; \cdot \vdash N : A}{\Gamma; \Omega \vdash M\, N : B} \rightarrow E$$

Note that the ordered context in the minor premise of $\rightarrow_E$ must be empty. This ensures that unrestricted functions, which may use their argument arbitrarily, may not be applied to ordered arguments, which must be used exactly once.

The rules for ordered functions follow.

$$\frac{}{\Gamma; z{:}A \vdash z : A}\ \textbf{ovar} \qquad \frac{\Gamma; (\Omega, z{:}A) \vdash M : B}{\Gamma; \Omega \vdash \overset{>}{\lambda} z{:}A.\ M : A \twoheadrightarrow B} \twoheadrightarrow I$$

$$\frac{\Gamma; \Omega_1 \vdash M : A \twoheadrightarrow B \qquad \Gamma; \Omega_2 \vdash N : A}{\Gamma; (\Omega_1, \Omega_2) \vdash M \overset{>}{\ } N : B} \twoheadrightarrow E$$

Note that the argument to an ordered function may only depend upon ordered hypotheses to the right of those use by the body of the function—the order of the hypotheses constrains their use by ordered functions.

Finally we give the rules for pairs and unit.

$$\frac{\Gamma;\Omega \vdash M : A \qquad \Gamma;\Omega \vdash N : B}{\Gamma;\Omega \vdash \langle M\,,\,N\rangle : A \mathbin{\&} B}\,\&_I \qquad\qquad \frac{}{\Gamma;\Omega \vdash \langle\rangle : \top}\,\top_I$$

$$\frac{\Gamma;\Omega \vdash M : A \mathbin{\&} B}{\Gamma;\Omega \vdash \pi_1\, M : A}\,\&_{E1} \qquad\qquad \frac{\Gamma;\Omega \vdash M : A \mathbin{\&} B}{\Gamma;\Omega \vdash \pi_2\, M : B}\,\&_{E2}$$

The reduction rules for OLL objects are simply β-reduction for both kinds of functions. The appropriate notion of equality of objects also includes η-conversion so that every well-typed object has an equivalent canonical form.

Our calculus enjoys subject reduction, as proved in [PP99a].

Theorem 1 (Subject Reduction)
If $M \Longrightarrow M'$ *and* $\Gamma;\Omega \vdash M : A$ *then* $\Gamma;\Omega \vdash M' : A$.

Proof: For each reduction, we apply inversion to the given typing derivation and then use a substitution lemma to obtain the typing derivation for the conclusion. □

Finally, we note that this calculus has canonical forms as shown in [PP99a]. Thus all terms of functional type may be converted to the form $\lambda x{:}A.\ M$ or $\overset{>}{\lambda} z{:}A.\ M$; all terms of conjunctive type may be converted to pairs $\langle M\,,\,N\rangle$; and all objects of atomic type may be reduced to a constant applied to zero or more canonical objects.

The existence of canonical forms for this simple implicational fragment of OLL provides a basis for an ordered logical framework. We conjecture that an ordered logical framework based on a full type theory can be constructed along the lines of the linear logical framework [CP99]. In this paper we only need a two-level fragment as explained in subsection 2.2.

2.2 Two-Level Framework

We extend the ordered λ-calculus from subsection 2.1 to a simple two-level logical framework. Level 2 type families p are indexed by level 1 objects M, and we can quantify only over level 1 objects.

$$\begin{array}{llll}
\textit{Level 2 types} & F & ::= & p\, M_1 \ldots M_n \\
 & & \mid & F_1 \rightarrow F_2 \\
 & & \mid & F_1 \twoheadrightarrow F_2 \\
 & & \mid & F_1 \mathbin{\&} F_2 \\
 & & \mid & \top \\
 & & \mid & \Pi x{:}A.\ F
\end{array}
\qquad
\begin{array}{llll}
\textit{Level 2 objects} & D & ::= & c \mid w \mid y \\
 & & \mid & \lambda w{:}F.\ D \mid D_1\, D_2 \\
 & & \mid & \overset{>}{\lambda} y{:}F.\ D \mid D_1 {}^{>} D_2 \\
 & & \mid & \langle D_1\,,\,D_2\rangle \mid \pi_1\, D \mid \pi_2\, D \\
 & & \mid & \langle\rangle \\
 & & \mid & \lambda x{:}A.\ D \mid D\, M
\end{array}$$

The extended typing judgement now has the form $\Gamma;\Omega \vdash D : F$, where Γ may contain declarations of the form $x{:}A$ or $w{:}F$ and Ω contains declarations $y{:}F$. We omit the typing rules which are very similar to the propositional case, except that we now need rules for the dependent types:

$$\frac{\Gamma, x{:}A;\Omega \vdash D : F}{\Gamma;\Omega \vdash \lambda x{:}A.\ D : \Pi x{:}A.\ F} \qquad\qquad \frac{\Gamma;\Omega \vdash D : \Pi x{:}A.\ F \qquad \Gamma;\cdot \vdash M : A}{\Gamma;\Omega \vdash D\, M : F[M/x]}$$

and a rule for type conversion:

$$\frac{\Gamma;\Omega \vdash D : F \qquad F \equiv_{\beta\eta} F'}{\Gamma;\Omega \vdash D : F'}$$

Since we stratify the type theory into two syntactically distinct levels, $\beta\eta$-equality for level-2 types immediately reduces to $\beta\eta$-equality for propositional objects. Since propositional objects possess canonical (= long $\beta\eta$-normal) forms, this equality is easy to decide, and type-checking in the fragment presented above can easily be seen to be decidable. Furthermore, canonical forms for level-2 objects likewise come as a consequence of level-1 objects having canonical forms. We will use the judgement

$$\Gamma; \Omega \vdash D \Uparrow F$$

to denote that object D is canonical at well-formed type F.

3 Terms & Transforms

This section introduces the direct-style language with exception raise and handle expressions, its CPS counterpart, and the transformation between them. We use underlined constants (e.g. $\underline{\mathsf{handle}}$) and lambdas ($\underline{\lambda}$) to distinguish these objects from their OLF representations which are given in section 4.

3.1 Direct Terms

We use the following syntax for direct-style (DS) terms:

$$\begin{array}{lrcl} \textit{DS Terms} & r & ::= & e \\ \textit{DS Expressions} & e & ::= & e_0\, e_1 \mid \underline{\mathsf{handle}}\, e_0\, (\underline{\lambda} x.\ e_1) \mid \underline{\mathsf{raise}}\, e \mid t \\ \textit{DS Trivial Expressions} & t & ::= & \underline{\lambda} x.\ r \mid x \end{array}$$

Evaluating the expression $\underline{\mathsf{raise}}\, e$ first evaluates e. It then aborts the normal execution and locates a handler by going up the current evaluation chain. The e's value is passed to handler. The handle expression $\underline{\mathsf{handle}}\, e_0\, (\underline{\lambda} x.e_1)$ evaluates e_0. If e_0 raises an exception with value v and there are no other handle expressions between the current one and the raise expression, then the current handler function $\underline{\lambda} x.e_1$ handles it: the v is bound to x in e_1. Otherwise, the value of the handle expression is the value of e_0.

We define the formal semantics of DS terms with a structural operational semantics [Plo81] using Felleisen's evaluation contexts [Fel87]. In doing so, we need to extend the expressions to contain a set of raised values $\bar{t}$ that are thrown from raise expressions: $e ::= \cdots \mid \bar{t}$. An evaluation context C is defined by the following grammar:

$$C \ ::= \ [\,] \mid C\ e \mid t\ C \mid \underline{\mathsf{handle}}\, C\, \underline{\lambda} x.\ e \mid \underline{\mathsf{raise}}\, C$$

This context defines a left-to-right, call-by-value reduction. As usual, we write $C[e]$ if the hole in context C is filled with e. We use this context to define the reduction rule for arbitrary expressions:

$$\frac{e \mapsto e'}{C[e] \mapsto C[e']}$$

The single reduction step $e \mapsto e'$ for a redex e consists of normal and exceptional reduction steps:

Normal reduction steps	Exceptional reduction steps
$(\underline{\lambda} x.\ e)\, t \ \mapsto\ [t/x]e$	$\underline{\mathsf{raise}}\, t \ \mapsto\ \bar{t}$
$\underline{\mathsf{handle}}\, t\, (\underline{\lambda} x.\ e) \ \mapsto\ t$	$\underline{\mathsf{raise}}\, \bar{t} \ \mapsto\ \bar{t}$
	$\underline{\mathsf{handle}}\, \bar{t}\, (\underline{\lambda} x.\ e) \ \mapsto\ [t/x]e$
	$\bar{t}\, e \ \mapsto\ \bar{t}$
	$(\underline{\lambda} x.\ e)\, \bar{t} \ \mapsto\ \bar{t}$

Normal reduction steps are not concerned with exceptions. Exceptional reduction steps specify the generation, propagation and handling of exceptions.

3.2 CPS Terms

Rather than working directly with DS terms, we transform them into CPS terms where the exception mechanism is captured by having a second (handler) continuation in addition to the regular (success) continuation. This transformation exposes the semantics of exceptions in the program text. We use the following grammar for CPS terms:

Root Terms	r	$::=$	$\underline{\lambda} k.\ e$
Serious Terms	e	$::=$	$t_0\, t_1\, p \mid c\, t$
Trivial Terms	t	$::=$	$\underline{\lambda} x.\ r \mid x \mid v$
Continuation Pairs	p	$::=$	$\underline{\mathsf{pair}}(c_0\ ,\ c_1) \mid k$
Continuation Terms	c	$::=$	$\underline{\lambda} x.\ e \mid \underline{\lambda} v.\ e \mid \underline{\mathsf{nrml}}\, p \mid \underline{\mathsf{hnd}_0}\, p \mid \underline{\mathsf{hnd}_1}\, v\, p$

Note that in the CPS syntax, we are distinguishing variables x which are parameters of functions or continuations from variables v which are only parameters to continuations. This distinction will be used to differentiate abstractions introduced by the transform from those already present in the DS term. $\underline{\mathsf{nrml}}$ and $\underline{\mathsf{hnd}_0}$ are essentially the projections for the continuation pairs; and $\underline{\mathsf{hnd}_1}$ is used to explicitly differentiate the case when a stacked intermediate value must be popped (aborted) to reach the handler.

The formal semantics are defined similarly to that for DS terms. However, rather than using special exception values ($\bar{t}$); exceptional flows ($\underline{\mathsf{raise}}$ and $\underline{\mathsf{handle}}$ expressions) are simulated by continuation functions. Let the set of result values, γ, consist of trivial terms and immediate functions:

$$\gamma\ ::=\ t \mid \underline{\lambda} x.e \mid \underline{\lambda} v.e \mid \underline{\lambda} k.e$$

An evaluation context C is extended for the cases of continuation pairs:

$$\begin{array}{rcl} C & ::= & [\,] \mid C\,\gamma \mid \gamma\, C \\ & \mid & \underline{\mathsf{pair}}(C\ ,\ c) \mid \underline{\mathsf{pair}}(\gamma\ ,\ C) \\ & \mid & \underline{\mathsf{nrml}}\, C \mid \underline{\mathsf{hnd}_0}\, C \mid \underline{\mathsf{hnd}_1}\, v\, C \end{array}$$

The single reduction step $e \mapsto e'$ for a redex e is:

Reduction steps

$$\begin{array}{lcl@{\qquad\qquad}rcl} (\underline{\lambda} x.\ r)t & \mapsto & [t/x]r & \underline{\mathsf{nrml}}\, \underline{\mathsf{pair}}(\gamma_0\ ,\ \gamma_1) & \mapsto & \gamma_0 \\ (\underline{\lambda} x.\ e)t & \mapsto & [t/x]e & \underline{\mathsf{hnd}_0}\, \underline{\mathsf{pair}}(\gamma_0\ ,\ \gamma_1) & \mapsto & \gamma_1 \\ (\underline{\lambda} v.\ e)t & \mapsto & [t/v]e & \underline{\mathsf{hnd}_1}\, v\, \underline{\mathsf{pair}}(\gamma_0\ ,\ \gamma_1) & \mapsto & \gamma_1 \end{array}$$

3.3 Continuation-passing-style (CPS) Transformation

We use an extension of the conventional continuation-passing-style (CPS) transformation [DF92, Plo75, Ste78] to get from a DS term to a CPS term. We remove the $\underline{\mathsf{raise}}$ and $\underline{\mathsf{handle}}$ expressions by passing two continuations to each expression: one for the normal course of execution, and a second one (the handler continuation) for exceptions.

Only $\underline{\mathsf{raise}}$ and $\underline{\mathsf{handle}}$ expressions use the handler continuation. A $\underline{\mathsf{raise}}$ expression is transformed to call the current handler continuation. A $\underline{\mathsf{handle}}$ expression is transformed to extend the handler function with the current handler continuation. For other expressions, the handler continuation is passively passed along, reflecting the dynamic scope of exceptions. Because

$$\begin{array}{l}
[\![-]\!]^R : \textit{DS Terms} \to \textit{Root Terms} \\
[\![-]\!]^E : \textit{DS Expressions} \to \textit{Continuation Pairs} \to \textit{Serious Terms} \\
[\![-]\!]^T : \textit{DS Trivial Expressions} \to \textit{Trivial Terms}
\end{array}$$

In mnemonic CPS term syntax:

$$\begin{array}{rcl}
[\![e]\!]^R & = & \underline{\lambda}\langle n,h\rangle.\ T_e\,[\![e]\!]\,\langle n,h\rangle \\
[\![e_0\ e_1]\!]^E\,\langle n,h\rangle & = & [\![e_0]\!]^E\,\langle\underline{\lambda}v_0.\ [\![e_1]\!]^E\,\langle\underline{\lambda}v_1.\ v_0\,v_1\,\langle n,h\rangle, h\rangle\,h\rangle \\
[\![\underline{\mathsf{handle}}\,e_0\,(\underline{\lambda}x.\ e_1)]\!]^E\,\langle n,h\rangle & = & [\![e_0]\!]^E\,\langle n, \underline{\lambda}x.\ [\![e_1]\!]^E\,\langle n,h\rangle\rangle \\
[\![\underline{\mathsf{raise}}\,e]\!]^E\,\langle n,h\rangle & = & [\![e]\!]^E\,\langle h,h\rangle \\
[\![t]\!]^E\,\langle n,h\rangle & = & n\,[\![t]\!]^T \\
[\![x]\!]^T & = & x \\
[\![\underline{\lambda}x.\ r]\!]^T & = & \underline{\lambda}x.[\![r]\!]^R
\end{array}$$

In the exact CPS term syntax:

$$\begin{array}{rcl}
[\![e]\!]^R & = & \underline{\lambda}k.\ [\![e]\!]^E\,k \\
[\![e_0\ e_1]\!]^E\,p & = & [\![e_0]\!]^E\,\underline{\mathsf{pair}}(\underline{\lambda}v_0.\ [\![e_1]\!]^E\,\underline{\mathsf{pair}}(\underline{\lambda}v_1.\ v_0\,v_1\,p\,,\ \underline{\mathsf{hnd}_1}\,v_0\,p)\,,\ \underline{\mathsf{hnd}_0}\,p) \\
[\![\underline{\mathsf{handle}}\,e_0\,(\underline{\lambda}x.\ e_1)]\!]^E\,p & = & [\![e_0]\!]^E\,\underline{\mathsf{pair}}(\underline{\mathsf{nrml}}\,p\,,\ \underline{\lambda}x.\ [\![e_1]\!]^E\,p) \\
[\![\underline{\mathsf{raise}}\,e]\!]^E\,p & = & [\![e]\!]^E\,\underline{\mathsf{pair}}(\underline{\mathsf{hnd}_0}\,p\,,\ \underline{\mathsf{hnd}_0}\,p) \\
[\![t]\!]^E\,p & = & (\underline{\mathsf{nrml}}\,p)\,[\![t]\!]^T \\
[\![x]\!]^T & = & x \\
[\![\underline{\lambda}x.\ r]\!]^T & = & \underline{\lambda}x.[\![r]\!]^R
\end{array}$$

Figure 1: CPS transformation function $[\![-]\!]^R$

the handler continuation encodes both how to handle a raised exception and how to proceed thereafter, we have to make the normal continuation ready to be captured by a handler continuation. Thus we keep passing two continuations (normal and handler continuations) to every expression.

Figure 1 shows the extended CPS transform in a conventional functional formulation: one in a mnemonic style and the other in our exact CPS term syntax. Notice the use of $\underline{\mathsf{hnd}_1}$ in the inner handler continuation of the application translation. That inner handler will only be invoked if the evaluation of e_0 succeeds, pushing an intermediate value v_0 onto the stack, and then the evaluation of e_1 causes an exception. $\underline{\mathsf{hnd}_1}$ is necessary, rather than $\underline{\mathsf{hnd}_0}$, because the stacked intermediate value v_0 must be popped to reach the the inner handler sitting beneath v_0 in the stack.

The correctness of this CPS transformation can be proven[KYD98] analogously to the proof of Plotkin's simulation theorem [HD97, Plo75].

3.4 CPS Transformation as a Judgement

In order to represent the transform in OLF, we reformulate it as three mutually recursive judgements corresponding to $[\![-]\!]^R$, $[\![-]\!]^E$, and $[\![-]\!]^T$ in Figure 1. A direct-style term r is transformed into a CPS term r' whenever the judgement

$$\vdash r \xrightarrow{DR} r'$$

is satisfied. Given a continuation pair p , a direct-style expression e is transformed into a CPS expression e' whenever the judgement

$$\vdash e\,;\,p \xrightarrow{DE} e'$$

is satisfied. Finally, a direct-style trivial expression t is transformed into a CPS trivial expression t' whenever the judgement

$$\vdash t \xrightarrow{DT} t'$$

is satisfied.

The derivation rules for the transform are as follows:

$$\frac{\vdash e\,;\,k \xrightarrow{DE} e'}{\vdash e \xrightarrow{DR} \underline{\lambda} k.\; e'}$$

$$\frac{\vdash t \xrightarrow{DT} t'}{\vdash t\,;\,p \xrightarrow{DE} (\underline{\mathsf{nrml}}\,p)\,t'} \qquad \frac{\vdash e\,;\,\underline{\mathsf{pair}}(\underline{\mathsf{hnd}_0}\,p\,,\,\underline{\mathsf{hnd}_0}\,p) \xrightarrow{DE} e'}{\vdash \underline{\mathsf{raise}}\,e\,;\,p \xrightarrow{DE} e'}$$

$$\frac{\vdash e_1\,;\,\underline{\mathsf{pair}}(\underline{\lambda} v_1.\; v_0\,v_1\,p\,,\,\underline{\mathsf{hnd}_1}\,v_0\,p) \xrightarrow{DE} e_1' \qquad \vdash e_0\,;\,\underline{\mathsf{pair}}(\underline{\lambda} v_0.\; e_1'\,,\,\underline{\mathsf{hnd}_0}\,p) \xrightarrow{DE} e'}{\vdash e_0\,e_1\,;\,p \xrightarrow{DE} e'} \quad \begin{array}{l} v_0 \text{ not free} \\ \text{in conclusion} \end{array}$$

$$\frac{\vdash e_1\,;\,p \xrightarrow{DE} e_1' \qquad \vdash e_0\,;\,\underline{\mathsf{pair}}(\underline{\mathsf{nrml}}\,p\,,\,\underline{\lambda} x.\; e_1') \xrightarrow{DE} e'}{\vdash \underline{\mathsf{handle}}\,e_0\,(\underline{\lambda} x.\; e_1)\,;\,p \xrightarrow{DE} e'}$$

$$\frac{}{\vdash x \xrightarrow{DT} x} \qquad \frac{\vdash r \xrightarrow{DR} r'}{\vdash \underline{\lambda} x.\; r \xrightarrow{DT} \underline{\lambda} x.\; r'}$$

3.5 Invariants for Results of CPS Transform

Terms resulting from a left-to-right call-by-value CPS translation of direct-style terms satisfy an invariant on occurrences of continuation identifiers k and parameters v. We shall formalize this property with five mutually recursive judgements:

$$\models^{\mathbf{Root}} r \qquad \Phi \models^{\mathbf{Exp}} e \qquad \Phi \models^{\mathbf{Triv}} t; \Phi' \qquad \Phi \models^{\mathbf{CPair}} p \qquad \Phi \models^{\mathbf{Cont}} c$$

where Φ is a stack of both continuation identifiers and parameters:

$$\Phi ::= \cdot \mid \Phi, k \mid \Phi, v$$

When Φ' is a prefix of Φ, we define $\Phi - \Phi'$ as the remainder of Φ.

The derivation rules for these judgements are as follows:

$$\frac{k \models^{\mathbf{Exp}} e}{\models^{\mathbf{Root}} \underline{\lambda} k.\; e}$$

$$\frac{\Phi \models^{\mathbf{Triv}} t; \Phi' \qquad \Phi' \models^{\mathbf{Cont}} c}{\Phi \models^{\mathbf{Exp}} c\, t} \qquad \frac{\Phi \models^{\mathbf{Triv}} t_1; \Phi' \qquad \Phi' \models^{\mathbf{Triv}} t_0; \Phi'' \qquad \Phi'' \models^{\mathbf{CPair}} p}{\Phi \models^{\mathbf{Exp}} t_0\, t_1\, p}$$

$$\frac{}{\Phi \models^{\mathbf{Triv}} x; \Phi} \qquad \frac{\models^{\mathbf{Root}} r}{\Phi \models^{\mathbf{Triv}} \underline{\lambda} x.\ r; \Phi} \qquad \frac{}{\Phi, v \models^{\mathbf{Triv}} v; \Phi}$$

$$\frac{}{k \models^{\mathbf{CPair}} k} \qquad \frac{\Phi \models^{\mathbf{Cont}} c_0 \qquad \Phi \models^{\mathbf{Cont}} c_1}{\Phi \models^{\mathbf{CPair}} \underline{\mathsf{pair}}(c_0\ , c_1)}$$

$$\frac{\Phi \models^{\mathbf{Exp}} e}{\Phi \models^{\mathbf{Cont}} \underline{\lambda} x.\ e} \qquad \frac{\Phi, v \models^{\mathbf{Exp}} e}{\Phi \models^{\mathbf{Cont}} \underline{\lambda} v.\ e} \qquad \frac{\Phi \models^{\mathbf{CPair}} p}{\Phi \models^{\mathbf{Cont}} \underline{\mathsf{nrml}}\, p} \qquad \frac{\Phi \models^{\mathbf{CPair}} p}{\Phi \models^{\mathbf{Cont}} \underline{\mathsf{hnd}_0}\, p} \qquad \frac{\Phi \models^{\mathbf{CPair}} p}{\Phi, v \models^{\mathbf{Cont}} \underline{\mathsf{hnd}_1}\, v\, p}$$

From the judgement rules, it is easy to see that continuation-pair identifiers, k, are used linearly in each root term, and that continuation parameters v (which were introduced by the CPS transform) form a stack in each serious term. In fact, the judgement actually implies the stronger property that continuation-pair identifiers and parameters are used together in a stack-like fashion. Each root term adds a new stack-frame, an identifier followed by parameters, which is fully consumed within that root term. This is apparent from the judgement on $\underline{\lambda} x.r$ which requires that r not depend upon anything currently in the stack.

We now state one further property of our cps transform.

Lemma 2 $\vdash e; p \xrightarrow{DE} e'$ *and* $\Phi \models^{\mathbf{CPair}} p$ *implies* $\Phi \models^{\mathbf{Exp}} e'$.

Proof: By structural induction on $\vdash e; p \xrightarrow{DE} e'$. For base case, $\vdash t; p \xrightarrow{DE} (\underline{\mathsf{nrml}}\, p)\, t'$, note that $t' \neq v$ thus $\Phi \models^{\mathbf{Triv}} t'; \Phi$ for all Φ. □

We would like to prove that $\vdash r \xrightarrow{DR} r'$ implies $\models^{\mathbf{Root}} r'$. Proving this directly with the above definitions requires a logical relations style argument [DDP99, DP95]. However, by using an ordered logical framework, this may be proved directly.

4 Ordered Logical Framework Representation

In this section, we show how to represent the terms and transform of section 3 in OLF; and how these representations immediately give our desired proof. Following standard practice for LF-style logical frameworks, we shall represent judgements as types and derivations as terms [HHP93] [2]. Furthermore, we will take care that all of our representations are compositional bijections— 1) for every actual object represented there is a corresponding OLF object (and vice-versa); and 2) the representation function and its inverse both commute with substitution. These two properties allow us to transfer results for the representations back to the actual objects and vice-versa. Representations which are compositional bijections are sometimes referred to as adequate.

Our proof proceeds in the following manner.

[2]For representing abstract syntax (e.g. DS terms) we may view each syntactic category as a judgement and the constructors for terms of the category as derivation rules for the judgement.

1. We give a representation for DS terms which is in compositional bijection with all actual DS terms.
2. We give a representation for CPS terms which is in compositional bijection with *only* actual CPS terms satisfying the invariants; our representation does not capture all terms within the CPS grammar of section 3.2.
3. We give a representation for the CPS transform of section 3.4. This representation relates represented DS terms to represented CPS terms. Furthermore it is in compositional bijection with *all* possible CPS transformations.
4. By using the preceding compositional bijections, we conclude that $\vdash r \xrightarrow{DR} r'$ implies $\models^{\mathbf{Root}} r'$.

4.1 DS Terms

Our representation of DS terms will use three basic types corresponding to the three kinds of DS terms.

$$\mathsf{droot} : \mathsf{type}. \qquad \mathsf{dexp} : \mathsf{type}. \qquad \mathsf{dtriv} : \mathsf{type}.$$

We will then build our representations from term constructors corresponding to DS terms. Note that representation uses higher-order abstract syntax, so object-level functions are represented by meta-level functions and likewise object-level variables are represented (implicitly) by meta-level variables.

$$\begin{array}{rcl}
\mathsf{e2r} & : & \mathsf{dexp} \rightarrow \mathsf{droot}. \\
\mathsf{dapp} & : & \mathsf{dexp} \rightarrow \mathsf{dexp} \rightarrow \mathsf{dexp}. \\
\mathsf{handle} & : & \mathsf{dexp} \rightarrow (\mathsf{dexp} \rightarrow \mathsf{dexp}) \rightarrow \mathsf{dexp}. \\
\mathsf{raise} & : & \mathsf{dexp} \rightarrow \mathsf{dexp}. \\
\mathsf{t2e} & : & \mathsf{dtriv} \rightarrow \mathsf{dexp}. \\
\mathsf{dabort} & : & \mathsf{dtriv}. \\
\mathsf{dlam} & : & (\mathsf{triv} \rightarrow \mathsf{droot}) \rightarrow \mathsf{dtriv}.
\end{array}$$

Given the previous signature, there is an obvious compositional bijection between DS terms and canonical objects in the above signature. This bijection is established by the following mutually recursive representation functions, $\ulcorner - \urcorner^R$,$\ulcorner - \urcorner^E$,$\ulcorner - \urcorner^T$, and their inverses $\llcorner - \lrcorner^R$,$\llcorner - \lrcorner^E$,$\llcorner - \lrcorner^T$.

$$\begin{array}{rcl}
\ulcorner e \urcorner^R & = & \mathsf{e2r}\,\ulcorner e \urcorner^E
\end{array}
\qquad
\begin{array}{rcl}
\llcorner \mathsf{e2r}\,E \lrcorner_R & = & \llcorner E \lrcorner_E
\end{array}$$

$$\begin{array}{rcl}
\ulcorner e_0\,e_1 \urcorner^E & = & \mathsf{dapp}\,\ulcorner e_0 \urcorner^E \ulcorner e_1 \urcorner^E
\end{array}
\qquad
\begin{array}{rcl}
\llcorner \mathsf{dapp}\,E_0\,E_1 \lrcorner_E & = & \llcorner E_0 \lrcorner_E \,\llcorner E_1 \lrcorner_E
\end{array}$$

$$\begin{array}{rcl}
\ulcorner \underline{\mathsf{handle}}\,e_0\,(\underline{\lambda} x.\ e_1) \urcorner^E & = & \mathsf{handle}\,\ulcorner e_0 \urcorner^E\,(\lambda x{:}\mathsf{dtriv}.\ \ulcorner e_1 \urcorner^E) \\
\llcorner \mathsf{handle}\,E_0\,(\lambda x{:}\mathsf{dtriv}.\ E_1) \lrcorner_E & = & \underline{\mathsf{handle}}\,\llcorner E_0 \lrcorner_E\,(\underline{\lambda} x.\ \llcorner E_1 \lrcorner_E)
\end{array}$$

$$\begin{array}{rcl}
\ulcorner \underline{\mathsf{raise}}\,e \urcorner^E & = & \mathsf{raise}\,\ulcorner e \urcorner^E \\
\ulcorner t \urcorner^E & = & \mathsf{d2e}\,\ulcorner t \urcorner^T
\end{array}
\qquad
\begin{array}{rcl}
\llcorner \mathsf{raise}\,E \lrcorner_E & = & \underline{\mathsf{raise}}\,\llcorner E \lrcorner_E \\
\llcorner \mathsf{d2e}\,T \lrcorner_E & = & \llcorner T \lrcorner_T
\end{array}$$

$$\begin{array}{rcl}
\ulcorner \underline{\lambda} x.\ r \urcorner^T & = & \mathsf{dlam}\,(\lambda x{:}\mathsf{dtriv}.\ \ulcorner r \urcorner^R) \\
\ulcorner x \urcorner^T & = & x
\end{array}
\qquad
\begin{array}{rcl}
\llcorner \mathsf{dlam}\,(\lambda x{:}\mathsf{dtriv}.\ R) \lrcorner_T & = & \underline{\lambda} x.\ \llcorner R \lrcorner_R \\
\llcorner x \lrcorner_T & = & x
\end{array}$$

4.2 CPS Terms

Next, we give a representation of CPS terms satisfying the invariants of section 3.5. The key idea behind this representation is that ordered types implicitly capture the invariants. Thus, we can directly represent CPS terms which satisfy the invariants, without explicitly representing the invariants. Our representation will use five basic types corresponding to the five basic kinds of CPS terms.

$$\mathsf{root} : \mathsf{type}. \qquad \mathsf{exp} : \mathsf{type}. \qquad \mathsf{triv} : \mathsf{type}. \qquad \mathsf{cont} : \mathsf{type}. \qquad \mathsf{cpair} : \mathsf{type}.$$

We will then build our representations from term constructors corresponding to CPS terms. The use of ordered types forces the CPS term representations to satisfy the invariants.

$$\begin{array}{rcl}
\mathsf{klam} & : & (\mathsf{cpair} \twoheadrightarrow \mathsf{exp}) \to \mathsf{root}. \\
\mathsf{app} & : & \mathsf{cpair} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
\mathsf{kapp} & : & \mathsf{cont} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
\mathsf{lam} & : & (\mathsf{triv} \to \mathsf{root}) \to \mathsf{triv}. \\
\mathsf{xlam} & : & (\mathsf{triv} \to \mathsf{exp}) \twoheadrightarrow \mathsf{cont}. \\
\mathsf{vlam} & : & (\mathsf{triv} \twoheadrightarrow \mathsf{exp}) \twoheadrightarrow \mathsf{cont}. \\
\mathsf{nrml} & : & \mathsf{cpair} \twoheadrightarrow \mathsf{cont}. \\
\mathsf{hnd}_0 & : & \mathsf{cpair} \twoheadrightarrow \mathsf{cont}. \\
\mathsf{hnd}_1 & : & \mathsf{cpair} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{cont}. \\
\mathsf{pair} & : & (\mathsf{cont} \,\&\, \mathsf{cont}) \twoheadrightarrow \mathsf{cpair}.
\end{array}$$

Note that a positive occurrence of an unrestricted function $\to$ as in the type of klam imposes a restriction on the corresponding argument: it may not depend upon continuation-pairs k nor parameters v which are always ordered variables. On the other hand, a negative occurrence of $\to$ as in the type of lam licenses the unrestricted use of the corresponding bound variable x. The right ordered functions $\twoheadrightarrow$ impose the stack-like discipline on parameters of continuations and the continuation-pairs themselves.

Given the previous signature, there is a compositional bijection between CPS terms satisfying the occurrence conditions and canonical objects in the above signature. This bijection is established by the following representation function, $\ulcorner - \urcorner$ and its inverse $\llcorner - \lrcorner$.

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\ulcorner \underline{\lambda} k.\, e \urcorner & = & \mathsf{klam}\,(\overset{>}{\lambda} k{:}\mathsf{cpair}.\, \ulcorner e \urcorner) & \llcorner \mathsf{klam}\,(\overset{>}{\lambda} k{:}\mathsf{cpair}.\, E) \lrcorner & = & \underline{\lambda} k.\, \llcorner E \lrcorner \\
\\
\ulcorner t_0\, t_1\, p \urcorner & = & \mathsf{app}^{>} \ulcorner p \urcorner^{>} \ulcorner t_0 \urcorner^{>} \ulcorner t_1 \urcorner & \llcorner \mathsf{app}^{>} P^{>} T_0{}^{>} T_1 \lrcorner & = & \llcorner T_0 \lrcorner \llcorner T_1 \lrcorner \llcorner P \lrcorner \\
\ulcorner c\, t \urcorner & = & \mathsf{kapp}^{>} \ulcorner c \urcorner^{>} \ulcorner t \urcorner & \llcorner \mathsf{kapp}^{>} C^{>} T \lrcorner & = & \llcorner C \lrcorner \llcorner T \lrcorner \\
\\
\ulcorner \underline{\lambda} x.\, r \urcorner & = & \mathsf{lam}\,(\lambda x{:}\mathsf{triv}.\, \ulcorner r \urcorner) & \llcorner \mathsf{lam}\,(\lambda x{:}\mathsf{triv}.\, R) \lrcorner & = & \underline{\lambda} x.\, \llcorner R \lrcorner \\
\ulcorner x \urcorner & = & x & \llcorner x \lrcorner & = & x \\
\ulcorner v \urcorner & = & v & \llcorner v \lrcorner & = & v \\
\\
\ulcorner \underline{\lambda} x.\, e \urcorner & = & \mathsf{xlam}^{>} (\underline{\lambda} x{:}\mathsf{triv}.\, \ulcorner e \urcorner) & \llcorner \mathsf{xlam}^{>} (\underline{\lambda} x{:}\mathsf{triv}.\, E) \lrcorner & = & \underline{\lambda} x.\, \llcorner E \lrcorner \\
\ulcorner \underline{\lambda} v.\, e \urcorner & = & \mathsf{vlam}^{>} (\overset{>}{\lambda} v{:}\mathsf{triv}.\, \ulcorner e \urcorner) & \llcorner \mathsf{vlam}^{>} (\overset{>}{\lambda} v{:}\mathsf{triv}.\, E) \lrcorner & = & \underline{\lambda} v.\, \llcorner E \lrcorner \\
\ulcorner \underline{\mathsf{nrml}}\, p \urcorner & = & (\mathsf{nrml}^{>} \ulcorner p \urcorner) & \llcorner \mathsf{nrml}^{>} P \lrcorner & = & \underline{\mathsf{nrml}}\, \llcorner P \lrcorner \\
\ulcorner \underline{\mathsf{hnd}_0}\, p \urcorner & = & (\mathsf{hnd}_0{}^{>} \ulcorner p \urcorner) & \llcorner \mathsf{hnd}_0{}^{>} P \lrcorner & = & \underline{\mathsf{hnd}_0}\, \llcorner P \lrcorner \\
\ulcorner \underline{\mathsf{hnd}_1}\, t\, p \urcorner & = & (\mathsf{hnd}_1{}^{>} \ulcorner p \urcorner^{>} \ulcorner t \urcorner) & \llcorner \mathsf{hnd}_1{}^{>} P^{>} T \lrcorner & = & \underline{\mathsf{hnd}_1}\, \llcorner T \lrcorner \llcorner P \lrcorner \\
\\
\ulcorner k \urcorner & = & k & \llcorner k \lrcorner & = & k \\
\ulcorner \underline{\mathsf{pair}}(c_0\,,\, c_1) \urcorner & = & \mathsf{pair}^{>} \langle \ulcorner c_0 \urcorner, \ulcorner c_1 \urcorner \rangle & \llcorner \mathsf{pair}^{>} \langle C_0\,,\, C_1 \rangle \lrcorner & = & \underline{\mathsf{pair}}(\llcorner C_0 \lrcorner\,,\, \llcorner C_1 \lrcorner)
\end{array}$$

Note that and $\llcorner\ulcorner u\urcorner\lrcorner = u$ for any term u. Additionally, since variables are mapped to variables, the representation function and its inverse are compositional (i.e., commute with substitution).

We formally prove the correspondence in two parts.

Theorem 3 (Representations are Canonical Forms)
Consider CPS terms r, e, t, c and p with free ordinary variables among $x_1, \ldots, x_n$. Let $\Gamma = x_1{:}\mathsf{triv} \ldots x_n{:}\mathsf{triv}$.

1. *If $\models^{\mathbf{Root}} r$ then $\Gamma; \cdot \vdash \ulcorner r\urcorner \Uparrow \mathsf{root}$.*
2. *If $\Phi \models^{\mathbf{Exp}} e$ then $\Gamma; \ulcorner\Phi\urcorner \vdash \ulcorner e\urcorner \Uparrow \mathsf{exp}$.*
3. *If $\Phi \models^{\mathbf{Triv}} t; \Phi'$ then $\Gamma; \ulcorner\Phi - \Phi'\urcorner \vdash \ulcorner t\urcorner \Uparrow \mathsf{triv}$.*
4. *If $\Phi \models^{\mathbf{Cont}} c$ then $\Gamma; \ulcorner\Phi\urcorner \vdash \ulcorner c\urcorner \Uparrow \mathsf{cont}$.*
5. *If $\Phi \models^{\mathbf{CPair}} p$ then $\Gamma; \ulcorner\Phi\urcorner \vdash \ulcorner p\urcorner \Uparrow \mathsf{cpair}$.*

Proof: By induction on the structure of the given derivations. □

Theorem 4 (Canonical Forms are Representations)
Let $\Gamma = x_1{:}\mathsf{triv}, \ldots, x_n{:}\mathsf{triv}$ be given.

1. *For any M such that $\Gamma; \cdot \vdash M \Uparrow \mathsf{root}$,*
   $\llcorner M\lrcorner$ is defined and $\models^{\mathbf{Root}} \llcorner M\lrcorner$.
2. *For any $\Omega = v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$ and M such that $\Gamma; k{:}\mathsf{cpair}, \Omega \vdash M \Uparrow \mathsf{exp}$,*
   $\llcorner M\lrcorner$ is defined and $\llcorner\Omega\lrcorner \models^{\mathbf{Exp}} \llcorner M\lrcorner$.
3. *For any $\Omega = v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$ and M such that $\Gamma; \Omega \vdash M \Uparrow \mathsf{triv}$,*
   $\llcorner M\lrcorner$ is defined and $\Phi, \llcorner\Omega\lrcorner \models^{\mathbf{Triv}} \llcorner M\lrcorner; \Phi$ for any Φ.
4. *For any $\Omega = v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$ and M such that $\Gamma; k{:}\mathsf{cpair}, \Omega \vdash M \Uparrow \mathsf{cont}$,*
   $\llcorner M\lrcorner$ is defined and $\llcorner\Omega\lrcorner \models^{\mathbf{Cont}} \llcorner M\lrcorner$.
5. *For any $\Omega = v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$ and M such that $\Gamma; k{:}\mathsf{cpair}, \Omega \vdash M \Uparrow \mathsf{cpair}$,*
   $\llcorner M\lrcorner$ is defined and $\llcorner\Omega\lrcorner \models^{\mathbf{CPair}} \llcorner M\lrcorner$.

Proof: By induction on the structure of the given canonical derivations. For the cases when $M = \mathsf{lam}\,(\lambda x{:}\mathsf{triv}.\ r)$, and $M = \mathsf{klam}\,(\overset{>}{\lambda} k{:}(\mathsf{triv} \rightarrow \mathsf{exp}).\ e)$ note that the ordered context Ω must be empty since no ordered variables can occur in the argument to an intuitionistic application. □

5 CPS Transform

We represent CPS transform with three basic types corresponding to the three judgements of the transform.

$\mathsf{cps\_r} : \mathsf{droot} \rightarrow \mathsf{root} \rightarrow \mathsf{type}.$ $\mathsf{cps\_e} : \mathsf{dexp} \rightarrow \mathsf{cpair} \rightarrow \mathsf{exp} \rightarrow \mathsf{type}.$ $\mathsf{cps\_t} : \mathsf{dtriv} \rightarrow \mathsf{triv} \rightarrow \mathsf{type}.$

We then use the following terms to construct representations of the CPS transform.

$$
\begin{array}{rcl}
\mathsf{cps\_root} & : & \Pi E{:}\mathsf{dexp}.\ \Pi E'{:}\mathsf{cpair} \twoheadrightarrow \mathsf{exp}.\\
& & (\Pi k{:}\mathsf{cpair}.\ \mathsf{cps\_e}\, E\, k\, (E' \,{}^{>} k)) \rightarrow \mathsf{cps\_r}\, (\mathsf{e2r}\, E)\, (\mathsf{klam}\, E').\\[1em]
\mathsf{cps\_triv} & : & \Pi T{:}\mathsf{dtriv}.\ \Pi P{:}\mathsf{cpair}.\ \Pi T'{:}\mathsf{triv}.\\
& & \mathsf{cps\_t}\, T\, T' \rightarrow \mathsf{cps\_e}\, (\mathsf{t2e}\, T)\, P\, (\mathsf{kapp}\,{}^{>} (\mathsf{nrml}\,{}^{>} P)\,{}^{>} T').\\[1em]
\mathsf{cps\_raise} & : & \Pi E{:}\mathsf{dexp}.\ \Pi E'{:}\mathsf{exp}.\ \Pi P{:}\mathsf{cpair}.\\
& & \mathsf{cps\_e}\, E\, (\mathsf{pair}\,{}^{>} \langle \mathsf{hnd}_0\,{}^{>} P\, ,\ \mathsf{hnd}_0\,{}^{>} P \rangle)\, E' \rightarrow\\
& & \mathsf{cps\_e}\, (\mathsf{raise}\, E)\, P\, E'.\\[1em]
\mathsf{cps\_app} & : & \Pi E_0{:}\mathsf{dexp}.\ \Pi E_1{:}\mathsf{dexp}.\ \Pi P{:}\mathsf{cpair}.\ \Pi E_1'{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{exp}.\\
& & \mathsf{cps\_e}\, E_0\, (\mathsf{pair}\,{}^{>} \langle \mathsf{vlam}\,{}^{>} E_1'\, ,\ \mathsf{hnd}_0\,{}^{>} P \rangle)\, E' \rightarrow\\
& & (\Pi v_0{:}\mathsf{triv}.\ \mathsf{cps\_e}\, E_1\, (\mathsf{pair}\,{}^{>} \langle \mathsf{vlam}\,{}^{>} \overset{>}{\lambda} v_1{:}\mathsf{triv}.\ \mathsf{app}\,{}^{>} P\,{}^{>} v_0\,{}^{>} v_1\, ,\ \mathsf{hnd}_1\,{}^{>} P\,{}^{>} v_0 \rangle)\, (E_1'\,{}^{>} v_0)) \rightarrow\\
& & \mathsf{cps\_e}\, (\mathsf{dapp}\, E_0\, E_1)\, P\, E'.\\[1em]
\mathsf{cps\_handle} & : & \Pi E_0{:}\mathsf{dexp}.\ \Pi E_1{:}\mathsf{dtriv} \rightarrow \mathsf{droot}.\ \Pi E_1'{:}\mathsf{dtriv} \rightarrow \mathsf{droot}.\ \Pi P{:}\mathsf{cpair}.\ \Pi E'{:}\mathsf{exp}.\\
& & \mathsf{cps\_e}\, E_0\, (\mathsf{pair}\,{}^{>} \langle \mathsf{nrml}\,{}^{>} P\, ,\ \mathsf{xlam}\,{}^{>} E_1' \rangle)\, E' \rightarrow\\
& & (\Pi x{:}\mathsf{dtriv}.\ \Pi x'{:}\mathsf{triv}.\ \mathsf{cps\_t}\, x\, x' \rightarrow \mathsf{cps\_e}\, (E_1\, x)\, P\, (E_1'\, x')) \rightarrow\\
& & \mathsf{cps\_e}\, (\mathsf{handle}\, E_0\, E_1)\, P\, E'.\\[1em]
\mathsf{cps\_lam} & : & \Pi R{:}\mathsf{dtriv} \rightarrow \mathsf{droot}.\ \Pi R'{:}\mathsf{triv} \rightarrow \mathsf{root}.\\
& & (\Pi x{:}\mathsf{dtriv}.\ \Pi x'{:}\mathsf{triv}.\ \mathsf{cps\_t}\, x\, x' \rightarrow \mathsf{cps\_r}\, (R\, x)\, (R'\, x')) \rightarrow\\
& & \mathsf{cps\_t}\, (\mathsf{dlam}\, R)\, (\mathsf{lam}\, R').
\end{array}
$$

We may now show the adequacy of above representation in two parts.

In the actual transformation we map variables x to themselves; in the representation we map each variable x from a DS term to a corresponding variable x' in a CPS term[3]. These variables and their relationship are captured in contexts

$$
\begin{array}{rcl}
\Gamma & = & x_1{:}\mathsf{dtriv} \ldots x_n{:}\mathsf{dtriv}\\
\Gamma' & = & x_1'{:}\mathsf{triv} \ldots x_n'{:}\mathsf{triv}\\
\Gamma_m & = & m_1{:}\mathsf{cps\_t}\, x_1\, x_1' \ldots m_n{:}\mathsf{cps\_t}\, x_n\, x_n'
\end{array}
$$

which always occur together in this manner. In addition we have contexts

$$
\begin{array}{rcl}
\Gamma_k & = & k_1{:}\mathsf{cpair} \ldots k_m{:}\mathsf{cpair}\\
\Gamma_v & = & v_1{:}\mathsf{triv} \ldots v_l{:}\mathsf{triv}
\end{array}
$$

which include all the continuation-pair identifiers k and temporary variables v which may occur in the continuation-pair p and CPS terms resulting from the translation. Note that ordering constraints are ignored during the translation, but will be nonetheless be satisfied by the resulting terms.

Theorem 5 (Representations are Canonical Forms)
Let $\Gamma^* = \Gamma, \Gamma', \Gamma_m, \Gamma_k, \Gamma_v$ be a context of the form explained above which contains all free variables occurring in the relevant judgement. Then

1. $\vdash r \xrightarrow{DR} r'$ *implies* $\exists M.\ \Gamma^*; \cdot \vdash M \Uparrow \mathsf{cps\_r} \ulcorner r \urcorner^R \ulcorner r' \urcorner$.

[3]This is accomplished by the cps_lam rule.

2. $\vdash e\ ;\ p \xrightarrow{DE} e'$ *and* $\Phi \models^{\mathbf{CPair}} p$ *implies* $\exists M.\ \Gamma^*, \ulcorner\Phi\urcorner; \cdot \vdash M \Uparrow \mathsf{cps\_e} \ulcorner e \urcorner^E \ulcorner p \urcorner \ulcorner e' \urcorner$

3. $\vdash t \xrightarrow{DT} t'$ *implies* $\exists M.\ \Gamma^*; \cdot \vdash M \Uparrow \mathsf{cps\_t} \ulcorner t \urcorner^T \ulcorner t' \urcorner$

Proof: By structural induction on the given derivation making use of Lemma 2. □

Theorem 6 (Canonical Forms are Representations) *Let* $\Gamma^* = \Gamma, \Gamma', \Gamma_m, \Gamma_k, \Gamma_v$ *a context of the form explained above and assume the types below are canonical.*

1. $\Gamma^*; \cdot \vdash M \Uparrow \mathsf{cps\_r}\, R\, R'$ *implies* $\vdash \llcorner R \lrcorner_R \xrightarrow{DR} \llcorner R' \lrcorner$.

2. $\Gamma^*; \cdot \vdash M \Uparrow \mathsf{cps\_e}\, E\, P\, E'$ *implies* $\vdash \llcorner E \lrcorner_E\ ;\ \llcorner P \lrcorner \xrightarrow{DE} \llcorner E' \lrcorner$.

3. $\Gamma^*; \cdot \vdash M \Uparrow \mathsf{cps\_t}\, T\, T'$ *implies* $\vdash \llcorner T \lrcorner_T \xrightarrow{DR} \llcorner T' \lrcorner$.

Proof: By structural induction on the given canonical derivation. □

The adequacy of our representation gives us a simple proof that the terms resulting from a CPS transformation satisfy the occurrence conditions of section 3.2.

Theorem 7 $\vdash r \xrightarrow{DR} r'$ *implies* $\models^{\mathbf{Root}} r'$.

Proof: By theorem 5 we know $\cdot; \cdot \vdash \ulcorner r' \urcorner \Uparrow \mathsf{root}$.
Then by theorem 4 we know $\models^{\mathbf{Root}} \llcorner \ulcorner r' \urcorner \lrcorner$.
Then we are done since $\llcorner \ulcorner r' \urcorner \lrcorner = r'$. □

The simplicity of the proof above may be surprising. It is so direct, because the work has been distributed to the proof of the adequacy theorems (which are clearly not trivial), combined with some deep properties of the logical framework such as the existence of canonical forms. This factoring of effort is typical in the use of logical frameworks.

6 Related Work

O'Hearn and Berdine have shown that the CPS transform with exceptions produces CPS terms which use their continuation-pair argument linearly [O'H00]. This work refines that analysis and shows that the immediate result of the transform actually uses the continuation-pair argument in an ordered fashion. However, our results are brittle in the sense that the ordering property is not preserved by arbitrary β reduction— β reducing underneath a lambda could result in a term which doesn't satisfy the ordering invariants.

In [PP00], Polakow and Pfenning show how OLF provides a convenient setting for reasoning about the CPS transform which doesn't treat exceptions. This work shows how those representation techniques easily extend to treat the CPS transform which removes exceptions.

7 Conclusion & Future Work

We formally proved the stackability and linearity of exception handlers with ML-style semantics using the help of an ordered logical framework (OLF) [PP00]. We transformed exceptions into continuation-passing-style (CPS) terms and formalizeed the exception properties as a judgement on the CPS terms. Then, rather than directly proving that the properties hold for terms, we proved our theorem for OLF representations of the CPS terms and transform. We used the correctness of our representations to transfer the results back to the actual CPS terms and transform. We further showed that the results in [DP95, DDP99] carry-over to the extended CPS transform which removes exceptions. Working with OLF representations allowed for a relatively simple proof in which we could directly use known properties of OLF terms (e.g. substitution properties) rather than re-proving similar properties for actual CPS terms satisfying our invariants.

We can also extend our analysis to cover evaluation of CPS terms. The invariants satisfied by CPS-transformed terms clearly suggest a stack-like evaluation mechanism. In fact, we can show (though space doesn't permit it in this paper) that a stack-like evaluation machine for CPS terms, which takes advantage of the ordering invariants, behaves the same as a regular evaluation machine which always uses substitution.

Our work can be seen as two-fold: it is a theoretical justification of existing compilers that use the stack mechanism to implement exceptions; and it demonstrates the value of a (ordered) logical framework as a conceptual tool in the theoretical study of programming languages. We conjecture that many systems with constrained resource access will have a natural representation in an ordered logical framework.

8 Acknowledgements

We would like to acknowledge insightful discussions with Peter O'Hearn and Frank Pfenning.

References

[App97] Andrew W. Appel. *Modern Compiler Implementation in ML/C/Java: Basic Techniques.* Cambridge University Press, 1997.

[CP99] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information and Computation*, 1999. To appear in the special issue with invited papers from LICS'96, E. Clarke, editor.

[DDP99] Olivier Danvy, Belmina Dzafic, and Frank Pfenning. On proving syntactic properties of cps programs. In *Third International Workshop on Higher Order Operational Techniques in Semantics (HOOTS'99)*, Paris, France, September 1999.

[DF92] Olivier Danvy and Andrzej Filinski. Representing control: a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, December 1992.

[DP95] Olivier Danvy and Frank Pfenning. The occurrence of continuation parameters in CPS terms. Technical Report CMU-CS-95-121, Department of Computer Science, Carnegie Mellon University, February 1995.

[Fel87] Matthias Felleisen. *The Calculi of λ-v-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages.* PhD thesis, Department of Computer Science, Indiana University, Bloomington, Indiana, August 1987.

[HD97] John Hatcliff and Olivier Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 7(3):303–320, 1997.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[KYD98] Jungtaek Kim, Kwangkeun Yi, and Olivier Danvy. Assessing the overhead of ml exceptions by selective cps transformation. In *The Proceedings of the ACM SIGPLAN Workshop on ML*, pages 103–114, September 1998.

[LDG$^+$00] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The objective caml system (relase 3.00), documentation and user's manual. http://caml.inria.fr/ocaml/htmlman/index.html, 2000.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised).* MIT Press, 1997.

[O'H00] P.W. O'Hearn. Personal communication with author. May 2000.

[Pfe96] Frank Pfenning. The practice of logical frameworks. In Hélène Kirchner, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming*, pages 119–134, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1059. Invited talk.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, September 1981.

[PP99a] Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In J.-Y. Girard, editor, *Proceedings of the Fourth International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 295–309, l'Aquila, Italy, April 1999. Springer-Verlag LNCS 1581.

[PP99b] Jeff Polakow and Frank Pfenning. Relating natural deduction and sequent calculus for intuitionistic non-commutative linear logic. In Andre Scedrov and Achim Jung, editors, *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, pages 311–328, New Orleans, Louisiana, April 1999. Electronic Notes in Theoretical Computer Science, Volume 20.

[PP00] Jeff Polakow and Frank Pfenning. Properties of terms in continuation passing style in an ordered logical framework. In *Workshop on Logical Frameworks and Meta-Languages (LFM 2000)*, Santa Barbara, California, June 2000.

[Ste78] Guy L. Steele Jr. Rabbit: A compiler for Scheme. Technical Report AI-TR-474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978.