Mechanically Verifying Correctness of CPS Compilation

Ye Henry Tian

School of Computer Science
McGill University,
Montreal, Quebec H3A 2A7, Canada,
Email: ytian8@cs.mcgill.ca

Abstract

In this paper, we study the formalization of one-pass call-by-value CPS compilation using higher-order abstract syntax. In particular, we verify mechanically that the source program and the CPS-transformed program have the same observable behavior. A key advantage of this approach is that it avoids any administrative redexes thereby simplifying the proofs about CPS-translations. The CPS translation together with its correctness proof is implemented and mechanically verified in the logical framework Twelf.

Keywords: Program transformation, correctness proofs, higher-order abstract syntax, logical framework

1 Introduction

Compilation is a program transformation from a source to a target language, each with a well-defined syntax and semantics. The problem is then to prove that the source and target program have the same observable behavior at execution time. Establishing the correctness of a compiler has been important from the beginning of compiler studies. However this is often found difficult since the semantics of a realistic programming language is often difficult to formalize and the execution models of source and target languages are often very different. Further more a (minor) change to the compiler (e.g. implementation of a new optimization strategy) which is likely to occur during the development cycle of a compiler, may require a renewed effort to reconstruct the correctness proof of the compiler. Mechanical verification can increase the confidence in the correctness proof, since they are often tedious and it is easy to make mistakes, especially if a language still evolves.

Continuation-passing style (CPS) is a program notation that makes every aspect of control flow and data flow explicit (Appel 1992). The original work of CPS study due to Plotkin goes back to the mid-70's (Plotkin 1975). In his work, he provided the first formalization of the CPS transformation which is first-order. However, this gives rise to annoying *administrative redexes* which are solely due to the CPS transformation and not corresponding to an actual reduction step in the original program (Danvy & Filinski 1992, Danvy 1991). These redexes are annoying because they interfere both with proving the correctness and properties of a CPS transformation (Plotkin 1975, Danvy & Nielsenand 2001) and with using it in a compiler (Guy L. Steele 1978).

Different studies of different versions of CPS transformations have been done since Plotkin's original work. To simplify the correctness proof of CPS transformation, and to simplify the reasoning of proving properties (CPS transformations preserve types) of CPS-transformed programs, O. Danvy and L. R. Nielsen presented a new first-order one-pass CPS transformation, which is compositional (Danvy & Nielsen 2002). They later presented a higher-order colon translation (Danvy & Nielsenand 2001), which links higher-order CPS transformation with Plotkin's original proof technique. They extended Plotkin's original first-order colon translation to a higher-order version in order to prove the correctness of the higher-order CPS transformation. However, they still need a colon translation to handle administrative redexes. Olivier Danvy, Belmina Dzafic and Frank Pfenning showed their approach of formalizing a higher-order CPS transformation using logical relations in order to prove a syntactic property of the CPS program (Danvy, Dzafic & Pfenning 1999). They proved the proper occurrence of the formal parameters of the continuations. They formalized their proofs in Elf, but they did not show how to formalize the correctness proof of the CPS transformation. Hongwei Xi and Carsten Schürmann showed that typing derivations can be CPS-transformed, so that we can lift the CPS transformation from the level of expressions to the level of type derivations (Xi & Schürmann 2001). They studied a call-by-value CPS transformation for a core subset of the DML language. They proved they can lift the CPS transformation from the level of expressions to the level of type derivations, and they formalized the languages and transformation in LF. However, they did not prove nor verify the correctness of their CPS transformation for DML types. Yasuhiko Minamide and Koji Okuma have verified several versions of CPS transformation in the Isabelle/HOL theorem prover (Minamide & Okuma 2003). Since they chose to use first-order abstract syntax with variable names for their formalization, they had the problems of bound variables renaming and they needed to implement their own "substitution" programs.

None of the related work discussed above showed how to use higher-order abstract syntax to formalize and verify CPS translation, which is our goal here. In this paper, we presents a higher-order setting of CPS transformation as a rewriting system that directly produces a CPS program without administrative redexes. We also show how to formalize CPS translation using higher-order abstract syntax and verify its correctness proof using logical framework. This approach has several advantages compared to the ones above: 1) No administrative redexes. 2) Proof is a structural induction proof. 3) The formal-

ization does not have the problems of fresh variables and α-equivalence.

The rest of this paper is organized as follows. Section 2 introduces the direct-style source language and the continuation-passing style target language we consider in our CPS translation and their operational semantics. The one-pass higher-order CPS transformation is presented at the end of this section. Section 3 shows how we prove the correctness of our CPS transformation by proving the "Soundness" and "Completeness" theorems. In Section 4, we present the formalization of the CPS translation and the correctness proof using higher-order abstract syntax in a logical framework. Section 5 concludes.

2 Languages and CPS Transformation

2.1 Operational semantics for Mini-ML

We will consider a fragment of a functional language as our source language. It serves as the jumping-off point for much of the studies of programming language concepts. In our CPS translation, we consider the Mini-ML expressions as the *direct-style* programs. We only consider a small subset of the language in this paper, the syntax of the Mini-ML language is defined as follows:

$$\begin{array}{rl} \text{Expressions } e ::= & \mathtt{zero} \mid (\mathtt{succ}\ e) \\ & \mid (\mathtt{case}\ e_1\ \mathtt{of}\ \mathtt{zero} \Rightarrow e_2 \mid \mathtt{succ}\ x \Rightarrow e_3) \\ & \mid \mathtt{pair}(e_1,\ e_2) \mid \mathtt{fst}\ e \mid \mathtt{snd}\ e \\ & \mid \mathtt{app}(e_1,\ e_2) \mid \mathtt{lam}\ x.\ e \\ & \mid \mathtt{vl}(v) \end{array}$$

$$\text{Values } v ::= x \mid \mathtt{zero}^* \mid \mathtt{succ}^*\ v \mid \mathtt{pair}^*(v_1,\ v_2) \mid \mathtt{lam}^*\ x.\ e$$

Most of these constructs should be familiar from functional programming languages such as ML: `zero` stands for zero, (`succ` e) stands for the successor of e. A `case` expression chooses a branch based on whether the value of the first argument is zero or non-zero. Abstraction, `lam` $x.\ e$, forms functional expressions. Note that only value variables exist. The term $\mathtt{vl}(v)$ represents a coercion of a Mini-ML value to a Mini-ML expression. We again distinguish values from expressions since this will later simplify the formalization of the CPS translation.

$$\frac{}{\mathtt{vl}(v) \hookrightarrow v}\ ev\_\mathtt{vl}$$

$$\frac{}{\mathtt{zero} \hookrightarrow \mathtt{zero}^*}\ ev\_z \qquad \frac{e \hookrightarrow v}{\mathtt{succ}\ e \hookrightarrow \mathtt{succ}^*\ v}\ ev\_s$$

$$\frac{e_1 \hookrightarrow z \quad e_2 \hookrightarrow v}{(\mathtt{case}\ e_1\ \mathtt{of}\ z \Rightarrow e_2 \mid \mathtt{succ}\ x \Rightarrow e_3) \hookrightarrow v}\ ev\_case\_z$$

$$\frac{e_1 \hookrightarrow \mathtt{succ}^*\ v_1' \quad [v_1'/x]e_3 \hookrightarrow v}{(\mathtt{case}\ e_1\ \mathtt{of}\ \mathtt{zero} \Rightarrow e_2 \mid \mathtt{succ}\ x \Rightarrow e_3) \hookrightarrow v}\ ev\_case\_s$$

$$\frac{e_1 \hookrightarrow v_1 \quad e_2 \hookrightarrow v_2}{\mathtt{pair}(e_1,\ e_2) \hookrightarrow \mathtt{pair}^*(v_1,\ v_2)}\ ev\_pair$$

$$\frac{e \hookrightarrow \mathtt{pair}^*(v_1,\ v_2)}{\mathtt{fst}\ e \hookrightarrow v_1}\ ev\_fst \qquad \frac{e \hookrightarrow \mathtt{pair}^*(v_1,\ v_2)}{\mathtt{snd}\ e \hookrightarrow v_2}\ ev\_snd$$

$$\frac{}{\mathtt{lam}\ x.\ e \hookrightarrow \mathtt{lam}^*\ x.\ e}\ ev\_lam$$

$$\frac{e_1 \hookrightarrow \mathtt{lam}^*\ x.\ e' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e' \hookrightarrow v}{\mathtt{app}(e_1,\ e_2) \hookrightarrow v}\ ev\_app$$

Figure 1: The Mini-ML big-step evaluation semantics.

Evaluation rules of the Mini-ML expressions are shown in Figure 1. They are given via a big-step operational semantics and are described by the judgment $e \hookrightarrow v$ meaning the expression e evaluates to a value v. For example, to evaluate an application $\mathtt{app}(e_1,\ e_2)$, we need to evaluate e_1 to some value $\mathtt{lam}^*\ x.\ e'$, e_2 to some value v_2, and $[v_2/x]e'$ to the final value of the application. Note, the order of evaluation of these premises is left unspecified. The other rules are straightforward.

We will refer to the Mini-ML language as direct style to distinguish it from the continuation-passing style (CPS) calculus introduced below.

2.2 The continuation-passing style language

In this section, we define the target language of the CPS compilation, which describes programs in continuation-passing style.

CPS terms satisfy three properties (Plotkin 1975):

- Indifference: Evaluation order independent.
- Simulation: CPS encodes the evaluation order.
- Translation equational correspondence between direct-style and CPS-style.

The syntax of the CPS language we use to represent the continuation-passing style programs is defined as follows:

$$\begin{array}{rl} \text{Expressions } E ::= & \mathsf{App}(V_1,\ V_2,\ k) \mid \mathsf{Fst}(V,\ k) \mid \mathsf{Snd}(V,\ k) \\ & \mid (\mathsf{Case}\ \ V\ \mathsf{of}\ \mathsf{Zero} \Rightarrow E_2 \mid (\mathsf{Succ}\ x) \Rightarrow E_3) \\ & \mid \mathsf{vl}(V) \end{array}$$

$$\text{Values } V ::= x \mid \mathsf{Zero} \mid \mathsf{Succ}\ V \mid \mathsf{Pair}(V_1,\ V_2) \mid \mathsf{Lam}\ (x, k).E$$

$$\text{Continuations } k ::= \lambda x.E$$

Evaluation contexts are modeled via CPS continuations. A CPS continuation is represented as a meta-level function $\lambda x.E$ which essentially describes a CPS expression which has a hole x, that can only be filled by a CPS value. The initial or empty evaluation context is represented by $\lambda x.\mathsf{vl}(V)$. Substitution into a hole is modeled via meta-level application $((\lambda x.E)\ V)$ which beta-reduces to $[V/x]E$. An important property of CPS continuations is that they are in fact linear functions, i.e. the hole x in the CPS expression E occurs only once (Appel 1992, Danvy et al. 1999). We will not prove this property but it will be maintained as part of the invariants in this description.

The CPS expressions keep track of its evaluation context in which they will be executed using an extra parameter k. In the definition above, we separate cleanly CPS values from CPS expressions and $\mathsf{vl}(V)$ denotes the coercion of a CPS value to a CPS expression. Note that the arguments to CPS expressions must be CPS values and cannot be arbitrary subexpressions.

Figure 2 shows the operational semantics of the CPS language. The single-step reductions for the CPS expressions are defined first. The multi-step reduction relation $\longmapsto^*$ is the reflexive, transitive closure of single-step reduction $\longmapsto$. Deductions of the judgment $E \longmapsto^* E'$ have a very simple form: They all consist of a sequence of single steps terminated by an application of the reflexivity rule. We will follow standard practice and use a linear notation for sequences of steps:

$$E_1 \longmapsto E_2 \longmapsto E_3 \longmapsto \ldots \longmapsto E_n$$

Similarly, we will mix multi-step and single-step transitions in sequences with the obvious meaning. Using the rules above we can reduce CPS expressions to CPS values (in the form of $\mathsf{vl}(V)$). There are two important properties of this multi-reduction system.

The CPS language single-step reductions

$$\frac{}{(\mathsf{Case}\ \ \mathsf{Zero\ of\ Zero}\ \Rightarrow E_2 \mid \mathsf{Succ}\ x \Rightarrow E_3) \longmapsto E_2}\ red\_case\_z$$

$$\frac{}{(\mathsf{Case}\ \ (\mathsf{Succ}\ V)\ \mathsf{of\ Zero}\ \Rightarrow E_2 \mid \mathsf{Succ}\ x \Rightarrow E_3) \longmapsto [V/x]E_3}\ red\_case\_s$$

$$\frac{}{\mathsf{Fst}(\mathsf{Pair}(V_1,\ V_2),\ k) \longmapsto (k\ V_1)}\ red\_fst$$

$$\frac{}{\mathsf{Snd}(\mathsf{Pair}(V_1,\ V_2),\ k) \longmapsto (k\ V_2)}\ red\_snd$$

$$\frac{}{\mathsf{App}((\mathsf{Lam}\ (x,k').\ E),\ V,\ k) \longmapsto [V/x][k/k']E}\ red\_app$$

The CPS language multi-step reductions

$$\frac{}{E \longmapsto^* E}\ red\_reflexive$$

$$\frac{E \longmapsto E_1 \quad E_1 \longmapsto^* E_2}{E \longmapsto^* E_2}\ red\_transitive$$

Figure 2: The CPS language operational semantics.

Theorem 1 (Uniqueness).
If $E \longmapsto^* \mathsf{vl}(V_1)$ *and* $E \longmapsto^* \mathsf{vl}(V_2)$, *then* $V_1 = V_2$.

Proof. Proof by structural induction on the reduction rules. □

Theorem 2 (Termination).
For every CPS expression E *there is some CPS value* V *such that* $E \longmapsto^* \mathsf{vl}(V)$.

Proof. Note that no variable can be bound in more than one place within a continuation expression, and no variable can be mentioned outside its syntactic scope. So we can observe that each reduction step reduces the size of the term and that size is a termination measure because the usual order on the natural numbers is well founded. □

2.3 The one-pass CPS transformation

In this section, we define two CPS-translations, one for Mini-ML values and one for Mini-ML expressions. The translations are mutual recursive. While the CPS-translation for values is essentially closed and independent of the CPS continuations, the CPS-translation for expressions must take into account the current evaluation state, which is captured by the CPS continuations. We are also very explicit about free variables which will all be bound in a context Γ, which is defined as follows:

$$\Gamma ::= . \mid \Gamma, x \mid \Gamma, k$$

The context Γ can be either empty or contain Mini-ML and CPS variables as well as continuation variable k.

The CPS transformation we present here is similar to an optimized version of CPS transformation by Danvy and Nielsen (Danvy & Nielsen 2002). The transformation operates in one pass and is both compositional and higher-order. Because it operates in one pass, it directly yields compact CPS programs that are comparable to what one would write by hand. Because it is compositional, it allows proofs by structural induction. Because it is higher-order, we avoid administrative redexes which in turn simplifies the proof about CPS-translation. We give the translation as an inference system, which may be slightly unusual but emphasizes the behavior of the CPS translation.

For the purpose of our formalization, Figure 3 expresses the one-pass CPS transformation as inference rules. It uses two judgments. A Mini-ML value v is transformed into a CPS value V whenever the judgment

$$\Gamma \vdash v \overset{v}{\sim} V$$

is satisfied. Given a (higher-order) continuation k, a Mini-ML expression e is transformed into a CPS expression E whenever the judgment

$$\Gamma \vdash (e,\ k) \overset{e}{\sim} E$$

is satisfied.

These judgments can be interpreted operationally by assuming that v, e and k are given and V and E are to be constructed by building a CPS-translation derivation in a bottom-up fashion. Since the variables in the Mini-ML languages and the CPS language can only contain values, we say that a Mini-ML variable x is CPS-transformed into a CPS variable x, by $\Gamma \vdash x \overset{v}{\sim} x$, if x is in the context Γ. All the free variables are captured by the context Γ during the CPS-translation. The context Γ can actually contains both Mini-ML and CPS variables (one can think of that Γ contains implicit CPS-translation from Mini-ML variables to CPS variables). The free variables bound in Γ may occur in e, k or E. Note that in order to translate $\texttt{lam}^*\ x.\ e$ we need to recursively translate the function body e.

Transforming a Mini-ML expression into its corresponding CPS expression needs to represent continuation contexts as λ-abstractions. A Mini-ML expression e is transformed with a continuation context k, it ranges over meta-level functions, which map CPS values to CPS expressions. The result of transforming an expression e into CPS expression E in an empty context is given by

$$\Gamma \vdash (e,\ (\lambda x.\mathsf{vl}(x))) \overset{e}{\sim} E$$

.

3 Correctness of the one-pass CPS transformation

Plotkin proved three properties of the CPS transformation: Indifference, Simulation, and Translation (Plotkin 1975). We prove the Simulation property here, since it formalizes the correctness of the call-by-value CPS transformation.

To prove correctness, we need to show the correspondence between the high-level language and the low-level abstract machine. In other words, we can translate programs written in the high-level language into programs which run on the abstract machine. Moreover, the source program will have the same observable behavior as the target program. The proof is constructive and constitutes a program which translates derivations in the source language into derivations of the target language and vice versa.

Theorem 3 (Correctness).
Soundness:
If $\Gamma \vdash (e,\ (\lambda x.\mathsf{vl}(x))) \overset{e}{\sim} E'$ *and* $E' \longmapsto^* \mathsf{vl}(V)$ *then* $e \hookrightarrow v$ *and* $\Gamma \vdash v \overset{v}{\sim} V$.

Completeness:
If $e \hookrightarrow v$ *and* $\Gamma \vdash (e,\ \lambda x.\mathsf{vl}(x)) \overset{e}{\sim} E'$ *then* $\Gamma \vdash v \overset{v}{\sim} V'$ *and* $E' \longmapsto^* \mathsf{vl}(V')$.

The theorem shows that the source program (Mini-ML program) and the target program (CPS-transformed program) will result in the same value. We will prove the correctness of our CPS-translation by proving the soundness and completeness of the translation.

CPS-translation for Mini-ML values

$$\frac{x \in \Gamma}{\Gamma \vdash x \overset{v}{\sim} x}\ cps\_var \qquad \frac{}{\Gamma \vdash \texttt{zero}^* \overset{v}{\sim} \mathsf{Zero}}\ cps\_\texttt{zero}^*$$

$$\frac{\Gamma \vdash v \overset{v}{\sim} V}{\Gamma \vdash (\texttt{succ}^*\ v) \overset{v}{\sim} (\mathsf{Succ}\ V)}\ cps\_\texttt{succ}^* \qquad \frac{\Gamma \vdash v_1 \overset{v}{\sim} V_1 \quad \Gamma \vdash v_2 \overset{v}{\sim} V_2}{\Gamma \vdash \texttt{pair}^*(v_1,\ v_2) \overset{v}{\sim} \mathsf{Pair}(V_1,\ V_2)}\ cps\_\texttt{pair}^*$$

$$\frac{\Gamma, x, k' \vdash (e,\ k') \overset{e}{\sim} E' \quad \text{where } x \text{ and } k' \text{ are fresh}}{\Gamma \vdash (\texttt{lam}^*\ x.\ e) \overset{v}{\sim} \mathsf{Lam}\ (x, k').\ E'}\ cps\_\texttt{lam}^*$$

CPS-translation for Mini-ML expressions

$$\frac{\Gamma \vdash v \overset{v}{\sim} V}{\Gamma \vdash (\texttt{vl}(v),\ k) \overset{e}{\sim} (k\ V)}\ cps\_\texttt{vl} \qquad \frac{}{\Gamma \vdash (\texttt{zero},\ k) \overset{e}{\sim} (k\ \mathsf{Zero})}\ cps\_\texttt{zero}$$

$$\frac{\Gamma \vdash (e,\ (\lambda x.k\ (\mathsf{Succ}\ x))) \overset{e}{\sim} E'}{\Gamma \vdash ((\texttt{succ}\ e),\ k) \overset{e}{\sim} E'}\ cps\_\texttt{succ}$$

$$\frac{\begin{array}{l}\Gamma \vdash (e_2,\ k) \overset{e}{\sim} E_2 \\ \Gamma, x \vdash (e_3,\ k) \overset{e}{\sim} E_3 \\ \Gamma \vdash (e_1,\ (\lambda x_1.\mathsf{Case}\ x_1 \mathsf{of}\ \mathsf{z} \Rightarrow E_2 \mid (\mathsf{Succ}\ \mathsf{x}) \Rightarrow E_3)) \overset{e}{\sim} E'\end{array}}{\Gamma \vdash ((\texttt{case}\ e_1 \texttt{of}\ \texttt{zero} \Rightarrow e_2 \mid (\texttt{succ}\ \texttt{x}) \Rightarrow e_3),\ k) \overset{e}{\sim} E'}\ cps\_\texttt{case}$$

$$\frac{\begin{array}{l}\Gamma, x_1 \vdash (e_2,\ (\lambda x_2.k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} E_2 \\ \Gamma \vdash (e_1,\ (\lambda x_1.E_2)) \overset{e}{\sim} E'\end{array}}{\Gamma \vdash (\texttt{pair}(e_1,\ e_2),\ k) \overset{e}{\sim} E'}\ cps\_\texttt{pair}$$

$$\frac{\Gamma \vdash (e,\ (\lambda x_1.\mathsf{Fst}(x_1,\ k))) \overset{e}{\sim} E'}{\Gamma \vdash (\texttt{fst}\ e,\ k) \overset{e}{\sim} E'}\ cps\_\texttt{fst} \qquad \frac{\Gamma \vdash (e,\ (\lambda x_2.\mathsf{Snd}(x_2,\ k))) \overset{e}{\sim} E'}{\Gamma \vdash (\texttt{snd}\ e,\ k) \overset{e}{\sim} E'}\ cps\_\texttt{snd}$$

$$\frac{\Gamma, x, k' \vdash (e,\ k') \overset{e}{\sim} E' \quad \text{where } x \text{ and } k' \text{ are fresh}}{\Gamma \vdash (\texttt{lam}\ x.\ e,\ k) \overset{e}{\sim} (k\ (\mathsf{Lam}\ (x, k').\ E'))}\ cps\_\texttt{lam}$$

$$\frac{\begin{array}{l}\Gamma, x_1 \vdash (e_2,\ (\lambda x_2.\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} E_2 \\ \Gamma \vdash (e_1,\ (\lambda x_1.\ E_2)) \overset{e}{\sim} E'\end{array}}{\Gamma \vdash (\texttt{app}(e_1,\ e_2),\ k) \overset{e}{\sim} E'}\ cps\_\texttt{app}$$

Figure 3: The one-pass CPS transformation formulated as inference rules.

To show that the abstract machine works correctly, we prove that all reduction sequences can be translated into evaluation derivations. To be more precise, we need to show that for all reduction sequences on the abstract machine starting with the empty context which evaluate some expression E ($\Gamma \vdash (e,\ (\lambda x.\mathsf{vl}(x))) \overset{e}{\sim} E$) to some CPS value V in multiple steps, there exists an evaluation in the Mini-ML operational semantics s.t. expression e evaluates to value v and $\Gamma \vdash v \overset{v}{\sim} V$.

This guarantees that the translation is sound. However, we will not be able to prove this statement directly, since the evaluation context k will change, which will prevent the application of the induction hypothesis in the proof. Therefore we will prove the following generalized statement: if we start in an arbitrary evaluation context k, with the state $\Gamma \vdash (e,\ k) \overset{e}{\sim} E$ and a reduction sequence $E \longmapsto^* \mathsf{vl}(W)$ then there exists an intermediate state $(k\ V)$ such that $e \hookrightarrow v$ in the Mini-ML semantics and $\Gamma \vdash v \overset{v}{\sim} V$ and $(k\ V) \longmapsto^* \mathsf{vl}(W)$. The lemma states that a complete computation with an appropriate initial state can be translated into an evaluation followed by another complete computation. This proof follows similar proofs about the correctness of compilers which are discussed in more detail in (Pfenning 2001).

3.1 Substitution lemmas

Before we proceed to prove this soundness lemma, we will first show that CPS-translation is preserved across substitution. This is an important property needed in the proof for the Soundness and Completeness lemmas.

Lemma 1 (Substitution lemma).

1. *For all Mini−ML value v and v'. If $\Gamma \vdash v' \overset{v}{\sim} V'$ and $\Gamma, x, \Gamma' \vdash v \overset{v}{\sim} V$ then $\Gamma, \Gamma' \vdash [v'/x]v \overset{v}{\sim} [V'/x]V$.*
2. *For all Mini−ML expressions e and values v'. If $\Gamma \vdash v' \overset{v}{\sim} V'$ and $\Gamma, x, \Gamma' \vdash (e,\ k) \overset{e}{\sim} E$ then $\Gamma, \Gamma' \vdash ([v'/x]e,\ [V'/x]k) \overset{e}{\sim} [V'/x]E$.*

Proof. By mutual induction on v and e. We only show a few cases in the proof in detail; the remaining ones follow the same pattern.

Case: $v = x$

$\Gamma, x, \Gamma' \vdash x \overset{v}{\sim} V$ By ass.
$V = x$ By *cps_var*
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, \Gamma' \vdash [v'/x]x \overset{v}{\sim} [V'/x]x$ By subst. def.

Case: $v = \mathtt{zero}^*$

$\Gamma, x, \Gamma' \vdash \mathtt{zero}^* \overset{v}{\sim} V$ By ass.
$V = \mathsf{Zero}$ By *cps_*$\mathtt{zero}^*$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, \Gamma' \vdash [v'/x]\mathtt{zero}^* \overset{v}{\sim} [V'/x]\mathsf{Zero}$ By subst. def.

Case: $v = \mathtt{pair}^*(v_1,\ v_2)$

$\Gamma, x, \Gamma' \vdash \mathtt{pair}^*(v_1,\ v_2) \overset{v}{\sim} V$ By ass.
$V = \mathsf{Pair}(V_1,\ V_2)$,
$\Gamma, x, \Gamma' \vdash v_1 \overset{v}{\sim} V_1$ and $\Gamma, x, \Gamma' \vdash v_2 \overset{v}{\sim} V_2$ By *cps_*$\mathtt{pair}^*$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, \Gamma' \vdash [v'/x]v_1 \overset{v}{\sim} [V'/x]V_1$ and
$\Gamma, \Gamma' \vdash [v'/x]v_2 \overset{v}{\sim} [V'/x]V_2$ By I.H.
$\Gamma, \Gamma' \vdash \mathtt{pair}^*([v'/x]v_1,\ [v'/x]v_2) \overset{v}{\sim} \mathsf{Pair}([V'/x]V_1,\ [V'/x]V_2)$
By *cps_*$\mathtt{pair}^*$
$\Gamma, \Gamma' \vdash [v'/x]\mathtt{pair}^*(v_1,\ v_2) \overset{v}{\sim} [V'/x]\mathsf{Pair}(V_1,\ V_2)$
By subst. def.

Case: $v = \mathtt{lam}^*\ y.\ e$ *(where* $y \neq x$*)*

$\Gamma, x, \Gamma' \vdash \mathtt{lam}^*\ y.\ e \overset{v}{\sim} V$ By ass.
$V = \mathsf{Lam}\ (y, k').\ E'$ and
$\Gamma, y, k', x, \Gamma' \vdash (e,\ k') \overset{e}{\sim} E'$ By *cps_*$\mathtt{lam}^*$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, y, k', \Gamma' \vdash ([v'/x]e,\ k') \overset{e}{\sim} [V'/x]E'$
By I.H. on subst. lemma 2
$\Gamma, \Gamma' \vdash \mathtt{lam}^*\ y.\ [v'/x]e \overset{v}{\sim} \mathsf{Lam}\ (y, k').\ [V'/x]E'$ By *cps_*$\mathtt{lam}^*$
$\Gamma, \Gamma' \vdash [v'/x]\mathtt{lam}^*\ y.\ e \overset{v}{\sim} [V'/x]\mathsf{Lam}\ (y, k').\ E'$ By subst. def.

Case: $e = \mathtt{vl}(v)$

$\Gamma, x, \Gamma' \vdash (\mathtt{vl}(v),\ k) \overset{e}{\sim} E$ By ass.
$E = (k\ V)$ and $\Gamma, x, \Gamma' \vdash v \overset{v}{\sim} V$ By *cps_*$\mathtt{vl}$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, \Gamma' \vdash [v'/x]v \overset{v}{\sim} [V'/x]V$ By I.H. on subst. lemma 1
$\Gamma, \Gamma' \vdash (\mathtt{vl}([v'/x]v),\ [V'/x]k) \overset{e}{\sim} ([V'x]k\ [V'/x]V)$ By *cps_*$\mathtt{vl}$
$\Gamma, \Gamma' \vdash ([v/x]\mathtt{vl}(v'),\ [V'/x]k) \overset{e}{\sim} [V'/x](k\ V)$ By subst. def.

Case: $e = \mathtt{zero}$

$\Gamma, x, \Gamma' \vdash (\mathtt{zero},\ k) \overset{e}{\sim} E$ By ass.
$E = (k\ \mathsf{Zero})$ By *cps_*$\mathtt{zero}$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, \Gamma' \vdash ([v'/x]\mathtt{zero},\ [V'/x]k) \overset{e}{\sim} ([V'/x]k\ \mathsf{Zero})$
By subst. def. and *cps_*$\mathtt{zero}$
$\Gamma, \Gamma' \vdash ([v'/x]\mathtt{zero},\ [V'/x]k) \overset{e}{\sim} [V'/x](k\ \mathsf{Zero})$ By subst. def.

Case: $e = \mathtt{pair}(e_1,\ e_2)$

$\Gamma, x, \Gamma' \vdash (\mathtt{pair}(e_1,\ e_2),\ k) \overset{e}{\sim} E'$ By ass.
$\Gamma, x_1, x, \Gamma' \vdash (e_2,\ (\lambda x_2.k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} E_2$ and
$\Gamma, x, \Gamma' \vdash (e_1,\ (\lambda x_1.E_2)) \overset{e}{\sim} E'$ By *cps_*$\mathtt{pair}$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, x_1, \Gamma' \vdash ([v'/x]e_2,\ [V'/x](\lambda x_2.k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} [V'/x]E_2$
By I.H.
$\Gamma, x_1, \Gamma' \vdash ([v'/x]e_2,\ (\lambda x_2.[V'/x]k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} [V'/x]E_2$
By subst. def.
$\Gamma, \Gamma' \vdash ([v'/x]e_1,\ [V'/x](\lambda x_1.E_2)) \overset{e}{\sim} [V'/x]E'$ By I.H.
$\Gamma, \Gamma' \vdash ([v'/x]e_1,\ (\lambda x_1.[V'/x]E_2)) \overset{e}{\sim} [V'/x]E'$ By subst. def.
$\Gamma, \Gamma' \vdash (\mathtt{pair}([v'/x]e_1,\ [v'/x]e_2),\ k) \overset{e}{\sim} [V'/x]E'$ By *cps_*$\mathtt{pair}$
$\Gamma, \Gamma' \vdash ([v'/x]\mathtt{pair}(e_1,\ e_2),\ k) \overset{e}{\sim} [V'/x]E'$ By subst. def.

Case: $e = \mathtt{app}(e_1,\ e_2)$

$\Gamma, x, \Gamma' \vdash (\mathtt{app}(e_1,\ e_2),\ k) \overset{e}{\sim} E'$ By ass.
$\Gamma, x_1, x, \Gamma' \vdash (e_2,\ (\lambda x_2.\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} E_2$ and
$\Gamma, x, \Gamma' \vdash (e_1,\ (\lambda x_1.\ E_2)) \overset{e}{\sim} E'$ By *cps_*$\mathtt{app}$
$\Gamma \vdash v' \overset{v}{\sim} V'$ By ass.
$\Gamma, x_1, \Gamma' \vdash ([v'/x]e_2,\ [V'/x](\lambda x_2.\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} [V'/x]E_2$
By I.H.
$\Gamma, x_1, \Gamma' \vdash ([v'/x]e_2,\ (\lambda x_2.\mathsf{App}(x_1,\ x_2,\ [V'/x]k))) \overset{e}{\sim} [V'/x]E_2$
By subst. def.
$\Gamma, \Gamma' \vdash ([v'/x]e_1,\ [V'/x](\lambda x_1.\ E_2)) \overset{e}{\sim} [V'/x]E'$ By I.H.
$\Gamma, \Gamma' \vdash ([v'/x]e_1,\ (\lambda x_1.\ [V'/x]E_2)) \overset{e}{\sim} [V'/x]E'$ By subst. def.
$\Gamma, \Gamma' \vdash (\mathtt{app}([v'/x]e_1,\ [v'/x]e_2),\ [V'/x]k) \overset{e}{\sim} [V'/x]E'$
By *cps_*$\mathtt{app}$
$\Gamma, \Gamma' \vdash ([v'/x]\mathtt{app}(e_1,\ e_2),\ [V'/x]k) \overset{e}{\sim} [V'/x]E'$ By subst. def.

□

3.2 Soundness

Lemma 2 (Soundness).
If $\mathcal{C} : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$ and $\mathcal{S} : E \longmapsto^* \mathsf{vl}(W)$ then there exist derivations $\mathcal{D} : e \hookrightarrow v$, $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$ and a rest computation $\mathcal{S}' : (k\ V) \longmapsto^* \mathsf{vl}(W)$, where $\mathcal{S}'$ is a subsequence of $\mathcal{S}$.

Proof. By structural induction on the derivation $\mathcal{C} : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$ and the CPS-reductions $\mathcal{S} : E \longmapsto^* \mathsf{vl}(W)$. We can apply the induction hypothesis if either the CPS-translation is applied to a subderivation of $\mathcal{C} : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$ or we have a shorter sequence of CPS-reductions $\mathcal{S}'$ where $\mathcal{S}' < \mathcal{S}$. We show most cases in the proof in detail; the remaining ones follow the same pattern. Throughout the proof we consider CPS expressions modulo $\overset{\beta}{=}$.

Case: $\mathcal{C}$ =

$$\dfrac{\begin{array}{l}\overset{\mathcal{C}_2}{\Gamma, x_1 \vdash (e_2, \underset{\mathcal{C}_1}{(\lambda x_2.k\ (\mathsf{Pair}(x_1,\ x_2)))}) \overset{e}{\sim} E_2} \\ \Gamma \vdash (e_1,\ (\lambda x_1.E_2)) \overset{e}{\sim} E'\end{array}}{\Gamma \vdash (\mathtt{pair}(e_1,\ e_2),\ k) \overset{e}{\sim} E'}\ \mathit{cps\_pair}$$

$\mathcal{S} : E' \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{D}_1 : e_1 \hookrightarrow v_1$,
$\mathcal{C}'_1 : \Gamma \vdash v_1 \overset{v}{\sim} V_1$, and
$\mathcal{S}'_1 : \underbrace{((\lambda x_1.E_2)\ V_1)}_{\overset{\beta}{=}[V_1/x_1]E_2} \longmapsto^* \mathsf{vl}(W)$ By I.H. on $\mathcal{C}_1$
$\Gamma \vdash ([v_1/x_1]e_2,\ (\lambda x_2.[V_1/x_1]k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} [V_1/x_1]E_2$
By the Substitution lemma
$\Gamma \vdash (e_2,\ (\lambda x_2.k\ (\mathsf{Pair}(V_1,\ x_2)))) \overset{e}{\sim} [V_1/x_1]E_2$
x_1 occurs only once in the continuation
$\mathcal{S}'_1 : [V_1/x_1]E_2 \longmapsto^* \mathsf{vl}(W)$ By previous lines
$\mathcal{D}_2 : e_2 \hookrightarrow v_2$,
$\mathcal{C}'_2 : \Gamma \vdash v_2 \overset{v}{\sim} V_2$, and
$\mathcal{S}'_2 : \underbrace{((\lambda x_2.k\ (\mathsf{Pair}(V_1,\ x_2)))\ V_2)}_{\overset{\beta}{=}(k\ \mathsf{Pair}(V_1,\ V_2))} \longmapsto^* \mathsf{vl}(W)$ By I.H. $(\mathcal{S}'_1 < \mathcal{S})$
$\mathcal{D} : \mathtt{pair}(e_1,\ e_2) \hookrightarrow \mathtt{pair}^*(v_1,\ v_2)$ By $ev\_\mathtt{pair}$
$\mathcal{C}' : \Gamma \vdash \mathtt{pair}^*(v_1,\ v_2) \overset{v}{\sim} \mathsf{Pair}(V_1,\ V_2)$ By $cps\_\mathtt{pair}^*$
$\mathcal{S}' : (k\ \mathsf{Pair}(V_1,\ V_2)) \longmapsto^* \mathsf{vl}(W)$ By $\mathcal{S}'_2$

Case: $\mathcal{C}$ =

$$\dfrac{\overset{\mathcal{C}_1}{\Gamma \vdash (e,\ (\lambda x_1.\mathsf{Fst}(x_1,\ k))) \overset{e}{\sim} E'}}{\Gamma \vdash (\mathtt{fst}\ e,\ k) \overset{e}{\sim} E'}\ \mathit{cps\_fst}$$

$\mathcal{S} : E' \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{D}_1 : e \hookrightarrow v$,
$\mathcal{C}'_1 : \Gamma \vdash v \overset{v}{\sim} V$, and
$\mathcal{S}'_1 : \underbrace{((\lambda x_1.\mathsf{Fst}(x_1,\ k))\ V)}_{\overset{\beta}{=}\mathsf{Fst}(V,\ k)} \longmapsto^* \mathsf{vl}(W)$ By I.H. on $\mathcal{C}_1$
$V = \mathsf{Pair}(V_1,\ V_2)$ and
$\mathcal{S}_1 : \mathsf{Fst}(\mathsf{Pair}(V_1,\ V_2),\ k) \longmapsto \underbrace{(k\ V_1) \longmapsto^* \mathsf{vl}(W)}_{\mathcal{S}'}$ By red_fst
$v = \mathtt{pair}^*(v_1,\ v_2)$, $\Gamma \vdash v_1 \overset{v}{\sim} V_1$, and $\Gamma \vdash v_2 \overset{v}{\sim} V_2$
By inversion on $cps\_\mathtt{pair}^*$
$\mathcal{D}_1 : e \hookrightarrow \mathtt{pair}^*(v_1,\ v_2)$ By previous lines
$\mathcal{D} : \mathtt{fst}\ e \hookrightarrow v_1$ By ev_fst
$\mathcal{S}' : (k\ V_1) \longmapsto^* \mathsf{vl}(w)$ By previous lines

Case: $\mathcal{C}$ =

$$\dfrac{\overset{\mathcal{C}_1}{\Gamma, x, k' \vdash (e,\ k') \overset{e}{\sim} E' \quad \text{where } x \text{ and } k' \text{ are fresh}}}{\Gamma \vdash (\mathtt{lam}\ x.\ e,\ k) \overset{e}{\sim} (k\ (\mathsf{Lam}\ (x, k').\ E'))}\ \mathit{cps\_lam}$$

$\mathcal{S} : (k\ (\mathsf{Lam}\ (x, k').\ E') \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{D} : \mathtt{lam}\ x.\ e \hookrightarrow \mathtt{lam}^*\ x.\ e$ By ev_lam
$\mathcal{C}' : \Gamma \vdash \mathtt{lam}^*\ x.\ e \overset{v}{\sim} \mathsf{Lam}\ (x, k').\ E'$ By $\mathcal{C}_1$ and $cps\_\mathtt{lam}^*$
$\mathcal{S}' : (k\ (\mathsf{Lam}\ (x, k').\ E') \longmapsto^* \mathsf{vl}(W)$ By ass.

Case: $\mathcal{C}$ =

$$\dfrac{\begin{array}{l}\overset{\mathcal{C}_2}{\Gamma, x_1 \vdash (e_2, \underset{\mathcal{C}_1}{(\lambda x_2.\mathsf{App}(x_1,\ x_2,\ k))}) \overset{e}{\sim} E_2} \\ \Gamma \vdash (e_1,\ (\lambda x_1.\ E_2)) \overset{e}{\sim} E'\end{array}}{\Gamma \vdash (\mathtt{app}(e_1,\ e_2),\ k) \overset{e}{\sim} E'}\ \mathit{cps\_app}$$

$\mathcal{S} : E' \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{D}_1 : e_1 \hookrightarrow v_1$,
$\mathcal{C}'_1 : \Gamma \vdash v_1 \overset{v}{\sim} V_1$, and
$\mathcal{S}'_1 : \underbrace{((\lambda x_1.\ E_2)\ V_1)}_{\overset{\beta}{=}[V_1/x_1]E_2} \longmapsto^* \mathsf{vl}(W)$ By I.H. on $\mathcal{C}_1$
$\Gamma \vdash ([v_1/x_1]e_2,\ (\lambda x_2.[V_1/x_1]\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} [V_1/x_1]E_2$
By the Substitution lemma
$\Gamma \vdash (e_2,\ (\lambda x_2.\mathsf{App}(V_1,\ x_2,\ k))) \overset{e}{\sim} [V_1/x_1]E_2$
x_1 occurs only once in the continuation
$\mathcal{S}'_1 : [V_1/x_1]E_2 \longmapsto^* \mathsf{vl}(W)$ By previous lines
$\mathcal{D}_2 : e_2 \hookrightarrow v_2$,
$\mathcal{C}'_2 : \Gamma \vdash v_2 \overset{v}{\sim} V_2$, and
$\mathcal{S}'_2 : \underbrace{((\lambda x_2.\mathsf{App}(V_1,\ x_2,\ k))\ V_2)}_{\overset{\beta}{=}\mathsf{App}(V_1,\ V_2,\ k)} \longmapsto^* \mathsf{vl}(W)$ By I.H. $(\mathcal{S}'_1 < \mathcal{S})$
$V_1 = \mathsf{Lam}\ (x, k').\ E$ (for some E) and
$\mathcal{S}_2 : \mathsf{App}((\mathsf{Lam}\ (x, k').\ E),\ V_2,\ k) \longmapsto \underbrace{[V_2/x][k/k']E \longmapsto^* \mathsf{vl}(W)}_{\mathcal{S}_3}$
By red_app
$v_1 = \mathtt{lam}^*\ x.\ e$ and $\Gamma, x, k' \vdash (e,\ k') \overset{e}{\sim} E$
By inversion on $cps\_\mathtt{lam}^*$
$\Gamma \vdash ([v_2/x]e,\ k) \overset{e}{\sim} [V_2/x][k/k']E$ By the Substitution lemma
$\mathcal{S}_3 : [V_2/x][k/k']E \longmapsto^* \mathsf{vl}(W)$ By previous lines
$\mathcal{D}_3 : [v_2/x]e \hookrightarrow v$, $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$ (for some V), and
$\mathcal{S}' : (k\ V) \longmapsto^* \mathsf{vl}(W)$ By I.H. $(\mathcal{S}_3 < \mathcal{S})$
$\mathcal{D} : \mathtt{app}(e_1,\ e_2) \hookrightarrow v$ By ev_app

□

It is important to realize that this lemma cannot be (automatically) proven only by applying structural induction on the length of reduction sequences $\mathcal{S} : E \longmapsto^* \mathsf{vl}(W)$ nor purely on the CPS-translation $C : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$. Clearly the CPS-reduction sequence is not decreasing in several cases. For the CPS-translation, the difficulty is to establish that we apply the CPS-translation always to a smaller expression. However this is non-trivial and relies essentially on the property that every variable in the CPS-continuation occurs only once, i.e. CPS-continuations as linear. An important observation is that we do not necessarily need linearity to justify the soundness proof, since in the cases where we cannot directly establish that the CPS-translation is applied to a smaller term, the CPS-reduction sequence is decreasing.

Theorem 4 (Soundness).
If $\mathcal{C} : \Gamma \vdash (e,\ \lambda x.\mathsf{vl}(x)) \overset{e}{\sim} E$, *and* $\mathcal{S} : E \longmapsto^* \mathsf{vl}(V)$ *then* $\mathcal{D} : e \hookrightarrow v$, *and* $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$.

Proof. By assumption we know that $\mathcal{C} : \Gamma \vdash (e,\ \lambda x.\mathsf{vl}(x)) \overset{e}{\sim} E$ and $\mathcal{S} : E \longmapsto^* \mathsf{vl}(V)$. By the previous lemma, we know that $\mathcal{D} : e \hookrightarrow v$, $C' : \Gamma \vdash v \overset{v}{\sim} V$ and $\mathcal{S}' : \underbrace{(\lambda x.\mathsf{vl}(x))\ V}_{\overset{\beta}{=}\mathsf{vl}(V)} \longmapsto^* \mathsf{vl}(V)$. □

3.3 Completeness

In this section, we prove the completeness of our CPS-translation. Similar to the soundness theorem, which we cannot prove directly, we will need to first prove the "Completeness" lemma. In order to prove the Completeness lemma, a utility lemma needs to be proven first.

Lemma 3 (Totality lemma).

1. *For any Mini−ML value v, there exists some CPS value V such that $\Gamma \vdash v \overset{v}{\sim} V$ is derivable.*
2. *For any Mini−ML expression e and continuation k, there exists some CPS expression E such that $\Gamma \vdash (e,\ k) \overset{e}{\sim} E$ is derivable.*

Proof. Proof by structural induction on the Mini−ML value v and Mini−ML expression e. For each Mini−ML value v and Mini−ML expression e, we have a corresponding CPS translation inference rule, which transforms the Mini−ML value v or Mini−ML expression e into their corresponding CPS value V or CPS expression E. □

The totality lemma states that for any well-formed value v in the direct-style source program (the Mini-ML program), we can always CPS-transform it into a well-formed CPS value v by a CPS translation $\Gamma \vdash v \overset{v}{\sim} V$. Similar meaning as for the Mini-ML expressions. Note, that in order to prove the totality lemma for the case of Mini-ML value $\texttt{lam}^*\ x.\ e$, we need to use the totality lemma of Mini-ML expression for the body e.

Lemma 4 (Completeness).
If $\mathcal{D} : e \hookrightarrow v$ and $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$, and $\mathcal{S}' : (k\ V) \longmapsto^* \mathsf{vl}(W)$ and $\mathcal{C} : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$ then $\mathcal{S} : E \longmapsto^* \mathsf{vl}(W)$ is derivable.

Proof. By structural induction on the derivation $\mathcal{D} : e \hookrightarrow v$. We only show a few cases in the proof in detail; the remaining ones follow the same pattern.

Case: $\mathcal{D} =$

$$\frac{\overset{\mathcal{D}_1}{e_1 \hookrightarrow v_1} \quad \overset{\mathcal{D}_2}{e_2 \hookrightarrow v_2}}{\texttt{pair}(e_1,\ e_2) \hookrightarrow \texttt{pair}^*(v_1,\ v_2)}\ ev\_pair$$

$\mathcal{C}' : \Gamma \vdash \texttt{pair}^*(v_1,\ v_2) \overset{v}{\sim} V$ By ass.
$V = \mathsf{Pair}(V_1,\ V_2)$ and $\Gamma \vdash v_1 \overset{v}{\sim} V_1$, $\Gamma \vdash v_2 \overset{v}{\sim} V_2$ By $cps\_\texttt{pair}^*$
$\mathcal{C} : \Gamma \vdash (\texttt{pair}(e_1,\ e_2),\ k) \overset{e}{\sim} E'$ By ass.
$\mathcal{C}_2 : \Gamma, x_1 \vdash (e_2,\ (\lambda x_2.k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} E_2$, and
$\mathcal{C}_1 : \Gamma \vdash (e_1,\ (\lambda x_1.E_2)) \overset{e}{\sim} E'$ By inversion on $cps\_\texttt{pair}$
$\mathcal{S}' : (k\ \mathsf{Pair}(V_1,\ V_2)) \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{S}'_2 : ((\lambda x_2.k\ (\mathsf{Pair}(V_1,\ x_2)))\ V_2) \longmapsto^* \mathsf{vl}(W)$ By β-reduction
$\mathcal{C}_3 : \Gamma \vdash ([v_1/x_1]e_2,\ (\lambda x_2.[V_1/x_1]k\ (\mathsf{Pair}(x_1,\ x_2)))) \overset{e}{\sim} [V_1/x_1]E_2$ By the Substitution lemma
$\mathcal{C}_3 : \Gamma \vdash (e_2,\ (\lambda x_2.k\ (\mathsf{Pair}(V_1,\ x_2)))) \overset{e}{\sim} [V_1/x_1]E_2$ x_1 occurs only once in the continuation
$\mathcal{S}_3 : [V_1/x_1]E_2 \longmapsto^* \mathsf{vl}(W)$ By I.H.
$\mathcal{S}'_1 : ((\lambda x_1.E_2)\ V_1) \longmapsto^* \mathsf{vl}(W)$ By previous lines
$\mathcal{S} : E' \longmapsto^* \mathsf{vl}(W)$ By I.H.

Case: $\mathcal{D} =$

$$\frac{}{\texttt{lam}\ x.\ e \hookrightarrow \texttt{lam}^*\ x.\ e}\ ev\_lam$$

$\mathcal{C}' : \Gamma \vdash \texttt{lam}^*\ x.\ e \overset{v}{\sim} V$ By ass.
$V = \mathsf{Lam}\ (x, k').\ E'$ (for some E') By $cps\_\texttt{lam}^*$
$\mathcal{S}' : (k\ (\mathsf{Lam}\ (x, k').\ E')) \longmapsto^* \mathsf{vl}(W)$ By ass.
$\mathcal{C} : \Gamma \vdash (\texttt{lam}\ x.\ e,\ k) \overset{e}{\sim} E$ By ass.
$E = (k\ (\mathsf{Lam}\ (x, k').\ E'))$ By $cps\_\texttt{lam}$
$\mathcal{S} : (k\ (\mathsf{Lam}\ (x, k').\ E')) \longmapsto^* \mathsf{vl}(W)$ By ass.

Case: $\mathcal{D} =$

$$\frac{\overset{\mathcal{D}_1}{e_1 \hookrightarrow \texttt{lam}^*\ x.\ e'} \quad \overset{\mathcal{D}_2}{e_2 \hookrightarrow v'_2} \quad \overset{\mathcal{D}_3}{[v'_2/x]e' \hookrightarrow v}}{\mathsf{app}(e_1,\ e_2) \hookrightarrow v}\ ev\_app$$

$\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$, $\mathcal{S}' : (k\ V) \longmapsto^* \mathsf{vl}(W)$ and
$\mathcal{C} : \Gamma \vdash (\mathsf{app}(e_1,\ e_2),\ k) \overset{e}{\sim} E'$ By ass.
$\mathcal{C}'_1 : \Gamma \vdash \texttt{lam}^*\ x.\ e' \overset{v}{\sim} \mathsf{Lam}\ (x, k').\ E$ (for some E), and
$\Gamma, x, k' \vdash (e',\ k') \overset{e}{\sim} E$ By the Totality lemma
$\mathcal{C}'_2 : \Gamma \vdash v'_2 \overset{v}{\sim} V'_2$ (for some V'_2) By the Totality lemma
$\mathcal{C}_3 : \Gamma \vdash ([v'_2/x]e',\ k) \overset{e}{\sim} [V_2/x][k/k']E$ By the Substitution lemma
$\mathcal{S}_3 : [V_2/x][k/k']E \longmapsto^* \mathsf{vl}(W)$ By I.H.
$\mathcal{S}'_3 : \mathsf{App}((\mathsf{Lam}\ (x, k').\ E),\ V_2,\ k) \longmapsto [V_2/x][k/k']E$ By red_app
$\mathcal{S}'_2 : ((\lambda x_2.\mathsf{App}((\mathsf{Lam}\ (x, k').\ E),\ x_2,\ k))\ V_2) \longmapsto^* \mathsf{vl}(W)$ By β-reduction
$\mathcal{C}_2 : \Gamma, x_1 \vdash (e_2,\ (\lambda x_2.\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} E_2$,
$\mathcal{C}_1 : \Gamma \vdash (e_1,\ (\lambda x_1.E_2)) \overset{e}{\sim} E'$ By inversion on cps_app
$\mathcal{C}_3 : \Gamma \vdash (e_2,\ (\lambda x_2.[\mathsf{Lam}\ (x, k').\ E/x_1]\mathsf{App}(x_1,\ x_2,\ k))) \overset{e}{\sim} E'_2$
and $E'_2 = [(\mathsf{Lam}\ (x, k').\ E)/x_1]E_2$ By the Substitution lemma
$\mathcal{C}_3 : \Gamma \vdash (e_2,\ (\lambda x_2.\mathsf{App}((\mathsf{Lam}\ (x, k').\ E),\ x_2,\ k))) \overset{e}{\sim} E'_2$ x_1 occurs only once in the continuation
$\mathcal{S}_2 : [(\mathsf{Lam}\ (x, k').\ E)/x_1]E_2 \longmapsto^* \mathsf{vl}(W)$ By I.H.
$\mathcal{S}'_1 : ((\lambda x_1.E_2)\ (\mathsf{Lam}\ (x, k').\ E)) \longmapsto^* \mathsf{vl}(W)$ By previous lines
$\mathcal{S} : E' \longmapsto^* \mathsf{vl}(W)$ By I.H.

□

Theorem 5 (Completeness).
If $\mathcal{D} : e \hookrightarrow v$, and $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$, and $\mathcal{C} : \Gamma \vdash (e,\ \lambda x.\mathsf{vl}(x)) \overset{e}{\sim} E$ then $\mathcal{S} : E \longmapsto^* \mathsf{vl}(V)$.

Proof. By assumption we know that $\mathcal{D} : e \hookrightarrow v$, and $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$, and $\mathcal{C} : \Gamma \vdash (e,\ \lambda x.\mathsf{vl}(x)) \overset{e}{\sim} E$, and we also know that $\mathcal{S}' : \underbrace{(\lambda x.\mathsf{vl}(x))\ V}_{\overset{\beta}{=}\mathsf{vl}(V)} \longmapsto^* \mathsf{vl}(V)$. By the previous lemma, we know that $\mathcal{S} : E \longmapsto^* \mathsf{vl}(V)$. □

By proving the soundness and completeness of our CPS translation, we can show the correctness of our CPS transformation. We will present the formalization of our CPS translation and proofs in the next section.

4 Formalization in a meta-logical framework

From the previous section, one may have the feeling that, as transformations become more and more sophisticated, their hand-written correctness proofs become less and less readable and reliable. Thus, it is desirable to formalize and verify the correctness of program transformations such as CPS transformation automatically with theorem provers (Minamide & Okuma 2003) or other deductive systems.

We formalize our CPS translation in the meta-logical framework Twelf (Pfenning & Schürmann 1999, Pfenning & Schürmann 2002). It is a framework used for the specification, implementation, and meta-theory of deductive systems from the theory of programming languages and logics. The encoding style in Twelf is very intuitive and simple from a logic programming point of view, thus it is easier to understand. The formalization task consists of four stages:

1. The representation of the abstract syntax of the Mini-ML and CPS language.
2. The representation of the Mini-ML big-step operational semantics and the small-step reduction semantics of the CPS language.
3. The representation of the CPS transformation inference rules between the Mini-ML and CPS language.

4. The representation of meta-theory of the language (for example, the correctness proof of CPS transformation).

4.1 Formalization of the languages

Twelf employs the representation methodology and underlying type theory of the LF logical framework. The first task in the formalization of a language in a logical framework is the representation of its expressions. We base the representation on abstract (rather than concrete) syntax in order to expose the essential structure of the object language so we can concentrate on semantics and meta-theory, rather than details of lexical analysis and parsing. Expressions are represented as LF objects using the technique of *higher-order abstract syntax* whereby variables of an object language are mapped to variables in the meta-language. This means that common operations such as renaming of bound variables or capture-avoiding substitution are directly supported by the framework, so that we get bound variables for free, and do not need to implement any new capture-avoiding substitution for each object language.

The meta-language of LF is the $\lambda\Pi$-calculus. It is a three-level hierarchical calculus for *objects*, *families* and *kinds*. Families are classified by kinds and objects are classified by types that is families of kind type (Pfenning 2001). So for each (abstract) syntactic category of the object language we introduce a new type constant in the meta-language via a declaration of the form a:type. Thus, in order to represent CPS language expressions and values we declare a type cexp and cval in the meta-language (the representation of the Mini-ML language is similar).

```
cexp : type.
cval : type.
```

We intend that every LF object M of type cexp represents a CPS language expression and vice versa. The zero constant Zero is now represented by an LF constant Z declared in the meta-language to be of type cval.

```
Z : cval.
```

The successor Succ is a value constructor. It is represented by a constant of functional type that maps CPS values to CPS values so that, for example, Succ Zero has type cval.

```
Succ : cval -> cval.
```

As discussed in Section 2 a continuation k is a function that takes in a CPS value and returns a CPS expression. So a continuation k has the type of cval $\rightarrow$ cexp. Recall the application CPS expression App(V_1, V_2, k), it has the type cexp.

```
App : cval -> cval -> (cval -> cexp) -> cexp.
```

Other constructs are defined in the same pattern. The definition of the abstract syntax of the CPS language is encoded in Twelf as follows:

```
cexp  : type.  %name cexp CE.
cval  : type.  %name cval CV.

%% CPS expressions
app+  : cval -> cval -> (cval -> cexp) -> cexp.
fst+  : cval -> (cval -> cexp) -> cexp.
snd+  : cval -> (cval -> cexp) -> cexp.
case+ : cval -> cexp -> (cval -> cexp) -> cexp.
vl+   : cval -> cexp.

% CPS values
z+    : cval.
s+    : cval -> cval.
pair+ : cval -> cval -> cval.
lam+  : (cval -> (cval -> cexp) -> cexp) -> cval.
```

We annotate the CPS language terms with the symbol "+", in order to distinguish them from the terms of the Mini-ML language.

4.2 Formalization of the operational semantics

For semantic specification, LF uses the *judgments-as-types* representation technique. This means that a derivation is coded as an object whose type represents the judgment it establishes. Checking the correctness of a derivation is thereby reduced to type-checking its representation in the logical framework (which is efficiently decidable) (Pfenning & Schürmann 1999, Pfenning & Schürmann 2002).

In the big-step operational semantics of Mini-ML, the evaluation judgment: $e \hookrightarrow v$ is defined in Twelf as the eval judgment.

```
eval : exp -> value -> type.
```

The eval predicate is very like a function, which takes in a Mini-ML expression and returns the value of evaluating this expression.

The following shows the Twelf encoding of the inference rules for eval of pairs and functions. The other cases follow the same pattern. Throughout the formalization in Twelf we reverse the function arrows writing $A_2 \leftarrow A_1$, instead of $A_1 \rightarrow A_2$ following logic programming notation. A more detailed discussion of this encoding style is given in (Pfenning 2001).

```
% Pairs
ev_pair : eval (pair E1 E2) (pair* V1 V2)
           <- eval E1 V1
           <- eval E2 V2.
ev_fst  : eval (fst E) V1
           <- eval E (pair* V1 V2).
ev_snd  : eval (snd E) V2
           <- eval E (pair* V1 V2).

% Functions
ev_lam  : eval (lam E) (lam* E).
ev_app  : eval (app E1 E2) V
           <- eval E1 (lam* E1')
           <- eval E2 V2
           <- eval (E1' V2) V.
```

The term (E1' V2) formalizes substitution by β-reduction. The expression E1' is of function type $(\lambda x.(\texttt{E1'}x))$, and we pass V2 into the body of the function (V2 substitutes the occurrence of variable x in E1').

The single-step and multi-step reduction relations between two CPS expressions are formalized by two predicates in Twelf.

```
=>  : cexp -> cexp -> type.
=>* : cexp -> cexp -> type.
```

All the reduction inference rules are expressed by:

```
%infix none 6 =>.
%infix none 5 =>*.
stop : E =>* E.     % reflexivity
<<   : E =>* E'     % transitivity
        <- E => E1
        <- E1 =>* E'.

cred_case+_z : (case+ z+ E2 E3) => E2.
cred_case+_s : (case+ (s+ V1') E2 E3) => (E3 V1').
cred_fst+  : (fst+ (pair+ V1 V2) K) => (K V1).
cred_snd+  : (snd+ (pair+ V1 V2) K) => (K V2).
cred_app+  : (app+ (lam+ E1') V2 K) => (E1' V2 K).
```

The predicates => and =>* are set as $infix$ operators. The reflexivity rule is indicated as a stopped state.

4.3 Formalization of CPS transformation

The CPS transformation is formulated by two judgments: $\Gamma \vdash v \overset{v}{\sim} V$ and $\Gamma \vdash (e,\ k) \overset{e}{\sim} E$. We can interpret these two judgments by assuming that v, e and k are given as inputs and V, E are to be constructed as

outputs. The two judgments are formalized in Twelf by two predicates: cpsExp and cpsValue. The Twelf definitions of these two judgments follow the same as their type definitions:

```
cpsValue : value -> cval -> type.
cpsExp   : exp -> (cval -> cexp) -> cexp -> type.
```

The CPS transformation rules for Mini-ML values are expressed by:

```
cpsV_z*   : cpsValue z* z+.
cpsV_s*   : cpsValue (s* V) (s+ V+)
              <- cpsValue V V+.
cpsV_pair*: cpsValue (pair* V1 V2) (pair+ V1+ V2+)
              <- cpsValue V1 V1+
              <- cpsValue V2 V2+.
cpsV_lam* : cpsValue (lam* E) (lam+ E')
              <- ({x:value} {x':cval}
                   cpsValue x x'
                  -> {k:cval -> cexp}
                       cpsExp (E x) k (E' x' k)).
```

In the case of CPS transformation for `lam`*, we implicitly create a new continuation `k`, which we use it to transform the function body `E` to its corresponding CPS expression, by plugging in a value into `E`. The term `{x:value}` means that for all `x` that is of type `value`. The big-brackets `{ }` in Twelf has the similar meaning to the universal quantifier ($\forall$) in first-order logic.

The following shows some cases of the formalization of CPS transformation rules for Mini-ML expressions in Twelf. The other cases follow the same pattern.

```
cps_pair: cpsExp (pair E1 E2) K E'
           <- ({x1':cval}
            cpsExp E2 ([x2':cval] K (pair+ x1' x2'))
                                                (E2' x1'))
           <- cpsExp E1 E2' E'.

cps_lam: cpsExp (lam E) K (K (lam+ E'))
          <- ({x:value} {x':cval}
               cpsValue x x'
          -> {k:cval -> cexp}
                 cpsExp (E x) k (E' x' k)).

cps_app: cpsExp (app E1 E2) K E'
          <- ({x1:cval}
            cpsExp E2 ([x2:cval] app+ x1 x2 K) (E2' x1))
          <- cpsExp E1 ([x1:cval] E2' x1) E'.
```

The term `[x:cval]` is equal to λx, where x is of type `cval`. The predicate `cpsExp` can be viewed as a function, where its inputs are a Mini-ML expression e and a continuation (`K`), and its output is a CPS expression (of type `cexp`).

4.4 Formalization of the correctness proof

In Twelf we can express the meta-theory of deductive systems using *higher-level judgments* (Pfenning & Schürmann 1999). A higher-level judgment describes a relation between derivations inherent in a (constructive) meta-theoretic proof. So we can execute a meta-theoretic proof using the operational semantics for LF. Twelf checks the proof by type-checking the judgments. However, type-checking a higher-level judgment does not by itself guarantee that it correctly implements a proof.

Recall the Soundness lemma in Section 2. It is represented in Twelf as follows:

```
cpsd : cpsExp E K M
       -> M =>* (vl+ W)
       -> {V:value}eval E V
       -> {V+:cval}cpsValue V V+
       -> (K V+) =>* (vl+ W)
       -> type.
%name cpsd CS.
%mode cpsd +C +S -V -D -V+ -C' -S'.
```

The mode declaration:

```
%mode cpsd +C +S -V -D -V+ -C'-S'.
```

specifies the inputs and outputs of the predicate `cpsd`. The "+" mode indicates inputs and the "−" mode indicates outputs. This follows exactly the statements of the lemma, where all the assumptions are indicated as inputs whereas the conclusions are indicated as outputs. The mode checker verifies that all inputs are known when the predicate is called and all output arguments are known after successful execution of the predicate (Pfenning & Schürmann 2002). The predicate `cpsd` is defined just like what the Soundness lemma is stated. The inputs of the predicate assume that $\mathcal{C} : \Gamma \vdash (e,\ k) \overset{e}{\sim} E$ and $\mathcal{S} : E \longmapsto^* \mathsf{vl}(W)$ exist, and the outputs are to be $\mathcal{D} : e \hookrightarrow v$, $\mathcal{C}' : \Gamma \vdash v \overset{v}{\sim} V$ and $\mathcal{S}' : (k\ V) \longmapsto^* \mathsf{vl}(W)$.

The following shows the formalization of the Soundness lemma proof for the cases of `pair`, `lam` and `app`. The other cases follow the same pattern.

```
% Pairs
cpsd_pair: cpsd (cps_pair ES1 ES2) C
                (pair* V1 V2) (ev_pair D2 D1)
                (pair+ V1+ V2+) (cpsV_pair* T2 T1) C2
            <- cpsd ES1 C V1 D1 V1+ T1 C1
            <- cpsd (ES2 V1+) C1 V2 D2 V2+ T2 C2.

% Functions
cpsd_lam: cpsd (cps_lam Es') C1 (lam* E) ev_lam
               (lam+ E') (cpsV_lam* Es') C1.
cpsd_app: cpsd (cps_app ES1 ES2) C
               V (ev_app D3 D2 D1) V+ T3 C3
           <- cpsd ES1 C
              (lam* E) D1 (lam+ E') (cpsV_lam* Es3) C1
           <- cpsd (ES2 (lam+ E')) C1
                    V2 D2 V2+ T2 (C2 << cred_app)
           <- cpsd (Es3 V2 V2+ T2 K) C2 V D3 V+ T3 C3.
```

To check the Twelf program actually constitutes a proof, meta-theoretic properties such as *coverage* and *termination* need to be established. Termination guarantees that the input of each recursive call (induction hypothesis) is smaller than the input of the original call (induction conclusion) (Pientka 2001, Pientka & Pfenning 2000). For termination checking the program needs to be well-moded. As discussed in Section 3, we specify that the predicate `cpsd` should terminate in the arguments `S` and `C` by

```
%terminates {S C} (cpsd C S V D V+ C' S').
```

Atomic, lexicographic subterm ordering is indicated by `{S C}`. For reduction checking we specify an explicit order relation between input and output elements. In the Soundness lemma proof, we declare

```
%reduces S' <= S (cpsd C S V D V+ C' S').
```

to verify that S' is a rest computation of S.

Coverage says that the execution will always make progress. In order to correctly coverage check the proof, we need to give correct specification of relation `cpsd`'s mode, and the specification of the applicable *world*. Worlds in Twelf work very much like modes. When specifying a world for a relation, it does two things. First, it specifies the appropriate world for that relation; and second, it checks that the entire computation rooted at that relation (ie, that relation, and every other one it calls) is well-worlded (Harper & Crary 2005, Pfenning & Schürmann 2002). The world declaration is defined for `cpsd` as follows:

```
%worlds () (cpsd C S V D V+ C' S').
```

this is actually a special case of regular world declaration, which declares that the type families in `cpsd C S V D V+ C' S'` do not introduce any new parameters or hypotheses. After specifying the mode

and world declaration, we can then activate the coverage checker by specifying the coverage declaration:

```
%covers cpsd +C +S -V -D -V' -C' -S'.
```

The coverage checker only checks for "input" coverage, that is if all the possible cases for a given collection of input arguments are covered (Schürmann & Pfenning 2003). In order to check "output" coverage (the output arguments to the subgoal cover all the possible values that may be returned in these positions) (Pfenning & Schürmann 2002), we need to specify the "totality" checker as well:

```
%total {S C} (cpsd C S V D V+ C' S').
```

The Soundness theorem is expressed by:

```
sound: cpsExp E ([x:cval] vl+ x) E+
       -> {V+:cval}E+ =>* vl+(V+)
       -> {V:value}eval E V
       -> cpsValue V V+
       -> type.
%mode sound +C +V+ +S -V -D -T.
```

The proof is simple, since it is the corollary of the Soundness lemma.

```
sound_cps: sound C V+ S V D T
            <- cpsd C S V D V+ T stop.
```

The `stop` state indicates the reflexivity rule in the CPS expression reductions. In the CPS language small-step reduction semantics, we define that the CPS expression $\mathsf{vl}(V)$ cannot be reduced. However, in the formalization of the proof, we say that $\mathsf{vl}(V)$ can be reduced in one step to itself as the reduction stops.

The formalizations of the proofs for the Completeness lemma and theorem follow the same pattern as the Soundness ones, except we need to formalize the Totality lemma.

```
tcps: {E}{K}cpsExp E K E' -> type.
%mode tcps +E +K -D.

tcpsv: {V}cpsValue V V+ -> type.
%mode tcpsv +V -D'.
```

The predicate `tcpsv` says that for any Mini-ML value `V`, there exists a derivation `D'`, where `V` can be CPS transformed into a CPS value `V+`. The predicate `tcps` represents the same meaning for Mini-ML expressions.

The following shows two cases in the formalization of the proof for the Totality lemma, the other cases follow the same pattern.

```
tcps_pair: tcps (pair E1 E2) K (cps_pair D1 D2)
           <- tcps E1 K D1
           <- {x1':cval}
           tcps E2 ([x2':cval] K (pair+ x1' x2'))
                                          (D2 x1').

tcpsv_lam*: tcpsv (lam* E) (cpsV_lam* D')
             <- ({x:value}{x':cval}{u:cpsValue x x'}
                 {k:cval -> cexp}
                 tcps (E x) k (D' x x' u k)).
```

The Twelf code of the CPS translation (including all the formalization of the four stages) can be found at `http://www.cs.mcgill.ca/~ytian8/CPS/`.

5 Conclusions and Future Work

People from the research community of the programming languages are now interested in a question (Aydemir, Bohannon, Fairbairn, Foster, Pierce, Sewell, Vytiniotis, Washburn, Weirich & Zdancewic 2005): How close are we to a world where programming language papers are routinely supported by machine-checked metatheory proofs, where full-scale language definitions are expressed in machine-processed mathematics, and where language implementations are directly tested against those definitions? This paper is intend to be a guide to a formal development of CPS compilation techniques as well as a case study of the question above.

We have considered a typical example from verifying the correctness of abstract machines (Hannan & Pfenning 1992). To prove correctness, we need to show the correspondence between the high-level language and the low-level abstract machine. In other words, we can translate programs written in the high-level language into programs which run on the abstract machine. Moreover, the source program always has the same observable behavior as the target program. The proof is constructive and constitutes a program which translates derivations in the source language into derivations of the target language and vice versa. We consider Mini-ML as the source language, and the terms in continuation-passing style as the low-level target language.

We have presented a higher-order setting of CPS transformation, which operates in one-pass and directly produces compact CPS programs without administrative redexes (Danvy 1991, Danvy & Nielsen 2002, Danvy & Nielsenand 2001). This higher-order CPS transformation also simplified the process of proving the correctness of CPS transformation, as we do not need to use colon-translation to handle those administrative redexes. We have encoded the CPS translation and the correctness proofs in the meta-logical framework Twelf using higher-order abstract syntax. We have mechanically verified the correctness of our CPS transformation and also other properties like "termination" of our CPS transformation and "uniqueness" of the CPS expression reductions.

This paper has showed a lot of benefits of using meta-logical approach of formalizing metatheory proofs of programming languages and program compilations. People have also demonstrated that using meta-logical approach enables relatively rapid development of foundational certified code (Crary & Sarkar 2003). We are interested in exploring the power of using meta-logical approach of formalizing and verifying theory foundations of programming languages in the future.

6 Acknowledgements

This work could not have been accomplished without the insights, persistence and critical feedback of my advisor Brigitte Pientka.

References

Appel, A. W. (1992), *Compiling with Continuations*, Cambridge University Press.

Aydemir, B. E., Bohannon, A., Fairbairn, M., Foster, J. N., Pierce, B. C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S. & Zdancewic, S. (2005), Mechanized metatheory for the masses: The POPLmark Challenge, *in* 'Proceedings of TPHOLs 2005: the 18th International Conference on Theorem Proving in Higher Order Logics (Oxford)'.

Crary, K. & Sarkar, S. (2003), Foundational certified code in a metalogical framework., *in* 'CADE', pp. 106–120.

Danvy, O. (1991), Three steps for the cps transformation, Technical Report CIS-92-2, Department

of Computing and Information Sciences, Kansas State University.

Danvy, O., Dzafic, B. & Pfenning, F. (1999), On proving syntactic properties of CPS programs, *in* 'Proceedings of the Third International Workshop on Higher Order Operational Techniques in Semantics (HOOTS'99)', Paris, pp. 19–31. Electronic Notes in Theoretical Computer Science, Volume 26.

Danvy, O. & Filinski, A. (1992), 'Representing control: A study of the CPS transformation', *Mathematical Structures in Computer Science* **2**(4), 361–391.

Danvy, O. & Nielsen, L. R. (2002), A first-order one-pass cps transformation, *in* 'Proceedings of Foundations of Software Science and Computation Structures, FOSSACS', 2303 of LNCS, pp. 98–113.

Danvy, O. & Nielsenand, L. R. (2001), A higher-order colon translation, *in* 'Proceedings of International Symposium on Functional and Logic Programming, FLOPS', pp. 78–91.

Guy L. Steele, J. (1978), Rabbit: A compiler for scheme, Technical report, Cambridge, MA, USA.

Hannan, J. & Pfenning, F. (1992), Compiler verification in LF, *in* A. Scedrov, ed., 'Seventh Annual IEEE Symposium on Logic in Computer Science', Santa Cruz, California, pp. 407–418.

Harper, R. & Crary, K. (2005), How to believe a twelf proof.
*`http://www-2.cs.cmu.edu/~rwh/papers/how/believe-twelf.pdf`

Minamide, Y. & Okuma, K. (2003), Verifying cps transformations in isabelle/hol, *in* 'Proceedings of the 2003 Workshop on Mechanized Reasoning about Languages with Variable Binding', pp. 1–8.

Pfenning, F. (2001), *Computation and Deduction*, Cambridge University Press. In preparation. Draft from April 1997 available electronically.

Pfenning, F. & Schürmann, C. (1999), System description: Twelf — a meta-logical framework for deductive systems, *in* 'Proceedings of the 16th International Conference on Automated Deduction (CADE-16)', Springer-Verlag LNAI 1632, Trento, Italy, pp. 202–206.

Pfenning, F. & Schürmann, C. (2002), *Twelf User's Guide*, 1.4 edn. Available as Technical Report CMU-CS-98-173, Carnegie Mellon University.

Pientka, B. (2001), Termination and reduction checking for higher-order logic programs, *in* 'IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning', Springer-Verlag, London, UK, pp. 401–415.

Pientka, B. & Pfenning, F. (2000), Termination and reduction checking in the logical framework, *in* C. Schürmann, ed., 'Workshop on Automation of Proofs by Mathematical Induction', Pittsburgh, Pennsylvania.

Plotkin, G. (1975), 'Call-by-name, call-by-value and the $\lambda-$calculus', *Theoretical Computer Science* **1**, 125–159.

Schürmann, C. & Pfenning, F. (2003), A coverage checking algorithm for lf., *in* 'TPHOLs', pp. 120–135.

Xi, H. & Schürmann, C. (2001), 'CPS Transform for Dependent ML (abstract)', *Logic Journal of IGPL* **9**(5), 739–754.