Study in Continuation Passing Styles

A Report for
CS320N – Mini Project

Under the guidance of
Dr. Sanjiva Prasad
Computer Science Department

Akash Lal – 1999131
Deepak Garg – 1999144
Computer Science Department
Acknowledgement

We would like to sincerely thank Dr. Sanjiva Prasad, Computer Science and Engg. Deptt, IIT-Delhi for his guidance and support throughout the project.
Contents

1. Continuation Passing Styles 4
   1.1 Informal Introduction to CPS 4
   1.2 Formal CPS 4
2. Simplest form of CPS 4
   2.1 Call-by-value 5
   2.2 Call-by-name 6
   2.3 CPS Transform 6
3. CPS Transform with Control Operators 8
   3.1 Evaluation Contexts 8
   3.2 Control Operators 8
   3.3 CPS Transform for C and A 9
4. Typing the CPS Transform 10
   4.1 Introduction to Typing 10
   4.2 Type Inference for the CPS Transform 12
5. Category Theory 13
   5.1 Introduction 13
   5.2 Typed λ-calculus Interpreted in CCC 16
   5.3 Interpreting CPS Transform in a Category 17

Bibliography 19
Diagrams 20
1 Continuation Passing Style

1.1 Informal Introduction to CPS

Any function making a function call usually saves its pending computation on a stack. This remaining computation is also called the *continuation* at the given point in time. The continuation passing style attempts to avoid saving this continuation by explicitly passing it as an argument to the called function. This is done by creating a procedural abstraction of the remaining computation. This gives the called function control over the continuation. It can under suitable circumstances decide to ignore it. Consider the following recursively defined factorial function,

\[ \text{fact}(n) = \text{if}(n = 0) \text{ then } 1 \text{ else } n \times \text{fact}(n-1); \]

Observe that with every recursive call, the remaining computation of multiplying with previous values of the parameter has to be stored somewhere. A corresponding CPS transformed program could be the following,

\[ \text{fact}_{\text{cps}}(n, k) = \text{if}(n = 0) \text{ then } k(1) \]
\[ \text{else } \text{fact}_{\text{cps}}(n-1, \text{fn } x \Rightarrow (x \times k(n))); \]

The continuation is now captured in the abstraction \( \text{fn } x \Rightarrow (x \times k(n)) \). It can easily be seen that \( \text{fact}(n) = \text{fact}_{\text{cps}}(n, \text{fn } x \Rightarrow x) \) for all values of \( n \).

1.2 Formal CPS

Formally, the continuation passing style is a transformation applied to \( \lambda \)-calculus terms. The technique can be used to simulate a call-by-name language using a call-by-value language and vice-versa. If we consider \( \lambda \)-terms equivalent to Natural Deduction proofs by the Curry-Howard isomorphism, continuation passing style acts as an embedding of classical logic into intuitionistic logic. This result we will prove later.

2 Simplest Form of CPS

The \( \lambda \)-calculus terms can be defined as follows,

\[ M ::= x \mid \lambda x.M \mid M_1M_2 \]
2.1 Call-by-value

The call-by-value reduction attempts to simplify the arguments to a value before applying them to an abstraction. The notion of value in this reduction strategy may be defined by the following syntax

\[ V ::= x \mid \lambda x. M \]

Big-step reduction \( eval_V \) may be defined as follows,

\[
\begin{align*}
\text{eval}_V(x) &= x \\
\text{eval}_V(MN) &= \text{eval}_V(M'[N'/x]) \\
&\text{if eval}_V(M) = \lambda x. M' \text{ and eval}_V(N) = N'
\end{align*}
\]

The notion of \( \beta \)-equality that makes this a calculus is as follows,

\[
\lambda x. M = \lambda y. M[y/x] \quad y \notin \text{FV}(M) \quad (\alpha \text{ rule})
\]

\[
(\lambda x. M)N = M[N/x] \quad \text{if } N \text{ is a value. (\beta \text{ rule})}
\]

\[
\begin{align*}
M &= M \\
M = N &\quad N = L \\
M &= L \\
M = N &\quad N = M \\
M = N &\quad MZ = NZ \\
MZ = NZ &\quad M = N \\
ZM = ZN &\quad M = N \\
\lambda x. M &= \lambda x. N
\end{align*}
\]

We say that \( \lambda V \vdash M = N \) if this is provable using the above rules.
2.2 Call-by-name

The call-by-name reduction does not simplify arguments before applying them to abstractions. Consequently, the big step reduction is modified as follows,

\[ \text{eval}_N(x) = x \]
\[ \text{eval}_N(MN) = \text{eval}_N(M'[N/x]) \text{ if } \text{eval}_N(M) = \lambda x. M' \]

The notion of \( \beta \)-equality that makes this a calculus is as follows,

\[ \lambda x. M = \lambda y. M[y/x] \quad y \notin \text{FV}(M) \ (\alpha \text{ rule}) \]

\[ (\lambda x. M)N = M[N/x] (\beta \text{ rule}) \]

\[
\begin{align*}
M &= M \\
M &= N \quad N = L \\
M &= L \\
M &= N \\
N &= M \\
M &= N \\
MZ &= NZ \\
& \quad MZ = NZ \\
& \quad ZM = ZN \\
M &= N \\
\lambda x. M &= \lambda x. N
\end{align*}
\]

We say that \( \lambda_N \vdash M = N \) if this is provable using the above rules.

2.3 CPS Transform

The simplest form of CPS, as defined by G.D. Plotkin [1], is mentioned below. This can be used to simulate call-by-value using call-by-name in the untyped \( \lambda \)-calculus. Later we shall see an extension of this to the typed \( \lambda \)-calculus and a calculus with control operators.
Given a call-by-value calculus $\lambda_V$, define a simulation map on its terms as follows,

$$
\bar{x} = \lambda \kappa. (\kappa \ x)
$$

$$
\overline{\lambda x.M} = \lambda \kappa. (\kappa (\lambda x. \overline{M}))
$$

$$
\overline{M \overline{N}} = \lambda \kappa. (\overline{M} (\lambda \alpha. \overline{N} (\lambda \beta. (\alpha \ \beta \ \kappa)))
$$

We will refer this map as the transformation to the continuation passing style. Thus $\overline{M}$ is the CPS transform of $M$.

An auxiliary function $\Psi$ sending values to values is defined as,

$$
\Psi(x) = x
$$

$$
\Psi(\lambda x. M) = \lambda x. \overline{M}
$$

The following three theorems show that it is possible to simulate a call-by-value calculus using a call-by-name one.

**Theorem 1.** (Indifference) $\text{eval}_N(\overline{M}(\lambda x. x)) = \text{eval}_V(\overline{M}(\lambda x. x))$ for any closed term $M$.

*Comments: Given the identity continuation $\lambda x. x$, CPS terms are indifferent to reduction in either strategy.*

**Theorem 2.** (Simulation) $\Psi(\text{eval}_V(M)) = \text{eval}_N(\overline{M} \lambda x. x)$

*Comments: Since $\lambda x. x$ is the identity continuation, the simulation is nearly perfect.*

**Theorem 3.** (Translation) If $\lambda^L \vdash M = N$ then $\lambda^L \vdash \overline{M} = \overline{N}$ and then $\lambda^N \vdash \overline{M} = \overline{N}$. The second but not the first implication is reversible. Here $L$ is the call-by-value language given to us, $L'$ is the language of CPS terms interpreted using call-by-name and $L''$ is the latter language interpreted in a call-by-value sense.

*Comments: In CPS transform, the $\beta$-equality of terms is unchanged whether we interpret the transformed terms in the call-by-value or call-by-name sense. Further, $\beta$-equality of untransformed terms implies $\beta$-equality of transformed terms.*

It is worth mentioning that G.D.Plotkin[1] also mentions a second CPS transform that can be used to simulate call-by-name languages using call-by-value strategy.
3 CPS Transform with Control Operators

3.1 Evaluation Contexts

One way to look at the notion of continuations or remaining computation is to define formally the context in which a particular reduction is taking place. For the call-by-value strategy, one can define the evaluation context as,

\[ E ::= [] \mid EN \mid VE \]

An evaluation context is actually a \( \lambda \)-term with exactly one subterm replaced by a [], which we call a hole. Any subterm of \( E \) containing this hole cannot be a value. A representation like \( E[M] \) means that we have completed the context \( E \) by replacing its hole using \( M \). The following lemma establishes the uniqueness of representation of a term in this form.

**Lemma 1.** Any closed term \( M \) can be uniquely written as \( E[N] \) for some \( \beta_V \)-redex \( N \).

Further, if we define reduction on these terms by the rule,

\[ E[(\lambda x. M)V] \xrightarrow{\beta_V} E[M[V/x]], \]

then it can be shown that for closed terms, \( eval_V(M) = V \) iff \( M \xrightarrow{\beta_V} V \). It can also be established that reduction on \( E[M] \) first reduces \( M \) to a value and then reduces other subterms of \( E \). Thus the following lemma holds,

**Lemma 2.** If \( E[M] \xrightarrow{\beta_V} V \), then \( \exists V_0 \) such that \( E[M] \xrightarrow{\beta_V} E[V_0] \xrightarrow{\beta_V} V \).

3.2 Control Operators

The \( \lambda \)-calculus can be extended to provide it as much power as **Scheme** that has the control operators `call/cc`. This operator makes a procedural abstraction of the current continuation and applies it to its argument. Idealized Scheme (IS) is an extension of the call-by-value \( \lambda \)-calculus with two operators \( A \) and \( C \). So the grammar for IS is,

\[ M ::= x \mid \lambda x. M \mid M_1 M_2 \mid C(M) \mid A(M) \]

\( A \) is called the abort operator and \( C \) is the control operator.

The reduction rules for \( C \) and \( A \) may be summarised as follows,

\[ E[A(M)] \xrightarrow{A} M \]
\[
E[C(M) \mapsto_c M(\lambda z. A(E[z])))
\]

Using these, we can define abort in terms of control.

\[
A(M) = C(\lambda d. M)
\]

Here \(d\) is a dummy variable not free in \(M\).

Informally, \(A\) throws away the current continuation and \(C\) passes a procedural abstraction of the current continuation as an argument to \(M\). If ever this abstraction is invoked, the then context is aborted and this context is restored. The \textit{call/cc} operator can be operationally substituted by \(K\).

\[
E[K(C(M))] \mapsto_K E[M(\lambda z. A(E[z]))]
\]

### 3.3 CPS Transform for \(C\) and \(A\)

The CPS transform for the control operators may be defined as follows,

\[
\overline{C(M)} = \lambda \kappa. (\overline{M (\lambda \alpha. \alpha \ (\lambda z. \lambda d. \kappa z) \ \lambda x. A(x)))}
\]

\[
\overline{A(M)} = \lambda \kappa. \overline{M(\lambda x. A(x))}
\]

This is the transform given Timothy G. Griffin[2]. It differs from the transform given by Felleisen[3]. The transform suggested by Felleisen[3] was,

\[
\overline{C(M)} = \lambda \kappa. (\overline{M (\lambda \alpha. \alpha \ (\lambda z. \lambda d. \kappa z) \ \lambda x. x)})
\]

\[
\overline{A(M)} = \lambda \kappa. \overline{M(\lambda x. x)}
\]

In our opinion, this is more appropriate semantically. The first transform for abort, given a continuation ignores it since \(\kappa\) does not appear free in the remaining term. However, it also passes as the next continuation an abstraction which when applied would generate another abort operator. That is clearly not the reduction rule \(\mapsto_A\). The second rule for abort, aborts the continuation passed to it and then passes \text{\textit{id}} as the continuation for the remaining computation. This is more appropriate. Operationally, the two rules are equivalent as continuations are applied only after the rest of the computation has ended. As a result this extra abort continuation would attempt to throw away continuations generated before it. These, however, have already been aborted. Hence it has no effect.

From the point of view of typing and equivalence with proofs, the transform suggested by Timothy G. Griffin[2] is better. We shall use this in the remainder of our report.
4 Typing the CPS Transform

4.1 Introduction to Typing

The simply typed λ-calculus is a λ-calculus where each term has an associated type. The λ-calculus presented in the beginning has a type system that corresponds exactly to the minimal natural deduction with the rules \((Ax)\), \((\rightarrow E)\) and \((\rightarrow I)\). To make it correspond to the natural deduction with conjunction, we extend the λ calculus with the following terms,

\[ M ::= \ldots \mid \langle M, N \rangle \mid \pi_1(M) \mid \pi_2(M) \mid * \]

\(\langle M, N \rangle\) is the tupling operator, and \(\pi_i\) is the \(i\)th projection operator. \(*\) is a distinguished term characterised by rules given later. The typing rules are discussed later.

The λ-calculus with the control operators can be typed consistently if we make the assumption that every continuation has the return type \(o\). The type \(o\) corresponds to the formula false or \(\bot\) i.e. a formula that has no proof. As we shall see later, to interpret the λ-calculus in category theory, we need to have a distinguished type \(1\), that corresponds to the proposition true.

The types may be defined as follows (\(t_i\) is a primitive type),

\[ T ::= t_i \mid T_1 \rightarrow T_2 \mid T_1 \times T_2 \mid o \mid 1 \]

We define \(\neg T\) as \(T \rightarrow o\). Now the following typing rules can be shown consistent. It can also be proved that types are preserved under reduction using \(\rightarrow_{\beta_v}, \rightarrow_{\lambda} \) and \(\rightarrow_c\).

\[
\frac{\Gamma \vdash \star : 1}{(Ax)} \]

\[
\frac{x : \alpha \in \Gamma}{\Gamma \vdash x : \alpha} (Ax) \]

\[
\frac{\Gamma \vdash x : \alpha \quad \Gamma \vdash M : \beta}{\Gamma \vdash \lambda x. M : \alpha \rightarrow \beta} (\rightarrow I) \]

\[
\frac{\Gamma \vdash M : \alpha \rightarrow \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta} (\rightarrow E) \]

10
\[ \frac{\Gamma \vdash M : \alpha \quad \Gamma \vdash N : \beta}{\Gamma \vdash \langle M, N \rangle : \alpha \times \beta} \quad (\land I) \]
\[ \frac{\Gamma \vdash M : \alpha \times \beta}{\Gamma \vdash \pi_1(M) : \alpha} \quad \frac{\Gamma \vdash M : \alpha \times \beta}{\Gamma \vdash \pi_2(M) : \beta} \quad (\land E) \]
\[ \frac{\Gamma \vdash M : \neg \alpha}{\Gamma \vdash C(M) : \alpha} \quad (CI) \]
\[ \frac{\Gamma \vdash M : \varnothing}{\Gamma \vdash A(M) : \alpha} \quad (AI) \]
\[ \frac{\Gamma \vdash M : \neg\neg\alpha \rightarrow \alpha}{\Gamma \vdash K(M) : \alpha} \quad (KI) \]

It can be seen that if we omit the $\lambda$-terms, then all these rules exist or can be derived in classical natural deduction in the Gentzen’s intuitionistic style. Thus there exists a correspondence between typed terms of IS and the classical natural deduction. This is an extension of the Curry-Howard isomorphism. From now on, we will refer to type inference rules and corresponding proof rules by the same name.

The notion of $\beta$ equality can be extended to the typed $\lambda$ calculus without $C$ and $A$ operators as follows:

\[ \frac{M : 1}{M = \ast} \]
\[ \frac{\pi_1(M, N) = M}{\pi_2(M, N) = N} \]
\[ M : \alpha \times \beta \]

Using these rules, we can derive the following rules,

\[ \frac{M, N : \alpha \times \beta}{\pi_1(M) = \pi_1(N)} \]
\[ \frac{M, N : \alpha \times \beta}{\pi_2(M) = \pi_2(N)} \]
\[ M = M' \quad N = N' \]
\[ (M, N) = (M', N') \]
4.2 Type Inference for the CPS Transform

We will discuss the CPS transform for IS without the $A$ operator only. This we will assume to be defined in terms of the $C$ operator. For typing this IS, we first define a transform ($*$) on types.

$$t_i^* = t_i$$

$$(\alpha \rightarrow \beta)^* = \alpha^* \rightarrow \neg \beta^*$$

Using this transform on types, we will prove the following theorem,

**Theorem 4.** If $M : \alpha$, then $\overline{M} : \neg \alpha^*$. This is the only possible type of $\overline{M}$ subject to the constraint that all continuations have the return type $o$. Here $M$ is a term in IS.

**Proof.** (By induction on the length of the term)

**Base Case:** $M \equiv x : \alpha$.

$\overline{M} = \lambda x. (\kappa x)$. Since $x : \alpha^*$ in the transformed language and $\kappa$ must have a return type $o$, $\kappa : \alpha^* \rightarrow o$ i.e. $\kappa : \neg \alpha^*$. Therefore, the type of $\overline{M}$ can be inferred as $\neg \alpha^* \rightarrow o$ i.e. $\neg \alpha^*$.

**Induction Hypothesis:** The statement of the theorem is true for all $M$ of length less than $n$.

**Induction Step:**

Case 1. $M \equiv \lambda x . N$ and $\alpha \equiv \beta \rightarrow \gamma$.

By I.H., $\overline{N} : \neg \neg \gamma^*$. Then, since in the transformed language, $x : \beta^*$, $\kappa : (\beta^* \rightarrow \neg \neg \gamma^*) \rightarrow o \equiv \delta$. Now, the type of $\overline{M}$ is $\delta \rightarrow o$. Now using the transform on types and the value of $\delta$, we have $\overline{M} : \neg \neg \alpha^*$.

Case 2. $M \equiv M_1 M_2$. This case is similar to the above case.

Case 3. $M \equiv C(N)$. This case is very long and hence omitted. It is similar to case 1, except that according to the typing rule mentioned before, the $A$ operator occuring in $\overline{M}$ can have any type. However, consistent typing forces the type to $o$.

The following theorem follows directly from the above theorem by simply observing that CPS terms do not have any subterm containing $C$. Let $M$ be the minimal implicational logic and $L$ be the classical implicational logic defined by adding the ($CI$) rule to $M$. We also define constructive implicational logic $J$ as $M$ enriched with ($AI$) rule.

**Theorem 5.** If $M$ corresponds to the proof $\Gamma \vdash_C \alpha$, then $\overline{M}$ is a proof of $\Gamma^* \vdash_J \neg \neg \alpha^*$. Here $\Gamma^*$ is defined as $\{ \alpha^* \mid \alpha \in \Gamma \}$

A consequence of this theorem is that the CPS embeds the classical logic $C$ into the intuitionistic logic $J$.
5 Category Theory

Category Theory gives a convenient method to study the Typed λ-calculus. We will start by defining some basic concepts in category theory and then try to adapt it for analysing the typed λ-calculus.

5.1 Introduction

**Graph:** A graph $\mathcal{G}$ is a collection of nodes (or objects) and arrows. Each arrow has a source node and a target node. $\mathcal{G}_0$ refers to the collection of nodes, $\mathcal{G}_1$ refers to the collection of arrows and $\mathcal{G}_2$ is the collection of paths of length 2.

**Category:** A category is a graph $\mathcal{G}$ with two functions $\text{compose} : \mathcal{G}_2 \to \mathcal{G}_1$ and $\text{id} : \mathcal{G}_0 \to \mathcal{G}_1$ satisfying the properties listed below. We will use the notation $f;g$ to denote $\text{compose}(f,g)$ and $\text{id}_A$ to denote $\text{id}(A)$.

1. The source of $f;g$ is the source of $f$ and the target of $f;g$ is the target of $g$.
2. $(f;g);h = f;(g;h)$
3. Source and target of $\text{id}_A$ are $A$.
4. If $f : A \to B$ then $\text{id}_A;f = f;\text{id}_B = f$.

**Example:** The category of sets and functions is one whose nodes are sets and arrows are functions between them. Composition is the usual function composition. Identity arrows are the identity maps. This category is also called $\text{Set}$.

**Opposite Category:** The opposite category $\mathcal{C}^{\text{op}}$ of category $\mathcal{C}$ is one whose objects are the objects of $\mathcal{C}$ and whose arrows are obtained by reversing the arrows of $\mathcal{C}$.

**Definition:** An arrow $f : A \to B$ is called an isomorphism if there exists an arrow $g : B \to A$ such that $f;g = \text{id}_A$ and $g;f = \text{id}_B$. In this case, $A$ and $B$ are called isomorphic objects.

**Functor:** A functor $F : \mathcal{C} \to \mathcal{D}$ is a homomorphism from category $\mathcal{C}$ to category $\mathcal{D}$, satisfying the following properties,

1. If $f : A \to B$ in $\mathcal{C}$, then $F(f) : F(A) \to F(B)$ in $\mathcal{D}$.
2. $F(\text{id}_A) = \text{id}_{F(A)}$ for all objects $A$ of $\mathcal{C}$.
3. $F(f;g) = F(f);F(g)$, if either side is defined.
**Example:** Consider the category 1 that has only one object $\ast$ and a single arrow $id_{\ast}$. Then given any category $\mathcal{C}$, the function mapping all objects of $\mathcal{C}$ to $\ast$ and all its arrows to $id_{\ast}$, is a functor from $\mathcal{C}$ to 1.

**Definition:** A diagram is said to commute if the composition of any two paths with the same source and destination is the same.

**Hom Set:** Given any two objects $A$ and $B$ of a category $\mathcal{C}$, $\text{Hom}_{\mathcal{C}}(A, B)$ is the set of all arrows from $A$ to $B$. When there is no scope for confusion, this is also written without the subscript $\mathcal{C}$.

**Hom Functor:** Any functor induced using Hom Sets is called a Hom Functor. For example, given a category $\mathcal{C}$ and an object $A$ of it, the functor $\text{Hom}(A, -) : \mathcal{C} \to \text{Set}$, is defined as follows,

1. $\text{Hom}(A, -)(B) = \text{Hom}(A, B)$
2. $\text{Hom}(A, -)(f : B \to C) = \text{Hom}(A, f) : \text{Hom}(A, B) \to \text{Hom}(A, C)$ that takes $g : A \to B$ to $g; f : A \to C$.

**Natural Transformation:** Let $D, E : \mathcal{G} \to \mathcal{C}$ be functors. A natural transformation $\alpha : D \to E$ is a family $\alpha_A$ of arrows of $\mathcal{C}$ indexed by the nodes of $\mathcal{G}$ such that

1. $\alpha_A : D(A) \to E(A)$ for each object $A$ of $\mathcal{G}$.
2. For any arrow $f : A \to B$ in $\mathcal{G}$, Fig.1 commutes.

**Example:** Given an arrow $f : D \to C$ in a category $\mathcal{C}$, we can define a natural transformation $\alpha$ between the functors $\text{Hom}(C, -)$ and $\text{Hom}(D, -)$ as follows,

$\alpha_A$ is an arrow from $\text{Hom}(C, A)$ to $\text{Hom}(D, A)$ in $\text{Set}$ that takes an arrow $g : C \to A$ to the arrow $f; g : D \to A$.

In fact, every natural transformation between these functors is induced by a unique arrow from $D$ to $C$.

**Natural Isomorphism:** A natural transformation $\alpha : F \to G$ is called a natural isomorphism if every component $\alpha_A$ of this is an isomorphism. In that case we say that the functors $F$ and $G$ are isomorphic.

**Adjoint:** $F : \mathcal{A} \to \mathcal{B}$ and $U : \mathcal{B} \to \mathcal{A}$ are called left and right adjoint to each other respectively, if there is natural transformation $\eta : id_{\mathcal{A}} \to U \circ F$ such that for any objects $A$ of $\mathcal{A}$ and $B$ of $\mathcal{B}$ and any arrow $f : A \to U(B)$, there is a unique arrow $g : F(A) \to B$ such that $f = \eta_A ; U(g)$. In this case we say $F \dashv U$. Further, $\eta$ is called the **unit** of adjunction.
Proposition 1. Following the notation of the above definition, there exists a natural transformation $\epsilon : F \circ U \to \text{id}_B$ such that for any $g : F(A) \to B$, there is a unique arrow, $f : A \to U(B)$ with $g = F(f) ; \epsilon_B$. This transformation is called the counit of adjunction.

In view of the above definition and proposition, Fig. 2 commutes.

Theorem 6. Let $F : A \to B$ and $U : B \to A$ be functors. Then $F \dashv G$ iff $\text{Hom}(F \cdot \cdot \cdot, \cdot \cdot \cdot)$ and $\text{Hom}(\cdot \cdot \cdot, U \cdot \cdot \cdot)$ are naturally isomorphic as functors from $A^{op} \times B \to \text{Set}$.

Cartesian Product of objects: The product of two objects $A$ and $B$ is any object $A \times B$ together with two arrows $\text{proj}_1 : A \times B \to A$ and $\text{proj}_2 : A \times B \to B$ satisfying the following.

Given any object $V$ and arrows $f : V \to A$ and $g : V \to B$, $\exists$ a unique arrow $(f, g) : V \to A \times B$ such that $(f, g) ; \text{proj}_1 = f$ and $(f, g) ; \text{proj}_2 = g$. In other words, the diagram in Fig. 3 should commute.

Cartesian Product of arrows: The product of two arrows $f : S \to S'$ and $g : T \to T'$ is an arrow $f \times g : S \times T \to S' \times T'$ such that the diagram in Fig. 4 commutes.

Terminal object: An object $1$ is called terminal if given any object $A$ there is a unique arrow from $A$ to $1$.

Initial object: An object $0$ is called initial if given any object $A$ there is a unique arrow from $0$ to $A$.

Cartesian Closed Categories (CCC): A category $C$ is called a cartesian closed category if it satisfies the following properties,

1. There is a terminal object $1$.
2. Product of any two objects exists in the category.
3. Given any two objects $A$ and $B$, there is an object $[A \to B]$ and an arrow $\text{eval} : [A \to B] \times A \to B$ such that, given any arrow $f : C \times A \to B$, there is a unique arrow $\lambda f : C \to [A \to B]$ such that the composite

$$C \times A \xrightarrow{\lambda f \times id_A} [A \to B] \times A \xrightarrow{\text{eval}} B$$

is $f$.

15
Example: In a CCC $C$, consider the functor $\_ \times A : C \to C$. A right adjoint to this functor is $[A \to \_] : C \to C$ such that $[A \to \_](B) = [A \to B]$ and $[A \to \_](f) = [A \to f] \equiv \lambda (eval; f)$. This implies that $\text{Hom}(\_ \times A, \_)$ is naturally isomorphic to $\text{Hom}(\_,[A \to \_])$.

**Proposition 2.** The following hold in any CCC for all objects $B$ and $C$,  
(1) $\text{Hom}(\_ , B) \times \text{Hom}(\_ , C) \cong \text{Hom}(\_ , B \times C)$  
(2) $\text{Hom}(\_ , [B \to \_]) \cong \text{Hom}(\_ \times B, \_)$

### 5.2 Typed $\lambda$-Calculus Interpreted in CCC

The extra structure of a CCC gives us the ability to interpret a simply typed $\lambda$-calculus with the operations of tupling, projection, abstraction and application using category theory. The CCC can be constructed as follows,

(1) Primitive types of the $\lambda$-calculus are objects of the CCC. The derived type $A \to B$ is the object $[A \to B]$ and $A \times B$ is the object $A \times B$.

(2) Arrows of the CCC are proofs/$\lambda$-terms. An equivalence class (under $\beta$ equality) of $\lambda$-terms of type $B$ with at most one free variable of type $A$ is an arrow from $A \to B$.

(3) Composition of arrows is defined by substitution. If $\phi$ is a term of type $B$ with at most one free variable $x$ of type $A$ and $\psi$ is a term of type $C$ with at most one free variable $y$ of type $B$, then the composite of the arrows corresponding to these terms is $\psi[\phi/y]$.

(4) The identity arrow on the object $A$ is $id_A \equiv \lambda x^A.x$. Although this term has no free variable, we assume it to have a free variable of type 1. Then $id_A : 1 \to [A \to A]$ and using the properties of a CCC, this arrow can be proved isomorphic to an arrow from $A$ to $A$. The term satisfies the properties of the identity arrow.

(5) The arrow $\text{eval} : [A \to B] \times A \to B$ can be defined as follows. Take any variable $x$ of type $[A \to B] \times A$. Now take the term $(\pi_1(x)\pi_2(x))$. This has type $B$ and the corresponding arrow satisfies the properties of $\text{eval}$. Now, given an arrow $f : C \times A \to B$ i.e. a term $M$ of type $B$ with at most one free variable $x$ of type $C \times A$, consider the term $\lambda y^A.M[x/\langle z^C,y \rangle]$. This term has at most one free variable of type $C$ and has a type $[A \to B]$. Hence its corresponding arrow has the type $C \to [A \to B]$. This is the arrow $\lambda f$.

(6) The terminal object is the type 1. This is because there is only one term * of type 1 and this has no free variables. So this term corresponds to a unique arrow from any object $A$ to 1. Since we are not dealing with control and abort operators, we do not have the type $o$ in our type system. Later we shall see the implications of including $o$ in our category.
From this it follows that the intuitionistic natural deduction with → and ∧ can also be interpreted in a cartesian closed category. Then Proposition 2.1 captures all the rules for ∧ introduction and elimination and Proposition 2.2 is the deduction theorem.

5.3 Interpreting CPS Transform in a Category

As we have seen, it is possible to type the CPS transform. Theorem 4 suggests that doing this requires the introduction of the type o. We have typed the $A$ operator as the rule,
\[
\frac{\Gamma \vdash o}{\Gamma \vdash \alpha} \quad \forall \alpha
\]

Since $o$ corresponds to ⊥, it would translate into an initial object when we try to interpret in a categorical framework. Now we require a CCC with an initial element $o$. The following are important results about such a category.

**Result 1.** If $C$ is a CCC with an initial object $o$, then $o \times A \cong o$.

**Proof.** $\text{Hom}(o \times A, B) \cong \text{Hom}(o, [A \to B])$. Since $o$ is initial, $\text{Hom}(o, [A \to B])$ has exactly one element and so does $\text{Hom}(o \times A, B)$. So $o \times A$ is also initial which means that $o \cong o \times A$.

**Result 2.** If $C$ is a CCC with an initial object $o$ and $\text{Hom}(A, o) \neq \emptyset$, then $A \cong o$ i.e. $\text{Hom}(A, o)$ has atmost one element.

**Proof.** For any object $A$, let $\langle A \rangle$ denote the unique arrow $o \to A$. Then if $f : A \to o$, $\langle A \rangle$ holds $id_o$. Now since $o \times A \cong o$, $\text{proj}_1 : o \times A \to o = id_o$. Since $\langle f, id_A \rangle; \text{proj}_1 = f$, we have $\langle f, id_A \rangle = f$. Also, $\text{proj}_2 = \langle A \rangle$. Now, $\langle f, id_A \rangle; \text{proj}_2 = id_A$ and we have $f; \langle A \rangle = id_A$. Hence, $f$ and $\langle A \rangle$ are inverses of each other and $A \cong o$.

**Result 3.** If $C$ is a CCC interpreting classical logic and for objects $A$, $A \cong \neg \neg A$, then for all objects $A$ and $B$, there is atmost one arrow from $A$ to $B$.

**Proof.** First, $\bot$ must correspond to the initial element $o$ of the category. Next, $A \cong \neg \neg A$ for all objects $A$ in the category. Hence, for all objects $A$ and $B$, $\text{Hom}(A, B) \cong \text{Hom}(A, \neg \neg B) \equiv \text{Hom}(A, [\neg B \to o]) \cong \text{Hom}(A \times \neg B, o)$. By result 2, the latter has atmost one element and hence, our result is proved.

This result tells us that if we want to interpret the CPS in a category, then we cannot take $A \cong \neg \neg A$.  

17
We have seen that the CPS transform of a term of the type $\alpha$ has a type $\neg \neg \alpha^*$. Further, since CPS terms do not have the $\mathcal{C}$ operator in them, it follows that the CPS transform translates proofs in classical logic to proofs in intuitionistic logic as stated by Theorem 5. Consider a category $\mathcal{G}$ that interprets classical logic. Now consider the following two functors,

1. $\text{Hom}(\_, \_): \mathcal{G}^{op} \times \mathcal{G} \rightarrow \textbf{Set}$ defined as
   
   $(A, B) \mapsto \text{Hom}(A, B)$ and
   
   $(f^{op} : B \rightarrow A, g : C \rightarrow D) \mapsto \text{Hom}(f, g)$
   
   that takes an arrow $h : B \rightarrow C$ to $f; h; g : A \rightarrow D$

2. $\text{Hom}(\_, \neg \neg \_): \mathcal{G}^{op} \times \mathcal{G} \rightarrow \textbf{Set}$ defined as

   $(A, B) \mapsto \text{Hom}(A^*, \neg \neg B^*)$ and

   $(f^{op} : B \rightarrow A, g : C \rightarrow D) \mapsto \text{Hom}(f^*, \neg \neg g^*)$

   that takes an arrow $h : B \rightarrow C$ to $\mathcal{C}(\overline{f}); \mathcal{C}(\overline{g}); \mathcal{C}(\overline{g}) : A^* \rightarrow \neg \neg D^*$

In the second functor, $f$ is a proof of $B$ from $A$ and hence $\overline{f}$ is a proof of $\neg \neg B^*$ from $A^*$. A similar argument holds for $g$. Also, the $\mathcal{C}$ operator need not be the usual control operator. It is any map that translates a proof of $\neg \neg \alpha^*$ to a proof of $\alpha^*$ for all objects $\alpha$. Therefore, the given composition is a proof as desired. The important point to observe here is that we definitely need a proof of $A^*$ from $\neg \neg A^*$.

Intuitively, it appears that the CPS transform is a natural transformation between these functors. We will now show that this is not so. If it is a natural transformation (say $\text{CPS}$), then Fig. 5 must commute for all arrows $f : A \rightarrow B$ and $g : C \rightarrow D$, and all objects $A, B, C$ and $D$. However, if we take an arrow $h : B \rightarrow C$, the upper path on the diagram maps it to $\mathcal{C}(\overline{f}); \mathcal{C}(\overline{g})$ whereas the lower path takes it to $\overline{f}; \overline{h}; \overline{g}$. The second proof is an intuitionistic proof because of the CPS transform. On the other hand, the first one involves a proof of $A^*$ from $\neg \neg A^*$. Hence, the CPS transform can be a natural transformation if for all types $A$, we can prove $A^*$ from $\neg \neg A^*$ intuitionistically. However, if we take $A$ to be a primitive type, $A = A^*$ and hence we are looking for a proof of the type

$$ \neg \neg A \quad \forall \text{ primitive } A $$

This proof cannot exist intuitionistically and hence the CPS cannot be the assumed transform.
Bibliography

Research Papers


Books


Fig. 1

Fig. 2
Fig. 3

Fig. 4

Fig. 5