PFPL Supplement: CPS Transformation*

Robert Harper

Fall, 2019

1 Introduction

The subject of this supplement is the remarkable correspondence between two seemingly disparate facts about logics and languages:[1]

1. Classical logic can be interpreted as a mode of use of constructive logic by making explicit the diminished information content of classical proofs as compared to their constructive counterparts.

2. Programming languages with non-local control constructs such as exceptions or coroutines can be understood as purely functional programs without such constructs.

Both interpretations emerge as two aspects of the same phenomenon: the logical intepretation is about the types whereas the linguistic interpretation is about the programs (of those types).

These correspondences demonstrate, once again, that the more restrictive setting (constructive logic, pure functional programming) is *more general* than more liberal setting (classical logic, functional programs with jumps.) Although many regard the more liberal setting as the more general, in fact the exact opposite is the case, because *restrictions can be selectively relaxed*, whereas *affordances cannot be selectively restricted*. Once the law of the excluded middle is admitted as a valid principle, it cannot be rescinded within a particular proof, but if it is not so admitted, it can be selectively introduced for particular purposes, recovering the seemingly more general capabilities. Once jumps are admitted as a language construct, it is not possible to preclude their use, whereas if jumps are not so admitted, then they can be used in a particular program by explicit programming of control flow.

The situation is entirely analogous to that between so-called *untyped* (safe) languages and *typed* (safe) languages. Whereas it is often considered that the untyped languages are the more general, in fact the opposite is true, at least in the presence of recursive types. For in that setting "untyped" becomes "unityped"—it is not that there are *no* types, it is that there is *only one* type, a distinguished recursive type. Everything that can be done in the untyped setting can be done in the typed setting by focussing attention on the one distinguished type. The converse is impossible: once a language is committed to there being only one type, it cannot then recover type distinctions. The seemingly more restrictive setting is, in fact, the more liberal!

[1] The correspondence with classical logic discovered by Griffin (1990) and Murthy (1991); see Reynolds (1993) for a comprehensive survey of the uses of continuations in programming languages.

$$\frac{}{\Gamma, A\,\mathsf{true} \vdash A\,\mathsf{true}} \tag{1a}$$

$$\frac{}{\Gamma \vdash \top\,\mathsf{true}} \tag{1b}$$

$$\frac{\Gamma \vdash A\,\mathsf{true} \quad \Gamma \vdash B\,\mathsf{true}}{\Gamma \vdash A \wedge B\,\mathsf{true}} \tag{1c}$$

$$\frac{\Gamma \vdash A \wedge B\,\mathsf{true}}{\Gamma \vdash A\,\mathsf{true}} \tag{1d}$$

$$\frac{\Gamma \vdash A \wedge B\,\mathsf{true}}{\Gamma \vdash B\,\mathsf{true}} \tag{1e}$$

$$\frac{\Gamma \vdash \bot\,\mathsf{true}}{\Gamma \vdash C\,\mathsf{true}} \tag{1f}$$

$$\frac{\Gamma \vdash A\,\mathsf{true}}{\Gamma \vdash A \vee B\,\mathsf{true}} \tag{1g}$$

$$\frac{\Gamma \vdash B\,\mathsf{true}}{\Gamma \vdash A \vee B\,\mathsf{true}} \tag{1h}$$

$$\frac{\Gamma \vdash A \vee B\,\mathsf{true} \quad \Gamma, A\,\mathsf{true} \vdash C\,\mathsf{true} \quad \Gamma, B\,\mathsf{true} \vdash C\,\mathsf{true}}{\Gamma \vdash C\,\mathsf{true}} \tag{1i}$$

$$\frac{\Gamma, A\,\mathsf{true} \vdash B\,\mathsf{true}}{\Gamma \vdash A \supset B\,\mathsf{true}} \tag{1j}$$

$$\frac{\Gamma \vdash A \supset B\,\mathsf{true} \quad \Gamma \vdash A\,\mathsf{true}}{\Gamma \vdash B\,\mathsf{true}} \tag{1k}$$

Figure 1: Intuitionistic Propositional Logic

2 Classical Logic as Constructive Logic

The syntax of intuitionistic propositional logic is given by the following grammar:

$$A, B, C ::= \top \mid A \wedge B \mid \bot \mid A \vee B \mid A \supset B$$

Basic assertions are of the form $A\,\mathsf{true}$, asserting that A is true, and entailments, or hypothetical assertions, are of the form

$$A_1\,\mathsf{true}, \ldots, A_n\,\mathsf{true} \vdash A\,\mathsf{true}$$

in which the left-hand side is a finite, possibly empty, set, Γ, of atomic assertions, the hypotheses or assumptions, and the right-hand side is it conclusion. By taking the assumptions to be a finite set, it is implicitly closed under contraction and permutation (duplication and reordering of assumptions is tacitly admitted.)

The rules of intuitionistic propositional logic are given in Figure 1. Negation, $\neg A$, is defined in intuitionistic logic to mean $A \supset \bot$; it is easy to derive its introduction and elimination rules from those for implication under this interpretation. The rules are arranged to ensure that weakening and transitivity are admissible; the proof of this is omitted here.

$$\frac{\Gamma, A\,\mathsf{true} \vdash \bot\,\mathsf{true}}{\Gamma \vdash \neg A\,\mathsf{true}} \tag{2a}$$

$$\frac{\Gamma \vdash \neg A\,\mathsf{true} \quad \Gamma \vdash A\,\mathsf{true}}{\Gamma \vdash \bot\,\mathsf{true}} \tag{2b}$$

$$\frac{\Gamma, \neg A\,\mathsf{true} \vdash \bot\,\mathsf{true}}{\Gamma \vdash A\,\mathsf{true}} \tag{2c}$$

Figure 2: Classical Propositional Logic (Additional Rules)

Classical logic is defined as the extension of intuitionistic logic with the additional rules given in Figure 2. Rules (2a) and (2b) define negation primitively, rather than as $A \supset \bot$. Rule (2c) is a convenient form of *double-negation elimination*. It is notably orphaned in that it does not fit the introduction/elimination pattern enjoyed by intuitionistic logic.

When it is important to distinguish the two logics, the assertion $\Gamma \vdash_I A\,\mathsf{true}$ is used for the intuitionistic case, and $\Gamma \vdash_K A\,\mathsf{true}$ for the classical case.

It may seem as though classical logic is more general than intuitionistic logic, because it validates the principle of double-negation elimination, which is not otherwise admissible. But, in fact, it is intuitionistic logic that is the more general, because classical logic may be *embedded* within it. That is, classical logic is a mode of use of constructive logic, and not the other way around. Intuitionistic logic makes a distinction between the *irrefutability* of A, written $\neg\neg A\,\mathsf{true}$, and the *truth* of A, written $A\,\mathsf{true}$. Truth is easily seen to be sufficient for irrefutability, but the converse is not generally valid in intuitionistic logic. However, the principle of double-negation elimination is added to classical logic to ensure that the converse is derivable.

For example, the assertion $\neg\neg(A \vee \neg A)\,\mathsf{true}$, called the *law of the excluded middle*, is derivable in intuitionistic logic uniformly in A, but $A \vee \neg A\,\mathsf{true}$ is not, in general, so derivable. Proving the irrefutability of the excluded middle consists in showing that any refutation of it may be refuted—that is, that the law is merely not false, rather than true, in the intuitionistic sense. To see this, suppose that there is a refutation of $A \vee \neg A\,\mathsf{true}$, which is to say a proof of $\neg(A \vee \neg A)\,\mathsf{true}$. To refute this it is enough to show $A \vee \neg A\,\mathsf{true}$, which is possible *in the presence of the presumed refutation.* It suffices to show $\neg A\,\mathsf{true}$, which is obtained by assuming $A\,\mathsf{true}$ and deriving a contradiction. This is easily obtained by noting that $A \vee \neg A\,\mathsf{true}$ is derivable from the latter assumption, and this is refuted by the former assumption! This completes the argument.[2] The rule of double-negation elimination is contrived to license the transformation of irrefutability into truth. It's orphan status in Figure 2 shows that it is an *ad hoc* device for achieving precisely this inference. One may say that classical logic does not "really" affirm the excluded middle, it merely contents itself with its irrefutability, while boasting of its truth.

This example illiustrates a general principle: any proposition that can be proved classically can be transformed systematically into another proposition that can be proved intuitionistically, precisely by making explicit the uses of double-negation elimination in classical logic. There are many such transformations, all of which are unfortunately called *the* double-negation transformation. Here is one that aligns well with the material in the next section:

$$A^\dagger \triangleq \neg\neg A^* \triangleq (A^* \supset \bot) \supset \bot$$

[2]It is, admittedly, a rather devious, but entirely valid proof.

$$\begin{aligned}
\top^* &\triangleq \top \\
(A \wedge B)^* &\triangleq A^* \wedge B^* \\
\bot^* &\triangleq \bot \\
(A \vee B)^* &\triangleq A^* \vee B^* \\
(A \supset B)^* &\triangleq A^* \supset B^\dagger \\
(\neg A)^* &\triangleq A^* \supset \bot.
\end{aligned}$$

The star translation extends to assumptions $\Gamma = A_1\,\mathsf{true}, \ldots A_n\,\mathsf{true}$ by defining $\Gamma^* \triangleq A_1^*\,\mathsf{true}, \ldots, A_n^*\,\mathsf{true}$.

This translation is chosen so as to validate the following interpretation theorem:

Theorem 2.1. *If* $\Gamma \vdash_K A\ \mathsf{true}$*, then* $\Gamma^* \vdash_I A^\dagger\ \mathsf{true}$*.*

When Γ is empty, the theorem states that if $A\,\mathsf{true}$ classically, then $\neg\neg A^*\,\mathsf{true}$ intuitionistically. Notice that the classical entailment $\neg\neg A\,\mathsf{true} \vdash_K A\,\mathsf{true}$ becomes, under translation, the intuitionistic triviality $\neg\neg A^*\,\mathsf{true} \vdash_I \neg\neg A^*\,\mathsf{true}$, which is after all the heart of the matter.

Many variations on "the" double-negation translation are possible. One of particular interest defines instead

$$\begin{aligned}
(A \supset B)^* &\triangleq A^\dagger \supset B^\dagger \\
(A \wedge B)^* &\triangleq A^\dagger \wedge B^\dagger \\
(A \vee B)^* &\triangleq (A^\dagger \vee B^\dagger)^\dagger
\end{aligned}$$

It would validate the following interpretation theorem:

Theorem 2.2. *If* $\Gamma \vdash_K A\ \mathsf{true}$*, then* $\Gamma^\dagger \vdash_I A^\dagger\ \mathsf{true}$*.*

A proper comparison of the two translations is deferred to the next section; it is nearly impossible to motivate them on purely logical grounds.

The interpretation theorem is proved by induction on the derivation of its premise. For now it is best to consider just the three characteristic rules of classical logic given in Figure 2.

Consider first rule (2c). By induction

$$\Gamma^*, (\neg A)^*\,\mathsf{true} \vdash_I \bot^\dagger\,\mathsf{true}.$$

To show $\Gamma^* \vdash_I A^\dagger\,\mathsf{true}$, suffices to show

$$\Gamma^*, A^* \supset \bot\,\mathsf{true} \vdash_I \bot\,\mathsf{true}.$$

But $(\neg A)^* = A^* \supset \bot$ by definition, and $\bot \supset \bot\,\mathsf{true}$ holds trivially, which suffices for the result.

As for rule (2a), the induction hypothesis yields

$$\Gamma^*, A^*\,\mathsf{true} \vdash_I \bot^\dagger\,\mathsf{true},$$

which implies $\Gamma^*, A^*\,\mathsf{true} \vdash_I \bot\,\mathsf{true}$, and hence $\Gamma^* \vdash_I (\neg A)^*\,\mathsf{true}$. To derive $\Gamma^* \vdash_I (\neg A)^\dagger\,\mathsf{true}$, it suffices to derive

$$\Gamma^*, (\neg A)^* \supset \bot\,\mathsf{true} \vdash_I \bot\,\mathsf{true}.$$

But this immediate given the preceding.

As for rule (2b), induction yields

1. $\Gamma^* \vdash_I (\neg A)^\dagger\,\mathsf{true}$, and

2. $\Gamma^* \vdash_I A^\dagger\,\mathsf{true}$.

To show $\Gamma^* \vdash_I \bot^\dagger\,\mathsf{true}$, it is enough to derive $\Gamma^* \vdash_I \bot\,\mathsf{true}$ from the two inductive hypotheses. By the first it suffices to derive $\Gamma^* \vdash_I (\neg A)^* \supset \bot\,\mathsf{true}$, which is to say

$$\Gamma^*, (\neg A)^*\,\mathsf{true} \vdash_I \bot\,\mathsf{true}.$$

By the second it suffices to derive

$$\Gamma^*, (\neg A)^*\,\mathsf{true}, A^*\,\mathsf{true} \vdash_I \bot\,\mathsf{true}.$$

But this is immediate by applying the first assumption to the second.

It is possible to validate all of the rules of intuitionistic logic in a similar manner, but to do so is tedious, and not very enlightening—it has the feel of pure symbol-pushing with no clear meaning. It is, in fact, quite meaningful, but to see it requires that the transformation be viewed as a compilation strategy for programs.

3 Continuations as Higher-Order Functions

To understand what is really going on, it helps to view the interpretation given in the previous section in terms of types and programs. First, as is well known, the rules of intuitionistic logic correspond to the typing rules for programs with unit, product, void, sum and function types. Second, the extension of the correspondence to classical logic amounts to introducing *first-class continuations*. Put in terms of the dynamics for classical logic given by a stack machine, stacks become first-class values in the sense of being replicable and disposable at will (that is, *persistent*) rather than there being exactly one modified-in-place (that is, *ephemeral*) run-time stack.

The proof-term assignment for intuitionistic logic being well-known, it is not repeated here, though it can be summarized by the following correspondences:[3]

Prop		*Type*
$\top$	$\leftrightarrow$	`unit`
$A_1 \wedge A_2$	$\leftrightarrow$	$A_1 \times A_2$
$\bot$	$\leftrightarrow$	`void`
$A_1 \vee A_2$	$\leftrightarrow$	$A_1 + A_2$
$A_1 \supset A_2$	$\leftrightarrow$	$A_1 \to A_2$

Classical logic extends the correspondence to include

$$\neg A \quad \leftrightarrow \quad A\,\texttt{cont}$$

wherein the type $A\,\texttt{cont}$ is the type of *cont*exts = *cont*radictions = *cont*inuations of type A, whiich are none other than the stacks acting on values of type A. The corresponding proof terms, or

[3]These correspondences are known, rather grandly, as the *Curry-Howard Isomorphism*. But it is not an isomorphism, it is not solely due to Curry and Howard, and it is true by definition, rather than discovery. It might better be called the *Curry-Howard Tautology*.

$$\frac{\Gamma, x : A \vdash M : \mathtt{void}}{\Gamma \vdash \mathtt{cont}[\,A\,](\,x\,.\,M\,) : A\,\mathtt{cont}} \tag{3a}$$

$$\frac{\Gamma \vdash M : A\,\mathtt{cont} \quad \Gamma \vdash N : A}{\Gamma \vdash \mathtt{throw}(\,M\,;\,N\,) : \mathtt{void}} \tag{3b}$$

$$\frac{\Gamma, x : A\,\mathtt{cont} \vdash M : \mathtt{void}}{\Gamma \vdash \mathtt{letcc}[\,A\,](\,x\,.\,M\,) : A} \tag{3c}$$

Figure 3: Continuation Constructs

programs, being less familiar in the classical case, are given in Figure 3. Curiously, rule (3a) is definable in terms of the other constructs:

$$\mathtt{cont}[\,A\,](\,x\,.\,M\,) \triangleq \mathtt{letcc}[\,A\,\mathtt{cont}\,](\,r\,.\,\mathtt{let}\,x : A\,\mathtt{be}\,\mathtt{letcc}[\,A\,](\,k\,.\,\mathtt{throw}(\,r\,;\,k\,)\,)\,\mathtt{in}\,M\,).$$

The intuition is that in classical logic, all propositions can be considered negated, because they are equivalent to their double-negations. In programming terms a function of type $A \to \mathtt{void}$ can be turned into a continuation of type $A\,\mathtt{cont}$, namely the continuation that, when thrown a value of type A, plugs that value into the body of the function and runs it to completion.[4]

The informal argument for the classical validity of the excluded middle can be reformulated as a program, `em`, of type $A + A\,\mathtt{cont}$ given as follows:

$$\mathtt{letcc}[\,A + A\,\mathtt{cont}\,](\,k.\mathtt{throw}(\,k\,;\mathtt{r} \cdot \mathtt{letcc}[\,A\,\mathtt{cont}\,](\,r\,.\,\mathtt{throw}(\,k\,;\mathtt{l} \cdot \mathtt{letcc}[\,A\,](\,l\,.\,\mathtt{throw}(\,r\,;\,l\,)\,)\,)\,)\,)\,).$$

This code is aptly described as a pusillanimous proof, one that "changes its mind." Let us examine it in detail. In the first instance `em` throws to the return continuation, k, a right injection of a refutation of A to be described shortly. When `em` is executed on a stack k, it *bluffs*, baldly asserting that A is false, with a carefully prepared refutation of it as evidence. The caller will, in general, perform a case analysis, see that it is the right inject, and obtain the prepared refutation. To use it, the caller must throw it a value of type A, which is of course evidence that A is, contrarily, true, thereby calling `em`'s bluff. But the devious `em` is ready for that move! The prepared refutation, when thrown a value of type A, will change its mind by throwing the left inject of that value to k, thereby returning a second time to the caller. The exact same case analysis by the caller will now take the left branch, and continue as if nothing ever happened.

Notice that in the second eventuality `em` need not fear embarassment, because it merely sends back to the caller the value of type A that the caller already had for itself. Thus, `em`'s ruse is certain to work, provided that it is allowed to return to the caller a second time. There being no memory,[5] the caller is unaware that anything untoward has happened. As far as it is concerned `em` is a legitimate proof of a disjunction on which it can perform a case analysis. Yet, `em` *never comes up with the goods*. It does not, in fact, decide whether A is true or false. Instead, it ensures that A and $\neg A$ cannot both be refuted, which is the essence of disjunction in classical logic.

The double-negation translation of propositions can now be rephrased as the *continuation-*

[4]Incidentally, an $A\,\mathtt{cont}$ may conversely be turned into an $A \to \mathtt{void}$ by throwing its argument to the given continuation.

[5]This is all pure functional programming.

passing style, or cps, transform of types:

$$\begin{aligned}
A^\dagger &\triangleq (\, A^* \to \texttt{void}\,) \to \texttt{void} \\
\texttt{unit}^* &\triangleq \texttt{unit} \\
(A \times B)^* &\triangleq A^* \times B^* \\
\texttt{void}^* &\triangleq \texttt{void} \\
(A + B)^* &\triangleq A^* + B^* \\
(A \to B)^* &\triangleq A^* \to B^\dagger \\
(A\,\texttt{cont})^* &\triangleq A^* \to \texttt{void}.
\end{aligned}$$

The star translation is extended to typing contexts by defining Γ^*, for $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, to be $x_1 : A_1^*, \ldots, x_n : A_n^*$.

This transformation may be understood in programming terms as providing a *by-value* interpretation of a language with continuations into one without. The main idea is that stacks, aka continuations or contradictions, are represented as functions of type $A^* \to \texttt{void}$. The transformed program is applied to the representation of the stack as such a function. The type A^* represents the type of *values* of type A, and the type $A^\dagger$ represents the type of *computations* of type A, and $A\,\texttt{cont}\,\texttt{cont}$ serves as the computation modality. The alternative double-negation translation mentioned in the previous section may now be understood as a *by-name* interpretation of classical logic in which variables range over computations, and pairing and injections are evaluated lazily.

With this in mind it is now relatively straightforward to give the translation of the proof terms of classical logic, including its intuitionistic fragment, as programs of appropriate type. Some clauses of the translation are given in Figure 4. The translation is governed by the following intended property of the translation:

Theorem 3.1 (CPS Transform)**.** *If* $\Gamma \vdash_K M : A$*, then* $\Gamma^* \vdash_I M^\dagger : A^\dagger$.

The verification that the translation is correct would require relating the stack machine execution of the classical program to the transition dynamics of its translation, with the main idea being that the juxtaposition of a stack and a program corresponds to the application of its translation to the functional representation of the stack.

The most notable feature of the translation of classical into intuitionistic logic given in Figure 4 is that the translation of `throw` *abandons* the return continuation (because it jumps elsewhere), and the translation of `letcc` *replicates* the return continuation (by substitution). Thus, the stack must be a persistent data structure; higher-order functions certainly qualify.

It is enlightening to examine what happens to the devious classical proof `em` given in the previous section when it is translated into intuitionistic logic. Simplifying some of the details, the translation of `em` is a program of type

$$(\,(\, A^* + (\, A^* \to \texttt{void}\,)\,) \to \texttt{void}\,) \to \texttt{void}$$

that amounts to the function

$$\lambda\,(\, k : (\, A^* + (\, A^* \to \texttt{void}\,)\,) \to \texttt{void}\,)\, k(\,\texttt{r} \cdot \lambda\,(\, a : A^*\,)\, k(\,\texttt{l} \cdot a\,)\,).$$

Oddly, it is perhaps easier to understand the compiled code than the source code!

$$
\begin{aligned}
x_i^\dagger &\triangleq \lambda\,(\,k : A_i^* \to \mathtt{void}\,)\,k(\,x\,) \\
\langle\rangle^\dagger &\triangleq \lambda\,(\,k : \mathtt{unit} \to \mathtt{void}\,)\,k(\,\langle\rangle\,) \\
\langle M_1, M_2\rangle^\dagger &\triangleq \\
&\qquad \lambda\,(\,k : A_1^* \times A_2^* \to \mathtt{void}\,)\,M_1^\dagger(\,\lambda\,(\,a_1 : A_1^*\,)\,M_2^\dagger(\,\lambda\,(\,a_2 : A_2^*\,)\,k(\,\langle a_1, a_2\rangle\,)\,)\,) \\
(\mathtt{case}\,M\,\{\,\})^\dagger &\triangleq \lambda\,(\,k : A^* \to \mathtt{void}\,)\,M^\dagger(\,\lambda\,(\,x : \mathtt{void}\,)\,\mathtt{case}\,x\,\{\,\}\,) \\
(\mathtt{l}\cdot M)^\dagger &\triangleq \lambda\,(\,k : (\,A_1^* + A_2^*\,) \to \mathtt{void}\,)\,M^\dagger(\,\lambda\,(\,a_1 : A_1^*\,)\,k(\,\mathtt{l}\cdot a_1\,)\,) \\
(\mathtt{r}\cdot M)^\dagger &\triangleq \lambda\,(\,k : (\,A_1^* + A_2^*\,) \to \mathtt{void}\,)\,M^\dagger(\,\lambda\,(\,a_2 : A_2^*\,)\,k(\,\mathtt{r}\cdot a_2\,)\,) \\
(\mathtt{case}\,M\,\{\mathtt{l}\cdot x_1 \hookrightarrow M_1 \mid \mathtt{r}\cdot x_2 \hookrightarrow M_2\})^\dagger &\triangleq \\
&\qquad \lambda\,(\,k : C^* \to \mathtt{void}\,)\,M^\dagger(\,\lambda\,(\,x : A_1^* + A_2^*\,)\,\mathtt{case}\,x\,\{\mathtt{l}\cdot x_1 \hookrightarrow M_1^\dagger(\,k\,) \mid \mathtt{r}\cdot x_2 \hookrightarrow M_2^\dagger(\,k\,)\}\,) \\
(\lambda\,(\,x : A\,)\,M)^\dagger &\triangleq \lambda\,(\,k : (\,A^* \to B^\dagger\,) \to \mathtt{void}\,)\,k(\,\lambda\,(\,x : A^*\,)\,M^\dagger\,) \\
(M_1(\,M_2\,))^\dagger &\triangleq \\
&\qquad \lambda\,(\,k : A^* \to \mathtt{void}\,)\,M_1^\dagger(\,\lambda\,(\,a : A_2^* \to A^\dagger\,)\,M_2^\dagger(\,\lambda\,(\,a_2 : A_2^*\,)\,a(\,a_2\,)(\,k\,)\,)\,) \\
(\mathtt{throw}(\,M_1\,;\,M_2\,))^\dagger &\triangleq \lambda\,(\,k : \mathtt{void} \to \mathtt{void}\,)\,M_1^\dagger(\,\lambda\,(\,a_1 : A_2^* \to \mathtt{void}\,)\,M_2^\dagger(\,\lambda\,(\,a_2 : A_2^*\,)\,a_1(\,a_2\,)\,)\,) \\
(\mathtt{letcc}[\,A\,](\,x\,.\,M\,))^\dagger &\triangleq \lambda\,(\,k : A^* \to \mathtt{void}\,)\,\{k/x\}M^\dagger(\,\lambda\,(\,x : \mathtt{void}\,)\,x\,)
\end{aligned}
$$

Figure 4: CPS Transform

4 Variations

The formulation of intuitionistic logic can be streamlined by considering an additional judgment form, $\Gamma \vdash$, asserting that Γ is inconsistent. Using this judgment form, there would actually be an introduction rule for falsehood,

$$\frac{\Gamma \vdash}{\Gamma \vdash \bot\,\mathsf{true}} \tag{4}$$

That is, if Γ is inconsistent, then falsity is true. The corresponding cps translation for this rule amounts to showing that $\Gamma^* \vdash_I \bot\,\mathsf{true}$ suffices for $\Gamma^* \vdash_I (\bot \supset \bot) \supset \bot\,\mathsf{true}$.

The contradiction judgment also supports treating negation as a primitive, decoupling it from implication and falsehood:

$$\frac{\Gamma, A\,\mathsf{true} \vdash}{\Gamma \vdash \neg A\,\mathsf{true}} \tag{5a}$$

$$\frac{\Gamma \vdash \neg A\,\mathsf{true} \quad \Gamma \vdash A\,\mathsf{true}}{\Gamma \vdash} \tag{5b}$$

For classical logic it is advantageous to take this generalization even further by considering assertions of the form $\Gamma \vdash \Delta$, where both Γ and Δ are finite, possibly empty, sets of truth assertions. Allowing multiple, perhaps no, conclusions corresponds to the vagueness of classical logic in that the possibility of non-local control transfers weaken the specification of the behavior of a proof. The assertion $\Gamma \vdash \Delta$ may be read as asserting that the conjunctive falsity of Γ is inconsistent with the

disjunctive truth of Δ. Thus, the rules of negation are given as follows:

$$\frac{\Gamma, A\,\mathsf{true} \vdash \Delta}{\Gamma \vdash \neg A\,\mathsf{true}, \Delta} \tag{6a}$$

$$\frac{\Gamma \vdash \neg A\,\mathsf{true}, \Delta \quad \Gamma \vdash A\,\mathsf{true}, \Delta}{\Gamma \vdash \Delta} \tag{6b}$$

It would be interesting to work out a corresponding translation of this formulation of classical logic into intuitionistic logic (with two forms of assertion).

References

Timothy G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 47–58, New York, NY, USA, 1990. ACM. ISBN 0-89791-343-4. doi: 10.1145/96709.96714. URL `http://doi.acm.org/10.1145/96709.96714`.

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, Cambridge, England, Second edition, 2016.

Chetan R Murthy. An evaluation semantics for classical proofs. In *Logic in Computer Science, 1991. LICS'91., Proceedings of Sixth Annual IEEE Symposium on*, pages 96–107. IEEE, 1991.

John C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3):233–247, Nov 1993. ISSN 1573-0557. doi: 10.1007/BF01019459. URL `https://doi.org/10.1007/BF01019459`.