mathematics

MDPI

Article

On Correspondence between Selective CPS Transformation and Selective Double Negation Translation

Hyeonseung Im

Department of Computer Science, Kangwon National University, Chuncheon-si, Gangwon-do 24341, Korea; hsim@kangwon.ac.kr; Tel.: +82-33-250-8441

Abstract: A double negation translation (DNT) embeds classical logic into intuitionistic logic. Such translations correspond to continuation passing style (CPS) transformations in programming languages via the Curry-Howard isomorphism. A selective CPS transformation uses a type and effect system to selectively translate only nontrivial expressions possibly with computational effects into CPS functions. In this paper, we review the conventional call-by-value (CBV) CPS transformation and its corresponding DNT, and provide a logical account of a CBV selective CPS transformation by defining a selective DNT via the Curry-Howard isomorphism. By using an annotated proof system derived from the corresponding type and effect system, our selective DNT translates classical proofs into equivalent intuitionistic proofs, which are smaller than those obtained by the usual DNTs. We believe that our work can serve as a reference point for further study on the Curry-Howard isomorphism between CPS transformations and DNTs.

Keywords: CPS transformation; double negation translation; simply typed lambda-calculus; propositional logic; Curry-Howard isomorphism

Citation: Im, H. On Correspondence between Selective CPS Transformation and Selective Double Negation Translation. *Mathematics* **2021**, *9*, 385. https://doi.org/10.3390/math9040385

Academic Editor: Josep Lluis Usó-Doménech

Received: 5 January 2021
Accepted: 8 February 2021
Published: 15 February 2021

1. Introduction

The Curry-Howard isomorphism [1,2] states that formulas and proofs in mathematical logic correspond to types and programs in programming languages. For example, propositions and proofs in intuitionistic propositional logic (IPL) correspond to types and terms in the simply typed λ-calculus ($\lambda^{\to}$). A similar correspondence also exists between classical propositional logic (CPL) and the simply typed λ-calculus with control operators such as callcc (call-with-current continuation) and throw (λ^{cont}) [3,4].

Classical logic differs from intuitionistic logic, also known as constructive logic, in that it includes the law of excluded middle (EM), $A \vee \neg A$, as a theorem, which states that for any proposition A, either A is true or its negation $\neg A$ is true (i.e., A is false). Classical logic also includes Peirce's law, $((A \supset B) \supset A) \supset A$, and double negation elimination (DNE), $\neg\neg A \supset A$. Any of these three theorems can be added as an axiom to proof systems for intuitionistic logic in order to obtain a sound and complete proof system for classical logic. Taking Peirce's law and DNE as axioms corresponds to the control operators such as callcc and Felleisen et al.'s $\mathcal{C}$ operator, which allow a nonlocal transfer of program control, respectively [3–5].

A double negation translation (DNT), also known as a negative translation, translates a classically valid formula into an intuitionistically valid one [6–11], for example, by placing a double negation $\neg\neg$ in front of every subformula of the given formula [9]. More precisely, while intuitionistic logic admits double negation *introduction*, $A \supset \neg\neg A$, it only rejects double negation *elimination*, $\neg\neg A \supset A$. In other words, $\neg\neg A$ is weaker than A in intuitionistic logic, but they are equivalent in classical logic. Hence, in general, an instance of A in a classical proof derivation only corresponds to a weaker proposition $\neg\neg A$ in an intuitionistic proof derivation. In this regard, DNTs can be thought of as a proof transformation which eliminates every instance of DNE (or equivalently, EM or Peirce's

law) used in a classical derivation of A, yielding an intuitionistic proof of $\neg\neg A^*$, where A^* is the formula obtained by placing $\neg\neg$ in front of every proper subformula of A. One important consequence of DNTs is the equivalence between the consistency of classical Peano arithmetic and the consistency of intuitionistic Heyting arithmetic.

The programming language counterpart of DNTs is continuation passing style (CPS) transformations [4,12,13], which are often used in compiler optimization [14,15]. Operationally, a continuation of type A cont is a reification of an evaluation context, i.e., the rest of the program, that expects the result of the current expression of type A. Hence, a continuation type A cont can be understood as a function type $A \to Ans$, where Ans stands for the type of final results. If we interpret Ans as $\bot$ and define $\neg A$ as a syntactic abbreviation of $A \to \bot$ as usual, then DNTs correspond to CPS transformations, which translate an expression M of type A into a CPS function of type $(A^* \to \bot) \to \bot$ that explicitly takes a continuation k as an argument and applies it to the result of M in its body. Here, the continuation k expects a value of type A^* because, as in DNTs, CPS transformations also recursively translate every subexpression of the given expression into CPS expressions. The CPS transformations from λ^{cont} into $\lambda^{\to}$ [3,4], which eliminate control operators, correspond to a DNT from CPL with Peirce's law or DNE into IPL.

In this paper, we study the logical meaning of a selective CPS transformation [16] and present a selective DNT for CPL via the Curry-Howard isomorphism. The usual CPS transformations translate into CPS functions not only "nontrivial" expressions (possibly) with control operators (computational effects) but also "trivial" expressions, which lack them, yielding unnecessarily large expressions. In contrast, the selective CPS transformation only translates nontrivial expressions into CPS functions and keeps trivial expressions intact. To do so, it uses a type and effect system to keep track of the uses of control operators. Similarly, when viewed as a proof transformation, the usual DNTs also yield unnecessarily large proofs because they translate even intuitionistically valid proof derivations. Our selective DNT uses an annotated proof system to keep track of the uses of only classical axioms, which is similar to the type and effect system in [16], and only selectively translates the part of the given proof that exploits such axioms.

The main contributions of our paper are summarized as follows:

- We review the Curry-Howard isomorphism between the standard call-by-value (CBV) CPS transformation with control operators and the corresponding DNT from CPL into IPL.
- We review a CBV selective CPS transformation based on a type and effect system [16] and propose a corresponding selective DNT based on an annotated proof system, with its correctness proof showing that provability is preserved under the translation.
- Our work can serve as a reference point for the close correspondence between the selective CPS transformation and the selective DNT and further research on this topic.

The remainder of the paper is organized as follows. Section 2 reviews the standard CBV CPS transformation from λ^{cont} with control operators into $\lambda^{\to}$ without these operators [3]. In Section 3, we present the corresponding DNT from CPL with Peirce's law into IPL. Section 4 reviews a CBV selective CPS transformation based on a type and effect system. In Section 5, we present an annotated proof system and propose a selective DNT corresponding to the selective CPS transformation via the Curry-Howard isomorphism. We also prove the correctness of the translation by showing that provability is preserved under the translation. Finally, Section 6 discusses related work and concludes.

2. A Call-by-Value CPS Transformation

In this section, we review the standard CBV CPS transformation [3] which translates a program in the source language λ^{cont} with control operators callcc and throw into an operationally equivalent program in the target language $\lambda^{\to}$ without control operators.

2.1. Simply Typed λ-Calculus with Control Operators

The source language λ^{cont} is the simply typed λ-calculus with two control operators: callcc and throw [3]. Its abstract syntax is defined as follows.

Definition 1 (Abstract syntax of λ^{cont})**.**

$$\begin{array}{rrcl} \textit{expressions} & M, N & ::= & p \mid x \mid \lambda x.M \mid M\,N \mid \mathsf{callcc}\ x.M \mid \mathsf{throw}\ M\,N \\ \textit{types} & A, B, C & ::= & P \mid A \to B \mid A\ \mathsf{cont} \end{array}$$

An expression M is either a constant p, a variable x, a lambda abstraction (anonymous function) $\lambda x.M$, a lambda application (function application) $M\,N$, or a control operator to be explained below. As for types, P ranges over an arbitrary but fixed set of primitive types, $A \to B$ denotes the type of functions from domain A to range B, and A cont denotes the type of continuations that expect a value of type A.

We consider a left-to-right CBV operational semantics for λ^{cont} using evaluation contexts, where a well-typed expression evaluates to a value.

Definition 2 (Operational semantics of λ^{cont})**.**

$$\begin{array}{rrcl} \textit{extended expressions} & M, N & ::= & \cdots \mid \langle E \rangle \\ \textit{evaluation contexts} & E & ::= & [\,] \mid E\,N \mid V\,E \mid \mathsf{throw}\ E\,N \mid \mathsf{throw}\ V\,E \\ \textit{values} & V & ::= & p \mid \lambda x.M \mid \langle E \rangle \end{array}$$

$$\begin{array}{rcl} E[(\lambda x.M)\,V] & \mapsto & E[M[V/x]] \\ E[\mathsf{callcc}\ x.M] & \mapsto & E[M[\langle E \rangle / x]] \\ E[\mathsf{throw}\ \langle E_1 \rangle\ V] & \mapsto & E_1[V] \end{array}$$

An evaluation context E is an expression with a hole $[\,]$. Every expression M can be decomposed into a unique context E and a unique reducible expression (redex) N, i.e., $M = E[N]$, where $E[N]$ denotes the expression obtained by filling the hole in E with N. An evaluation context can be considered a pending computation which expects the result of another computation. Hence, we represent a continuation as a captured evaluation context of the form $\langle E \rangle$. A value V is then either a constant, a lambda abstraction, or a continuation.

We use a reduction judgment $M \mapsto M'$ to denote one-step reduction of M to M'. Given any program, the redex to be reduced is implicitly determined by the left-to-right CBV definition of evaluation contexts. There are three kinds of redexes. First, a lambda application of the form $(\lambda x.M)\,V$ reduces to $M[V/x]$ (β-reduction), where $M[V/x]$ denotes the standard capture-avoiding substitution which replaces every occurrence of x in M with V. Since we use a CBV semantics, β-reduction always substitutes a value for the formal parameter. Second, callcc $x.M$ captures the current context E, stores a continuation $\langle E \rangle$ in x, and proceeds to reduce M. Finally, throw $\langle E_1 \rangle\ V$ throws a value V to a continuation $\langle E_1 \rangle$ and discards the current evaluation context, thus allowing a nonlocal transfer of control.

We evaluate only well-typed closed expressions [3]. To type expressions, we use a typing judgment $\Gamma \vdash M : A$ which means that expression M is of type A under typing context Γ. Here, a typing context Γ is a set of type bindings of the form $x : A$, and we use $\cdot$ to denote an empty context. We assume that each variable is associated with at most one type, for example, by means of α-conversion.

$$\text{typing contexts} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, x : A$$

A closed expression M is well-typed with respect to a type A if $\cdot \vdash M : A$ is provable using the typing rules in Definition 3.

Definition 3 (Typing rules for λ^{cont})**.**

$$\frac{}{\Gamma, x : A \vdash x : A}\ \mathsf{Tvar} \qquad \frac{\texttt{CONST-TYPE}(p) = P}{\Gamma \vdash p : P}\ \mathsf{Tconst} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \mathsf{Tlam}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}\ \mathsf{Tapp} \qquad \frac{\Gamma, x : A\ \mathsf{cont} \vdash M : A}{\Gamma \vdash \mathsf{callcc}\ x.M : A}\ \mathsf{Tcallcc}$$

$$\frac{\Gamma \vdash M : A\ \mathsf{cont} \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{throw}\ M\ N : C}\ \mathsf{Tthrow}$$

The above typing rules are standard [3]. In the rule Tconst, `CONST-TYPE` is a predefined mapping from constants to primitive types. In the rule Tcallcc, callcc $x.M$ is of type A if M is also of type A assuming $x : A$ cont. Please note that x will at runtime be bound to a continuation that expects the result of M. Therefore, callcc itself can be given a type $(A\ \mathsf{cont} \to A) \to A$ for any type A, which corresponds to Peirce's law. In the rule Tthrow, if M and N are of type A cont and A, respectively, then throw $M\ N$ can be assigned an arbitrary type C because its reduction never returns to the current evaluation context. Therefore, the rule Tthrow corresponds to the law of non-contradiction (followed by false elimination).

The type system given in Definition 3 is sound with respect to the operational semantics given in Definition 2 in that well-typed expressions never go wrong, i.e., cannot cause type errors during evaluation [17].

2.2. A Call-by-Value CPS Transformation

The CBV CPS transformation [3] translates the source language λ^{cont} into the target language $\lambda^{\to}$ without callcc and throw. The abstract syntax of $\lambda^{\to}$ is defined as follows.

Definition 4 (Abstract syntax of $\lambda^{\to}$)**.**

$$\begin{array}{rrcl} \textit{expressions} & M, N & ::= & p \mid x \mid \lambda x.M \mid M\,N \\ \textit{types} & A, B, C & ::= & P \mid A \to B \mid \bot \\ \textit{values} & V & ::= & p \mid \lambda x.M \end{array}$$

The main idea of the transformation is twofold. First, it interprets a continuation of type A cont as a function of type $A \to \bot$, where we use $\bot$ to denote the type of final answers. Second, it translates every expression M into a CPS function that explicitly takes as argument a continuation k expecting the result of M and applies k to the result in its body. Formally, the CBV CPS transformation associates each type A with A^*, which is the type of values of type A after the CPS transformation and defined in Definition 5. Based on this, it translates each expression M of type A into a CPS expression $\mathcal{C}[\![M]\!]$ of type $(A^* \to \bot) \to \bot$, which is defined in Definition 6.

Definition 5 (CBV type translation)**.**

$$\begin{array}{rcl} P^* & = & P \\ (A \to B)^* & = & A^* \to ((B^* \to \bot) \to \bot) \\ (A\ \mathsf{cont})^* & = & A^* \to \bot \end{array}$$

Definition 6 (CBV CPS transformation)**.**

$$\begin{array}{rcl} \mathcal{C}[\![p]\!] & = & \lambda k.k\,p \\ \mathcal{C}[\![x]\!] & = & \lambda k.k\,x \\ \mathcal{C}[\![\lambda x.M]\!] & = & \lambda k.k\,(\lambda x.\mathcal{C}[\![M]\!]) \\ \mathcal{C}[\![M\,N]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\,(\lambda f.\mathcal{C}[\![N]\!]\,(\lambda v.f\,v\,k)) \\ \mathcal{C}[\![\mathsf{callcc}\ x.M]\!] & = & \lambda k.(\mathcal{C}[\![M]\!][k/x])\,k \\ \mathcal{C}[\![\mathsf{throw}\ M\ N]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\,(\lambda k'.\mathcal{C}[\![N]\!]\,k') \end{array}$$

The transformation simply translates values and variables into the CPS form. Please note that in the translation of a lambda abstraction, the body is also translated into a CPS expression. For a lambda application $M\,N$, let us first analyze the type structure of the transformed expression:

In the transformed expression $\lambda k : B^* \to \bot.\mathcal{C}[\![M]\!]\,(\lambda f : (A \to B)^*.\mathcal{C}[\![N]\!]\,(\lambda v : A^*.f\,v\,k))$, $\mathcal{C}[\![M]\!]$ will bind the CPS-transformed evaluation result of M to f and $\mathcal{C}[\![N]\!]$ will bind that of N to v. Then the function f is called with the argument v and the initial continuation k as $f\,v\,k$, which returns an answer of type $\bot$. In the translation of $\mathsf{callcc}\ x.M$, $\mathcal{C}[\![M]\!]$ of type $(A^* \to \bot) \to \bot$ will apply the initial continuation k of type $A^* \to \bot$ to the CPS-transformed evaluation result of M, which is of type A^*. Moreover, the continuation variable x of type A cont is substituted with the explicit continuation k in $\mathcal{C}[\![M]\!]$. Finally, in the translation of $\mathsf{throw}\ M\ N$, $\mathcal{C}[\![M]\!]$ will bind the continuation evaluated from M to k', and then $\mathcal{C}[\![N]\!]$ will apply k' to the CPS-transformed evaluation result of N. Please note that the initial continuation k is not used.

The above transformation preserves the operational meaning of the source program of primitive type: when applied to an initial continuation (an identity function), the translated program terminates if and only if the source program does, and in that case they evaluate to the same value of primitive type [3,16].

Theorem 1. *If M is a closed expression of type P, then*

$$M \mapsto^* V \quad \Longleftrightarrow \quad \mathcal{C}[\![M]\!]\,(\lambda x.x) \mapsto^* V$$

where $\mapsto^*$ is the reflexive and transitive closure of one-step reduction $\mapsto$.

The transformation also preserves typing [3], i.e., provability, as stated below, where Γ^* is the typing context such that $x : A^* \in \Gamma^*$ if and only if $x : A \in \Gamma$.

Theorem 2. *If $\Gamma \vdash M : A$, then $\Gamma^* \vdash \mathcal{C}[\![M]\!] : (A^* \to \bot) \to \bot$.*

Proof. By induction on the structure of M. □

3. A Double Negation Translation

By the Curry-Howard isomorphism, types and expressions in λ^{cont} correspond to formulas and proofs in classical implicational propositional logic (CPL) [4]. The formulas of CPL are defined as follows.

Definition 7 (CPL formulas)**.**

$$\textit{formulas} \quad A, B, C \quad ::= \quad P \mid A \supset B \mid \neg A$$

P ranges over atomic propositions. We choose only implication $\supset$ and negation $\neg$ as primitive connectives to clearly show the connection between λ^{cont} and CPL. Other connectives $A \vee B$ and $A \wedge B$ can be notionally defined as $\neg A \supset B$ and $\neg(A \supset \neg B)$, respectively. A continuation type A cont in λ^{cont} is interpreted as a negated formula $\neg A$ in CPL.

The type system for λ^{cont} given in Definition 3 can be interpreted as the proof system for CPL given in Definition 8. We use a hypothetical judgment $\Gamma \vdash_{\mathsf{K}} A$ to mean that a proposition A is true under a set Γ of hypotheses (where the subscript K stands for *K*lassical). Below, by removing the terms in each typing rule in Definition 3, we obtain the corresponding inference rule for CPL. (For simplicity, we omit the rule corresponding to Tconst because it is not necessary for studying the Curry-Howard isomorphism between CPL and IPL. However, it is needed for proving the observational soundness for the CPS transformation, stated in Theorem 1.)

Definition 8 (Proof system for CPL)**.**

$$\frac{}{\Gamma, A \vdash_{\mathsf{K}} A}\ \mathsf{Hyp} \qquad \frac{\Gamma, A \vdash_{\mathsf{K}} B}{\Gamma \vdash_{\mathsf{K}} A \supset B}\ \supset\mathsf{I} \qquad \frac{\Gamma \vdash_{\mathsf{K}} A \supset B \qquad \Gamma \vdash_{\mathsf{K}} A}{\Gamma \vdash_{\mathsf{K}} B}\ \supset\mathsf{E}$$

$$\frac{\Gamma, \neg A \vdash_{\mathsf{K}} A}{\Gamma \vdash_{\mathsf{K}} A}\ \mathsf{Peirce} \qquad \frac{\Gamma \vdash_{\mathsf{K}} \neg A \qquad \Gamma \vdash_{\mathsf{K}} A}{\Gamma \vdash_{\mathsf{K}} C}\ \mathsf{Contra}$$

The typing rules Tcallcc and Tthrow for control operators correspond to the rules Peirce and Contra which internalize Peirce's law and the law of non-contradiction, respectively. Using them, we can derive the following rules for double negation elimination and the law of excluded middle.

$$\frac{}{\Gamma \vdash_{\mathsf{K}} \neg\neg A \supset A}\ \mathsf{DNE} \qquad \frac{}{\Gamma \vdash_{\mathsf{K}} A \vee \neg A}\ \mathsf{EM}$$

The DNT corresponding to the CBV CPS transformation translates CPL into intuitionistic implicational propositional logic (IPL) which corresponds to $\lambda^{\to}$ in Definition 4. The formulas of IPL are defined as follows.

Definition 9 (IPL formulas)**.**

$$\textit{formulas} \quad A, B, C \quad ::= \quad P \mid A \supset B \mid \bot$$

A natural deduction system for IPL is defined below. We use a judgment $\Gamma \vdash_{\mathsf{I}} A$ to mean that a proposition A is true in IPL under a set Γ of hypotheses (where the subscript I stands for *I*ntuitionistic).

Definition 10 (Proof system for IPL)**.**

$$\frac{}{\Gamma, A \vdash_{\mathsf{I}} A}\ \mathsf{Hyp} \qquad \frac{\Gamma, A \vdash_{\mathsf{I}} B}{\Gamma \vdash_{\mathsf{I}} A \supset B}\ \supset\mathsf{I} \qquad \frac{\Gamma \vdash_{\mathsf{I}} A \supset B \qquad \Gamma \vdash_{\mathsf{I}} A}{\Gamma \vdash_{\mathsf{I}} B}\ \supset\mathsf{E} \qquad \frac{\Gamma \vdash_{\mathsf{I}} \bot}{\Gamma \vdash_{\mathsf{I}} C}\ \bot\mathsf{E}$$

Although we include the $\bot$E rule in Definition 10 for the completeness of the system, it is never used in the translation of CPL proof trees into IPL ones. In other words, our study is still valid without the $\bot$E rule. Please note that $\bot$ is not even included in CPL (although it can be encoded, for example, as $A \wedge \neg A$). Meanwhile, we regard $\neg A$ as a syntactic abbreviation of $A \supset \bot$ in IPL. Therefore, the following rules are special cases of the $\supset$ introduction and elimination rules.

$$\frac{\Gamma, A \vdash_{\mathsf{I}} \bot}{\Gamma \vdash_{\mathsf{I}} \neg A}\ \neg\mathsf{I} \qquad \frac{\Gamma \vdash_{\mathsf{I}} \neg A \qquad \Gamma \vdash_{\mathsf{I}} A}{\Gamma \vdash_{\mathsf{I}} \bot}\ \neg\mathsf{E}$$

We define a translation of CPL formulas into IPL formulas exactly like the CBV CPS transformation of types, as follows.

Definition 11 (DNT for formulas)**.**

$$\begin{array}{rcll} P^* & = & P & \\ (A \supset B)^* & = & A^* \supset ((B^* \supset \bot) \supset \bot) & \equiv \quad A^* \supset \neg\neg B^* \\ (\neg A)^* & = & A^* \supset \bot & \end{array}$$

Our translation slightly differs from the double negation translations proposed by Kolmogorov [9], Gödel and Gentzen [6,8], and Kuroda [10]. Kolmogorov's translation directly corresponds to the call-by-name CPS transformation [4,12,18] via the Curry-Howard isomorphism. In particular, in his translation, $A \to B$ is translated into $\neg\neg A^* \supset \neg\neg B^*$.

The correctness of the DNT is stated as follows, where Γ^* applies the translation elementwise: $A^* \in \Gamma^*$ if and only if $A \in \Gamma$.

Theorem 3. *If* $\Gamma \vdash_{\mathsf{K}} A$ *is provable, then* $\Gamma^* \vdash_{\mathsf{I}} \neg\neg A^*$ *is also provable.*

Proof. We define a recursive function that maps a derivation of $\Gamma \vdash_{\mathsf{K}} A$ into a derivation of $\Gamma^* \vdash_{\mathsf{I}} \neg\neg A^*$. In particular, for each case, we build a derivation tree corresponding to its counterpart CPS transformation in Definition 6 (although a smaller derivation may be constructed). □

4. A Call-by-Value Selective CPS Transformation

In this section, we review Nielsen's CBV selective CPS transformation [16]. The main idea is to distinguish trivial expressions whose evaluation does not give rise to computational effects (i.e., evaluation of callcc or throw) from nontrivial ones whose evaluation may give rise to effects. In general, it is undecidable to determine if the evaluation of an expression gives rise to effects, for example, in the presence of a recursive operator or recursive types. Since it is safe to assume that every expression might have an effect, the usual CPS transformation assumes every expression as nontrivial and translates into the CPS form.

To translate only nontrivial expressions selectively, while keeping trivial ones in the direct style, we use a type and effect system, where (definitely) trivial expressions are annotated with $\mathtt{T}$ while (possibly but not necessarily) nontrivial ones with $\mathtt{N}$. The following is the minimally annotated syntax of the source language λ^{cont}, where we omit annotations when clear from the structure of the given expression.

Definition 12 (Annotated syntax)**.**

$$\begin{array}{rrcll} \textit{annotations} & \mathtt{a} & ::= & \mathtt{T} \mid \mathtt{N} & \textit{where } \mathtt{N} < \mathtt{T} \\ \textit{expressions} & M, N & ::= & p \mid x \mid \lambda^{\mathtt{a}} x.M \mid (M \mathbin{@^{\mathtt{a}}} N)^{\mathtt{a}} \mid \mathsf{callcc}\ x.M \mid \mathsf{throw}\ M\ N \\ \textit{types} & A, B, C & ::= & P \mid A \xrightarrow{\mathtt{a}} B \mid A\ \mathsf{cont} \end{array}$$

We understand values and identifiers to always be implicitly annotated with $\mathtt{T}$ and control expressions with $\mathtt{N}$, and thus do not explicitly mark them in the syntax. Moreover, we denote an application by $M \mathbin{@} N$ instead of $M\,N$ for specifying annotations. Still, when annotations do not matter, we use the usual notation $M\,N$. Special care should be taken for abstractions and applications. For an application of the form $(\lambda x.M) \mathbin{@} V$, if the function body M is nontrivial, the evaluation of its β-reduct $M[V/x]$ gives rise to an effect. For such applications, we annotate @ with $\mathtt{N}$ as in $(\lambda x.M) \mathbin{@^{\mathtt{N}}} V$. An application $M \mathbin{@} N$ is annotated with $\mathtt{N}$, i.e., $(M \mathbin{@} N)^{\mathtt{N}}$, if the evaluation of M, N, or its β-reduct (obtained after both M and N are reduced to values) gives rise to an effect. Furthermore, in the presence of higher-order functions, different abstractions may be applied at the same application point (e.g., M and N in $(\lambda f.\mathsf{fst}\,(f\,M)\,(f\,N))\,(\lambda g.g\,p)$ where $\mathsf{fst} = \lambda x.\lambda y.x$ and p is a some constant), and such abstractions should be transformed into the same style. Therefore, some abstraction with a trivial body might sometimes need to be transformed into the CPS form. For such abstractions, we annotate λ with $\mathtt{N}$ as in $\lambda^{\mathtt{N}} x.M$. Correspondingly, if $\lambda x.M$ is of type $A \to B$, then $\lambda^{\mathtt{N}} x.M$ and $\lambda^{\mathtt{T}} x.M$ respectively have an annotated type $A \xrightarrow{\mathtt{N}} B$ and $A \xrightarrow{\mathtt{T}} B$.

We say that annotations are consistent (with respect to the program's behavior) if

- every expression annotated with $\mathtt{T}$ is indeed trivial,
- every abstraction annotated with $\mathtt{T}$ (resp. $\mathtt{N}$) is applied only at application points annotated with $\mathtt{T}$ (resp. $\mathtt{N}$), and
- every abstraction whose body is annotated with $\mathtt{N}$ is also annotated with $\mathtt{N}$.

To check the consistency of annotations, we use a type and effect system using a judgment $\Gamma \vdash M : A,\ \mathtt{a}$ which means that expression M is of type A and can be annotated with $\mathtt{a}$ under typing context Γ.

Definition 13 (Type and effect system)**.**

$$\frac{}{\Gamma, x : A \vdash x : A,\ \mathtt{T}}\ \mathsf{Tvar} \qquad \frac{\mathtt{CONST\text{-}TYPE}(p) = P}{\Gamma \vdash p : P,\ \mathtt{T}}\ \mathsf{Tconst}$$

$$\frac{\Gamma, x : A \vdash M : B,\ \mathtt{a}_1 \qquad \mathtt{a} \leq \mathtt{a}_1}{\Gamma \vdash \lambda^{\mathtt{a}} x.M : A \xrightarrow{\mathtt{a}} B,\ \mathtt{T}}\ \mathsf{Tlam}$$

$$\frac{\Gamma \vdash M : A \xrightarrow{\mathtt{a}_3} B,\ \mathtt{a}_1 \qquad \Gamma \vdash N : A,\ \mathtt{a}_2 \qquad \mathtt{a} \leq \min(\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3)}{\Gamma \vdash (M @^{\mathtt{a}_3} N)^{\mathtt{a}} : B,\ \mathtt{a}}\ \mathsf{Tapp}$$

$$\frac{\Gamma, x : A\ \mathsf{cont} \vdash M : A,\ \mathtt{a}}{\Gamma \vdash \mathsf{callcc}\ x.M : A,\ \mathtt{N}}\ \mathsf{Tcallcc} \qquad \frac{\Gamma \vdash M : A\ \mathsf{cont},\ \mathtt{a}_1 \qquad \Gamma \vdash N : A,\ \mathtt{a}_2}{\Gamma \vdash \mathsf{throw}\ M\ N : C,\ \mathtt{N}}\ \mathsf{Tthrow}$$

The relation $\mathtt{a} \leq \mathtt{a}_1$ in the rule Tlam allows us to annotate an abstraction with a trivial body with N. Similarly, in the rule Tapp, even if $\mathtt{a}_1$, $\mathtt{a}_2$, and $\mathtt{a}_3$ are all T, the application can be annotated with N. From now on, we consider only well-annotated expressions which are allowed by the type and effect system, i.e., typable by Definition 13. The following theorem shows that the type and effect system indeed ensures the consistency of annotations. As a corollary, if a closed expression M can be annotated with T, the evaluation of M does not give rise to an effect.

Theorem 4. *If* $\Gamma \vdash M : A,\ \mathtt{a}$*, annotations in* M *are consistent.*

Proof. By induction on a derivation of $\Gamma \vdash M : A,\ \mathtt{a}$. □

Corollary 1. *Given a closed expression* M*, if* $\cdot \vdash M : A,\ \mathtt{T}$*, no effects occur during the evaluation of* M*.*

The CBV selective CPS transformation associates each type A with $A^\star$ defined below. The main difference from Definition 5 is the translation of function types, where a trivial function type $A \xrightarrow{\mathtt{T}} B$ is kept in the direct style.

Definition 14 (Selective type translation)**.**

$$\begin{array}{rcl} P^\star & = & P \\ (A \xrightarrow{\mathtt{N}} B)^\star & = & A^\star \to ((B^\star \to \bot) \to \bot) \\ (A \xrightarrow{\mathtt{T}} B)^\star & = & A^\star \to B^\star \\ A\ \mathsf{cont}^\star & = & A^\star \to \bot \end{array}$$

In order to transform expressions selectively, we use two functions: $\mathcal{S}[\![-]\!]$ translates expressions of type A into those of type $(A^\star \to \bot) \to \bot$ and $\mathcal{S}_t[\![-]\!]$ translates expressions of type A into those of type $A^\star$. Given an expression, the former translates it into a CPS expression, whereas the latter keeps it in the direct style. In particular, $\mathcal{S}_t[\![-]\!]$ takes only trivial expressions.

Definition 15 (Selective CPS transformation)**.**

$$\begin{array}{rcl}
\mathcal{S}[\![M^{\mathtt{T}}]\!] & = & \lambda k.k\,\mathcal{S}_t[\![M]\!] \\
\mathcal{S}[\![(M\,@^{\mathtt{N}}\,N)^{\mathtt{N}}]\!] & = & \lambda k.\mathcal{S}[\![M]\!]\,(\lambda f.\mathcal{S}[\![N]\!]\,(\lambda v.f\,v\,k)) \\
\mathcal{S}[\![(M\,@^{\mathtt{T}}\,N)^{\mathtt{N}}]\!] & = & \lambda k.\mathcal{S}[\![M]\!]\,(\lambda f.\mathcal{S}[\![N]\!]\,(\lambda v.k\,(f\,v))) \\
\mathcal{S}[\![\mathsf{callcc}\ x.M]\!] & = & \lambda k.(\mathcal{S}[\![M]\!][k/x])\,k \\
\mathcal{S}[\![\mathsf{throw}\ M\,N]\!] & = & \lambda k.\mathcal{S}[\![M]\!]\,(\lambda k'.\mathcal{S}[\![N]\!]\,k') \\
 & & \\
\mathcal{S}_t[\![x]\!] & = & x \\
\mathcal{S}_t[\![p]\!] & = & p \\
\mathcal{S}_t[\![\lambda^{\mathtt{N}}x.M]\!] & = & \lambda x.\mathcal{S}[\![M]\!] \\
\mathcal{S}_t[\![\lambda^{\mathtt{T}}x.M]\!] & = & \lambda x.\mathcal{S}_t[\![M]\!] \\
\mathcal{S}[\![(M\,@^{\mathtt{T}}\,N)^{\mathtt{T}}]\!] & = & \mathcal{S}_t[\![M]\!]\,\mathcal{S}_t[\![N]\!]
\end{array}$$

The main difference between Definitions 6 and 15 is the transformation of functions and applications annotated with $\mathtt{T}$. In the translation of a function $\lambda^{\mathtt{T}}x.M$, the body M is kept in the direct style, i.e., translated using $\mathcal{S}_t[\![-]\!]$. Similarly, in the translation of an application $(M\,@^{\mathtt{T}}\,N)^{\mathtt{T}}$, both M and N are kept in the direct style. Finally, for an application $(M\,@^{\mathtt{T}}\,N)^{\mathtt{N}}$, the annotation $\mathtt{T}$ for $@$ indicates that M is of type $A \xrightarrow{\mathtt{T}} B$ for some types A and B. In other words, if M evaluates to an abstraction $\lambda x.M'$, then M' would be trivial. Therefore, in the transformed expression, $\mathcal{S}[\![M]\!]$ is applied to a continuation of type $(A \xrightarrow{\mathtt{T}} B)^\star \to \bot$. Specifically, at runtime, f will be bound to the result evaluated from M, which is of type $A^\star \to B^\star$, and $f\,v$ will evaluate to a value of type $B^\star$, which is in the direct style. Hence, we apply to $f\,v$ a continuation k that expects the result of $M\,N$.

The operational meaning of the source program of primitive type is preserved by the selective CPS transformation as stated below [16].

Theorem 5. *If M is a closed and well-annotated expression of type P, then*

$$M \mapsto^* V \quad \Longleftrightarrow \quad \mathcal{S}[\![M]\!]\,(\lambda x.x) \mapsto^* V.$$

The transformation also preserves typing, as stated below, where $\Gamma^\star$ is the typing context such that $x : A^\star \in \Gamma^\star$ if and only if $x : A \in \Gamma$.

Theorem 6 (Preservation of typing)**.**

1. *If* $\Gamma \vdash M^{\mathtt{a}} : A$, $\mathtt{a}$, *then* $\Gamma^\star \vdash \mathcal{S}[\![M]\!] : (A^\star \to \bot) \to \bot$.
2. *If* $\Gamma \vdash M^{\mathtt{T}} : A$, $\mathtt{T}$, *then* $\Gamma^\star \vdash \mathcal{S}_t[\![M]\!] : A^\star$.

Proof. By simultaneous induction on derivations. □

5. A Selective Double Negation Translation

In this section, we present an annotated proof system for CPL, which corresponds to the type and effect system given in Definition 13 via the Curry-Howard isomorphism. It keeps track of the uses of the Peirce and Contra rules using annotations. Based on the annotations in the proof, our selective DNT translates only the parts of the proof which use the Peirce or Contra rules into the double negation form.

We use annotated formulas defined below, where annotations are defined as in Definition 12.

Definition 16 (Annotated CPL formulas)**.**

$$\textit{formulas} \quad A, B, C \quad ::= \quad P \mid A \supset^{\mathtt{a}} B \mid \neg A$$

A formula $A \supset^{\mathtt{N}} B$ means that assuming A is provable, B is also provable under the assumption A, (possibly) using the Peirce or Contra rules. In contrast, a formula $A \supset^{\mathtt{T}} B$ means that assuming A is provable, B is also provable but without using those rules.

The proof system for annotated formulas is obtained from the type and effect system by removing the terms in each typing rule. (As in Definition 8, we omit the rule corresponding to Tconst for simplicity.) We use a judgment $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}$ which means that A is true under a set Γ of hypotheses, and its proof does not use the Peirce or Contra rules if $\mathtt{a}$ is $\mathtt{T}$; otherwise, the proof may use them.

Definition 17 (Annotated proof system)**.**

$$\frac{}{\Gamma, A \vdash_{\mathsf{K}} A,\ \mathtt{T}}\ \mathsf{Hyp} \qquad \frac{\Gamma, A \vdash_{\mathsf{K}} B,\ \mathtt{a}_1 \qquad \mathtt{a} \leq \mathtt{a}_1}{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{a}} B,\ \mathtt{T}}\ \supset \mathsf{I} \qquad \frac{\Gamma, \neg A \vdash_{\mathsf{K}} A,\ \mathtt{a}}{\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{N}}\ \mathsf{Peirce}$$

$$\frac{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{a}_3} B,\ \mathtt{a}_1 \qquad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2 \qquad \mathtt{a} \leq \min(\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3)}{\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{a}}\ \supset \mathsf{E}$$

$$\frac{\Gamma \vdash_{\mathsf{K}} \neg A,\ \mathtt{a}_1 \qquad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2}{\Gamma \vdash_{\mathsf{K}} C,\ \mathtt{N}}\ \mathsf{Contra}$$

The interpretation of the annotations in the rules $\supset$ I and $\supset$ E is similar to the one for the rules Tlam and Tapp in Definition 13. In the rule $\supset$ I, even if B is proved without using the Peirce or Contra rules, we may annotate the implication with $\mathtt{N}$ as if B is proved using them. In the rule $\supset$ E, if all of $\mathtt{a}_1$, $\mathtt{a}_2$, and $\mathtt{a}_3$ are $\mathtt{T}$, then it means that the derivation of $\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{a}$ does not use the Peirce or Contra rules. Nevertheless, it can be annotated with $\mathtt{N}$ as in the type and effect system. The annotated proof system is equivalent to the original proof system in the following sense.

Theorem 7 (Soundness and completeness)**.**

1. *If* $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}$ *is provable, then* $\mathit{erase}(\Gamma) \vdash_{\mathsf{K}} \mathit{erase}(A)$ *is also provable, where the function* erase *removes every annotation in the given context and formula.*
2. *If* $\Gamma \vdash_{\mathsf{K}} A$ *is provable, then there exists at least one annotation for* A *that is provable in the annotated proof system.*

Proof. (1) By straightforward induction on a derivation of $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}$. (2) The one where every implication is marked with $\mathtt{N}$ is always provable. □

We can further extend the completeness statement such that there exists a maximal annotation with as many $\mathtt{T}$'s as possible. We now define a selective translation of annotated CPL formulas into IPL formulas as follows, which directly corresponds to the selective type translation in Definition 14.

Definition 18 (Selective DNT for formulas)**.**

$$\begin{array}{rcll} P^\star & = & P & \\ (A \supset^{\mathtt{N}} B)^\star & = & A^\star \supset ((B^\star \supset \bot) \supset \bot) & \equiv \quad A^\star \supset (\neg\neg B^\star) \\ (A \supset^{\mathtt{T}} B)^\star & = & A^\star \supset B^\star & \\ \neg A^\star & = & A^\star \supset \bot & \end{array}$$

The correctness of our selective DNT is stated as follows, where $\Gamma^\star$ applies the translation elementwise: $A^\star \in \Gamma^\star$ if and only if $A \in \Gamma$.

Theorem 8 (Correctness)**.**

1. *If* $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}$ *is provable, then* $\Gamma^\star \vdash_{\mathsf{I}} \neg\neg A^\star$ *is also provable.*
2. *If* $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}$ *is provable, then* $\Gamma^\star \vdash_{\mathsf{I}} A^\star$ *is also provable.*

Proof. We define two mutually recursive functions f and g which translate derivations of $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}$ and $\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}$ into derivations of $\Gamma^\star \vdash_{\mathsf{I}} \neg\neg A^\star$ and $\Gamma^\star \vdash_{\mathsf{I}} A^\star$, respectively. Their definitions directly correspond to the selective CPS transformation $\mathcal{S}[\![-]\!]$ and $\mathcal{S}_t[\![-]\!]$ in

Definition 15. In the translations below, whenever necessary, we implicitly apply weakening on the induction hypothesis, i.e., the result of a recursive call.

1. $$f\big(\,\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}\,\big) = \dfrac{\dfrac{}{\Gamma^\star, \neg A^\star \vdash_{\mathsf{I}} \neg A^\star}\ \mathsf{Hyp} \qquad g(\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T})}{\dfrac{\Gamma^\star, \neg A^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg A^\star}\ \supset\mathsf{I}}\ \supset\mathsf{E}$$

2. $$f\left(\dfrac{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{a}_1 \quad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2}{\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{N}}\ \supset\mathsf{E}\right) =$$

$$\mathtt{let}\ \begin{cases} \Gamma_1^\star = \Gamma^\star, \neg B^\star \\ \Gamma_2^\star = \Gamma_1^\star, A^\star \supset B^\star \\ \Gamma_3^\star = \Gamma_2^\star, A^\star \end{cases}$$

$$\mathtt{and}\ \mathcal{D} = \begin{cases} \dfrac{\dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} \neg B^\star}\ \mathsf{Hyp} \quad \dfrac{\dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} A^\star \supset B^\star}\ \mathsf{Hyp} \quad \dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} A^\star}\ \mathsf{Hyp}}{\Gamma_3^\star \vdash_{\mathsf{I}} B^\star}\ \supset\mathsf{E}}{\Gamma_3^\star \vdash_{\mathsf{I}} \bot}\ \supset\mathsf{E} \end{cases}$$

$$\mathtt{in}$$

$$\dfrac{\dfrac{f(\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_1) \qquad \dfrac{\dfrac{\mathcal{D}}{\Gamma_2^\star \vdash_{\mathsf{I}} \neg A^\star}\ \supset\mathsf{I}}{\Gamma_2^\star \vdash_{\mathsf{I}} \bot}\ \supset\mathsf{E}}{\Gamma_1^\star \vdash_{\mathsf{I}} \neg(A^\star \supset B^\star)}\ \supset\mathsf{I}}{\dfrac{\Gamma_1^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg B^\star}\ \supset\mathsf{I}}\ \supset\mathsf{E}$$

3. $$f\left(\dfrac{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{N}} B,\ \mathtt{a}_1 \quad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2}{\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{N}}\ \supset\mathsf{E}\right) =$$

$$\mathtt{let}\ \begin{cases} \Gamma_1^\star = \Gamma^\star, \neg B^\star \\ \Gamma_2^\star = \Gamma_1^\star, A^\star \supset \neg\neg B^\star \\ \Gamma_3^\star = \Gamma_2^\star, A^\star \end{cases}$$

$$\mathtt{and}\ \mathcal{E} = \begin{cases} \dfrac{\dfrac{\dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} A^\star \supset \neg\neg B^\star}\ \mathsf{Hyp} \quad \dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} A^\star}\ \mathsf{Hyp}}{\Gamma_3^\star \vdash_{\mathsf{I}} \neg\neg B^\star}\ \supset\mathsf{E} \quad \dfrac{}{\Gamma_3^\star \vdash_{\mathsf{I}} \neg B^\star}\ \mathsf{Hyp}}{\Gamma_3^\star \vdash_{\mathsf{I}} \bot}\ \supset\mathsf{E} \end{cases}$$

$$\mathtt{in}$$

$$\dfrac{\dfrac{f(\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_1) \qquad \dfrac{\dfrac{\mathcal{E}}{\Gamma_2^\star \vdash_{\mathsf{I}} \neg A^\star}\ \supset\mathsf{I}}{\Gamma_2^\star \vdash_{\mathsf{I}} \bot}\ \supset\mathsf{E}}{\Gamma_1^\star \vdash_{\mathsf{I}} \neg(A^\star \supset \neg\neg B^\star)}\ \supset\mathsf{I}}{\dfrac{\Gamma_1^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg B^\star}\ \supset\mathsf{I}}\ \supset\mathsf{E}$$

4. $$f\left(\dfrac{\Gamma, \neg A \vdash_{\mathsf{K}} A,\ \mathtt{a}}{\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{N}}\ \mathsf{Peirce}\right) = \dfrac{f(\Gamma, \neg A \vdash_{\mathsf{K}} A,\ \mathtt{a}) \quad \dfrac{}{\Gamma^\star, \neg A^\star \vdash_{\mathsf{I}} \neg A^\star}\ \mathsf{Hyp}}{\dfrac{\Gamma^\star, \neg A^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg A^\star}\ \supset\mathsf{I}}\ \supset\mathsf{E}$$

5. $$f\left(\dfrac{\Gamma \vdash_{\mathsf{K}} \neg A,\ \mathtt{a}_1 \quad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2}{\Gamma \vdash_{\mathsf{K}} C,\ \mathtt{N}}\ \mathsf{Contra}\right) =$$

$$\dfrac{\dfrac{f(\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}_2) \qquad \dfrac{\dfrac{}{\Gamma^\star, \neg C^\star, \neg A^\star \vdash_{\mathsf{I}} \neg A^\star}\ \mathsf{Hyp}}{\Gamma^\star, \neg C^\star, \neg A^\star \vdash_{\mathsf{I}} \bot}\ \supset\mathsf{E}}{\Gamma^\star, \neg C^\star \vdash_{\mathsf{I}} \neg\neg A^\star}\ \supset\mathsf{I}}{\dfrac{f(\Gamma \vdash_{\mathsf{K}} \neg A,\ \mathtt{a}_1) \qquad \Gamma^\star, \neg C^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg C^\star}\ \supset\mathsf{I}}\ \supset\mathsf{E}$$

6. $$g\left(\dfrac{}{\Gamma, A \vdash_{\mathsf{K}} A,\ \mathtt{T}}\ \mathsf{Hyp}\right) = \dfrac{}{\Gamma^\star, A^\star \vdash_{\mathsf{I}} A^\star}\ \mathsf{Hyp}$$

7. $$g\left(\dfrac{\Gamma, A \vdash_{\mathsf{K}} B,\ \mathtt{T}}{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{T}}\ \supset \mathsf{I}\right) = \dfrac{g(\Gamma, A \vdash_{\mathsf{K}} B,\ \mathtt{T})}{\Gamma^\star \vdash_{\mathsf{I}} A^\star \supset B^\star}\ \supset \mathsf{I}$$
8. $$g\left(\dfrac{\Gamma, A \vdash_{\mathsf{K}} B,\ \mathtt{a}}{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{N}} B,\ \mathtt{T}}\ \supset \mathsf{I}\right) = \dfrac{f(\Gamma, A \vdash_{\mathsf{K}} B,\ \mathtt{a})}{\Gamma^\star \vdash_{\mathsf{I}} A^\star \supset \neg\neg B^\star}\ \supset \mathsf{I}$$
9. $$g\left(\dfrac{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{T} \quad \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}}{\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{T}}\ \supset \mathsf{E}\right) = \dfrac{g(\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{T}) \quad g(\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T})}{\Gamma^\star \vdash_{\mathsf{I}} B^\star}\ \supset \mathsf{E}$$

In the above translations, whereas f eliminates the instances of the rules Peirce and Contra used in the given derivation by double-negating the goal formula, g keeps the given derivation as it is since it is also valid in IPL. □

The following diagram summarizes the relationship between the selective CPS transformation and the selective DNT. We note that the selective CPS transformation can be derived from the selective DNT by assigning proof terms to each inference rule.

$$\begin{array}{ccc}
\left\{\begin{array}{l}\Gamma \vdash M : A,\ \mathtt{a}\\ \Gamma \vdash M : A,\ \mathtt{T}\end{array}\right. & \Longleftrightarrow & \left\{\begin{array}{l}\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{a}\\ \Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}\end{array}\right. \text{ in CPL}\\
\Big\downarrow \text{ Selective CPS transformation} & \text{Curry-Howard isomorphic} & \Big\downarrow \text{ Selective DNT}\\
\left\{\begin{array}{l}\Gamma^\star \vdash \mathcal{S}[\![M]\!] : (A^\star \to \bot) \to \bot\\ \Gamma^\star \vdash \mathcal{S}_t[\![M]\!] : A^\star\end{array}\right. & \Longleftrightarrow & \left\{\begin{array}{l}\Gamma^\star \vdash_{\mathsf{I}} \neg\neg A^\star\\ \Gamma^\star \vdash_{\mathsf{I}} A^\star\end{array}\right. \text{ in IPL}
\end{array}$$

6. Discussion and Conclusions

In this paper, we studied the logical meaning of a selective CPS transformation [16]. Although we considered only a CBV semantics, our analysis can also be applied to a call-by-name (CBN) CPS transformation [4,12,18]. In a CBN semantics, a lambda application $(\lambda x.M)\ N$ reduces to $M[N/x]$ regardless of whether N is a value or not. Therefore, under the CBN CPS transformation, a translated function is passed an argument in the CPS form, and thus a function type $A \to B$ is translated into $((A^* \to \bot) \to \bot) \to (B^* \to \bot) \to \bot$. The corresponding DNT, such as Kolmogorov's translation [9], translates $A \supset B$ into $\neg\neg A^* \supset \neg\neg B^*$, and we can exploit our annotated proof system, presented in Section 5, to derive a CBN selective DNT.

While our selective DNT is a direct logical interpretation of Nielsen's selective CPS transformation [16], there is still room for further improvement. First, unlike the rule Peirce, the rule Contra is valid not only in classical logic but also in intuitionistic logic, and therefore we may not need to translate the instances of Contra into the double negation form. We choose to translate them in order to keep the direct correspondence between the CPS transformation and the DNT. Second, by further analyzing annotations, we can produce a smaller proof. For example, the following derivation

$$\dfrac{\overset{\mathcal{D}}{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{T}} \quad \overset{\mathcal{E}}{\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}}}{\Gamma \vdash_{\mathsf{K}} B,\ \mathtt{N}}\ \supset \mathsf{E}$$

can be translated into

$$\dfrac{\dfrac{}{\Gamma^\star, \neg B^\star \vdash_{\mathsf{I}} \neg B^\star}\ \mathsf{Hyp} \quad \dfrac{g\left(\overset{\mathcal{D}}{\Gamma \vdash_{\mathsf{K}} A \supset^{\mathtt{T}} B,\ \mathtt{T}}\right) \quad g\left(\overset{\mathcal{E}}{\Gamma \vdash_{\mathsf{K}} A,\ \mathtt{T}}\right)}{\Gamma^\star, \neg B^\star \vdash_{\mathsf{I}} B^\star}\ \supset \mathsf{E}}{\dfrac{\Gamma^\star, \neg B^\star \vdash_{\mathsf{I}} \bot}{\Gamma^\star \vdash_{\mathsf{I}} \neg\neg B^\star}\ \supset \mathsf{I}}\ \supset \mathsf{E}$$

which is much smaller than the result of the selective DNT given in the proof of Theorem 8. The corresponding case of the selective CPS transformation given in Definition 15 can also be improved as follows.

$$\mathcal{S}[\![(M^{\mathtt{T}} \mathbin{@^{\mathtt{T}}} N^{\mathtt{T}})^{\mathtt{N}}]\!] \quad = \quad \lambda k.k\,(\mathcal{S}_t[\![M]\!]\,\mathcal{S}_t[\![N]\!])$$

We leave as future work a further investigation into whether or not this finer analysis of annotations still preserves the operational meaning of the source program in the sense of Theorem 5. Moreover, in this paper, we considered only control operators callcc and throw and the corresponding selective CPS transformation. Since there are other control structures (e.g., delimited continuations [19–21] and exceptions [22]) and corresponding selective CPS transformations, it will also be interesting to investigate their logical interpretation by parameterizing our analysis over such control operators. For further details on various control structures and CPS transformations, the reader may refer to [23].

Another interesting direction for future work is to include other connectives such as conjunction and disjunction which respectively correspond to product and sum types in programming languages. In addition, since we consider only propositional logic, one natural extension is to consider first-order predicate logic, which corresponds to dependently typed λ-calculus. However, it is challenging to define a computationally sound double negation proof translation for classical predicate logic, namely a CPS transformation for a dependently typed language. For example, the standard CPS transformation based on double negation is not type preserving for a dependently typed language with Σ types (strong dependent pairs which correspond to universal quantifiers) [24]. Moreover, unrestricted uses of callcc and throw together with Σ types and equality even leads to an inconsistent system [25]. Recently, Bowman et al. [26] developed type-preserving CBN and CBV CPS transformations for a dependently typed language with Σ and Π types based on answer type polymorphism [27]. It would be interesting to investigate the logical meaning of such translations, which we leave as future work.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1F1A1063272 and 2020R1A4A3079947).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We thank Sungwoo Park who proposed the problem of selective double negation translation. We also thank anonymous reviewers for their helpful comments on an earlier version of the paper.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CBV	call-by-value
CBN	call-by-name
CPL	Classical propositional logic
CPS	Continuation passing style
DNE	Double negation elimination
DNT	Double negation Translation
EM	Excluded middle
IPL	Intuitionistic propositional logic

References

1. Curry, H.B.; Feys, R.; Craig, W. *Combinatory Logic*; North-Holland Publishing Company: Amsterdam, The Netherlands, 1958; Volume 1.
2. Howard, W.A. The formulae-as-types notion of constructions. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*; Academic Press: London, UK, 1980; p. 479–490. Unpublished Manuscript of 1969.
3. Duba, B.; Harper, R.; MacQueen, D. Typing First-class Continuations in ML. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*; Association for Computing Machinery: New York, NY, USA,1991; pp. 163–173, doi:10.1145/99583.99608.
4. Griffin, T.G. A Formulae-as-type Notion of Control. In Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Association for Computing Machinery: New York, NY, USA, 1990; pp. 47–58, doi:10.1145/96709.96714.
5. Felleisen, M.; Friedman, D.P.; Kohlbecker, E.E.; Duba, B.F. Reasoning with Continuations. In Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, MA, USA, 16–18 June 1986; pp. 131–141.
6. Gentzen, G. Über das Verhältnis zwischen intuitionistischer und klassischer Arithmetik. *Archiv für Mathematische Logik und Grundlagenforschung* **1974**, *16*, 119–132, doi:10.1007/BF02015371.
7. Glivenko, V. Sur Quelques Points de la Logique de M. Brouwer. *Bull. Cl. Des. Sci.* **1929**, *15*, 183–188.
8. Gödel, K. Zur intuitionistischen arithmetik und zahlentheorie. *Ergeb. Eines Math. Kolloquiums* **1933**, *4*, 34–38.
9. Kolmogorov, A.N. On the principle of the excluded middle. *Mat. Sbornik* **1925**, *32*, 646–667. (In Russian)
10. Kuroda, S. Intuitionistische Untersuchungen der formalistischen Logik. *Nagoya Math. J.* **1951**, *2*, 35–47.
11. Miquey, É. Classical Realizability and Side-Effects. Ph.D. Thesis, Universidad de la República, Montevideo, Uruguay, 2017.
12. Plotkin, G. Call-by-name, call-by-value and the λ-calculus. *Theor. Comput. Sci.* **1975**, *1*, 125–159, doi:10.1016/0304-3975(75)90017-1.
13. Reynolds, J.C. Definitional Interpreters for Higher-order Programming Languages. In *Proceedings of the ACM Annual Conference*; Association for Computing Machinery: New York, NY, USA, 1972; pp. 717–740, doi:10.1145/800194.805852.
14. Appel, A.W. *Compiling with Continuations*; Cambridge University Press: New York, NY, USA, 1992.
15. Reppy, J.H. Optimizing Nested Loops Using Local CPS Conversion. *High. Order Symb. Comput.* **2002**, *15*, 161–180.
16. Nielsen, L.R. A Selective CPS Transformation. *Electron. Notes Theor. Comput. Sci.* **2001**, *45*, 311–331, doi:10.1016/S1571-0661(04)80969-1.
17. Wright, A.; Felleisen, M. A Syntactic Approach to Type Soundness. *Inf. Comput.* **1994**, *115*, 38–94, doi:10.1006/inco.1994.1093.
18. Danvy, O.; Hatcliff, J. On the Transformation Between Direct and Continuation Semantics. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*; Springer: Berlin, Germany, 1993; pp. 627–648.
19. Rompf, T.; Maier, I.; Odersky, M. Implementing First-Class Polymorphic Delimited Continuations by a Type-Directed Selective CPS-Transform. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*; Association for Computing Machinery: New York, NY, USA, 2009; pp. 317–328, doi:10.1145/1596550.1596596.
20. Materzok, M.; Biernacki, D. Subtyping Delimited Continuations. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*; Association for Computing Machinery: New York, NY, USA, 2011; pp. 81–93, doi:10.1145/2034773.2034786.
21. Asai, K.; Uehara, C. Selective CPS Transformation for Shift and Reset. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 40–52, doi:10.1145/3162069.
22. Kim, J.; Yi, K.; Danvy, O. Assessing the Overhead of ML Exceptions by Selective CPS Transformation. In *ACM SIGPLAN Workshop on ML*; ACM: New York, NY, USA, 1998; pp. 103–114.
23. Berdine, J. Linear and Affine Typing of Continuation-Passing Style. Ph.D. Thesis, Queen Mary, University of London, London, UK, 2004.
24. Barthe, G.; Uustalu, T. CPS Translating Inductive and Coinductive Types. In Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, Portland, OR, USA, 14–15 January 2002; pp. 131–142, doi:10.1145/503032.503043.
25. Herbelin, H. On the Degeneracy of Σ-Types in Presence of Computational Classical Logic. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*; Springer: Berlin, Germany, 2005; pp. 209–220.
26. Bowman, W.J.; Cong, Y.; Rioux, N.; Ahmed, A. Type-preserving CPS Translation of Σ and Π Types is Not Not Possible. *Proc. ACM Program. Lang.* **2018**, *2*, 22:1–22:33, doi:10.1145/3158110.
27. Thielecke, H. From Control Effects to Typed Continuation Passing. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*; ACM: New York, NY, USA, 2003; pp. 139–149, doi:10.1145/604131.604144.