Modal type theory based on the intuitionistic epistemic logic

Abstract

Modal intuitionistic epistemic logic IEL^- was proposed by S.Artemov and T. Protopopescu as the formal foundation for the intuitionistic theory of knowledge. We construct a modal simply typed lambda-calculus which is Curry-Howard isomorphic to IEL^- as formal theory of calculations with applicative functors in functional programming languages like Haskell or Idris. We prove that this typed lambda-calculus has the strong normalization and Church-Rosser properties.

1 Introduction

Modal intutionistic epistemic logic IEL was proposed by S. Artemov and T. Proropopescu [1]. IEL provides the epistimology and the theory of knowledge as based on BHK-semantics of intuitionistic logic. IEL^- is a variant of IEL, that corresponds to intuitionistic belief. Informally, $\mathbf{K}A$ denotes that A is verified intuitionistically.

Intuitionistic epistemic logic IEL^- is defined with by following axioms and derivation rules:

Definition 1. *Intuitionistic epistemic logic IEL:*
1) IPC axioms;
2) $\mathbf{K}(A \to B) \to (\mathbf{K}A \to \mathbf{K}B)$ *(normality);*
3) $A \to \mathbf{K}A$ *(co-reflection);*
Rule: MP.

We have the deduction theorem and necessitation rule which is derivable.

V. Krupski and A. Yatmanov provided the sequential calculus for IEL and proved that this calculus is PSPACE-complete [2].

It's not difficult to see that modal axioms in IEL^- and types of the methods of Applicative class in Haskell-like languages (which is described below) are syntactically similar and we are going to show that this coincidence has a non-trivial computational meaning.

Functional programming languages such as Haskell [3], Idris [4], Purescript [5] or Elm [6] have special type classes[1] for calculations with container types like `Functor` and `Applicative` [2]:

[1] Type class in Haskell is a general interface for special group of datatypes.

[2] Reader may read more about container types in the Haskell standard library documentation[7] or in the next one textbook [8]

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b

class Functor f => Applicative f where
  pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

By *container* (or *computational context*) type we mean some type-operator f, where f is a "function" from $*$ to $*$: type operator takes a simple type (which has kind $*$) and returns another simple type type with kind $*$. For more detailed description of the type system with kinds used in Haskell see [12].

The main goal of our research is a relationship between intuitionistic epistemic logic IEL^- and functional programming with effects. We show that relationship by building the type system (which is called $\lambda_{\mathbf{K}}$) which is Curry-Howard isomorphic to IEL^-. So we will consider **K**-modality as an arbitrary applicative functor.

$\lambda_{\mathbf{K}}$ consists of the rules for simply typed lambda-calculus and special typing rules for lifting types into the applicative functor **K**. We assume that our type system will axiomatize the simplest case of computation with effects with one container. We provide proof-theoretical view on this kind of computations in functional programming and prove strong normalization and confluence.

2 Typed lambda-calculus based on IEL$^-$

At first we define the natural deduction for IEL$^-$ with **K**-modality and binary connectives $\to$ and $\wedge$ (we call that calculus NIEL$^-_{\wedge,\to}$):

Definition 2. *Natural deduction $NIEL^-_{\wedge,\to}$ for IEL^- with $\to$ and $\wedge$:*

$$\frac{}{\Gamma, A \vdash A}\, ax$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_i \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to_i$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash A_1 \wedge A_2}{\Gamma \vdash A_i} \wedge_e, i \in \{1, 2\}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \boldsymbol{K}A} \boldsymbol{K_I} \qquad \frac{\Gamma \vdash \mathbf{K}\vec{A} \qquad \vec{A} \vdash B}{\Gamma \vdash \mathbf{K}B}$$

Where $\Gamma \vdash \mathbf{K}\vec{A}$ is a syntax sugar for $\Gamma \vdash \mathbf{K}A_1, \ldots, \Gamma \vdash \mathbf{K}A_n$.

Lemma 1. $\Gamma \vdash_{NIEL^-_{\wedge,\to}} A \Rightarrow IEL^- \vdash \bigwedge \Gamma \to A$.

Proof. Induction on the derivation.

Let us consider cases with modality.

1) If $\Gamma \vdash_{NIEL^-_{\wedge,\to}} A$, then $IEL^- \vdash \bigwedge \Gamma \to \mathbf{K}A$.

(1)	$\bigwedge\Gamma \to A$	assumption
(2)	$A \to \mathbf{K}A$	co-reflection
(3)	$(\bigwedge\Gamma \to A) \to ((A \to \mathbf{K}A) \to (\bigwedge\Gamma \to \mathbf{K}A))$	IPC theorem
(4)	$(A \to \mathbf{K}A) \to (\bigwedge\Gamma \to \mathbf{K}A)$	from (1), (3) and MP
(5)	$\bigwedge\Gamma \to \mathbf{K}A$	from (2), (4) and MP

2) If $\Gamma \vdash_{NIEL^-_{\wedge,\to}} \mathbf{K}\vec{A}$ and $\vec{A} \vdash B$, then $IEL^- \vdash \bigwedge\Gamma \to \mathbf{K}B$.

(1)	$\bigwedge\Gamma \to \bigwedge\limits_{i=1}^{n} \mathbf{K}A_i$	assumption
(2)	$\bigwedge\limits_{i=1}^{n} \mathbf{K}A_i \to \mathbf{K}\bigwedge\limits_{i=1}^{n} A_i$	IEL theorem
(3)	$\bigwedge\Gamma \to \mathbf{K}\bigwedge\limits_{i=1}^{n} A_i$	from (1), (2) and transitivity
(4)	$\bigwedge\limits_{i=1}^{n} A_i \to B$	assumption
(5)	$(\bigwedge\limits_{i=1}^{n} A_i \to B) \to \mathbf{K}(\bigwedge\limits_{i=1}^{n} A_i \to B)$	co-reflection
(6)	$\mathbf{K}(\bigwedge\limits_{i=1}^{n} A_i \to B)$	from (2), (3) and MP
(7)	$\mathbf{K}\bigwedge\limits_{i=1}^{n} A_i \to \mathbf{K}B$	from (6) and normality
(8)	$\bigwedge\Gamma \to \mathbf{K}B$	from (3), (7) and transitivity

□

Lemma 2. *If $IEL^- \vdash A$, then $NIEL^- \vdash A$.*

Proof. Straightforward derivation of modal axioms in NIEL$^-$. We consider this derivation below using terms. □

It is clearly that these lemmas could be extended for IEL$^-$ with $\vee$ and $\neg$ similary.

At the next step we build the typed lambda-calculus based on NIEL$^-_{\wedge,\to}$ by proof-assingment in rules.

At first, we define lambda-terms and types for this lambda-calculus.

Definition 3. *The set of terms:*

Let $\mathbb{V}$ be the set of variables. The set $\Lambda_{\mathbf{K}}$ of terms is defined by the grammar:

$$\Lambda_{\mathbf{K}} ::= \mathbb{V} \mid (\lambda\Lambda.\Lambda_{\mathbf{K}}) \mid (\Lambda_{\mathbf{K}}\Lambda_{\mathbf{K}}) \mid (\Lambda_{\mathbf{K}}, \Lambda_{\mathbf{K}}) \mid (\pi_1\Lambda_{\mathbf{K}}) \mid (\pi_2\Lambda_{\mathbf{K}}) \mid (\mathbf{pure}\ \Lambda_{\mathbf{K}}) \mid (\mathbf{let\ pure}\ \Lambda_{\mathbf{K}} = \Lambda_{\mathbf{K}}\ \mathbf{in}\ \Lambda_{\mathbf{K}})$$

Definition 4. *The set of types:*

Let $\mathbb{T}$ be the set of atomic types. The set $\mathbb{T}_{\mathbf{K}}$ of types with applicative functor $\mathbf{K}$ is generated by the grammar:

$$\mathbb{T}_{\mathbf{K}} ::= \mathbb{T} \mid (\mathbb{T}_{\mathbf{K}} \to \mathbb{T}_{\mathbf{K}}) \mid (\mathbb{T}_{\mathbf{K}} \times \mathbb{T}_{\mathbf{K}}) \mid (\mathbf{K}\mathbb{T}_{\mathbf{K}}) \quad (1)$$

Context, domain of context and range of context are defined standardly [11][12].

Our type system is based on the Curry-style typing rules:

Definition 5. *Modal typed lambda calculus $\lambda_{\mathbf{K}}$ based on NIEL$^-_{\wedge,\to}$:*

$$\dfrac{}{\Gamma, x : A \vdash x : A}\ ax$$

$$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \to_i \qquad \dfrac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash x : A}{\Gamma \vdash fx : B}\ \to_e$$

$$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle x, y \rangle : A \times B}\ \times_i \qquad \dfrac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i M : A_i}\ \times_e,\ i \in \{1, 2\}$$

$$\dfrac{\Gamma \vdash x : A}{\Gamma \vdash \mathbf{pure}\ x : \mathbf{K}A}\ \boldsymbol{K_I} \qquad \dfrac{\Gamma \vdash \vec{M} : \mathbf{K}\vec{A} \qquad \vec{x} : \vec{A} \vdash M : B}{\Gamma \vdash \mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ M : \mathbf{K}B}\ let_{\mathbf{K}}$$

$\mathbf{K}_I$-typing rule is the same as $\bigcirc$-introduction in lax logic (also known as monadic metalanguage [17]) and in typed lambda-calculus which is derived by proof-assignment for lax-logic proofs. $\mathbf{K}_I$ allows to inject an object of type α into the functor. $\mathbf{K}_I$ reflects the Haskell method **pure** for Applicative class. It plays the same role as the **return** method in Monad class.

let$_{\mathbf{K}}$ is the same as the $\Box$-rule in typed lambda calculus for intuitionistic normal modal logic **IK**, which is described in [19].

In fact, our calculus is the extention of typed lambda calculus for **IK** with typing rule appropriate to co-reflection.

Here are some examples of derivation trees.

$$\dfrac{\dfrac{x : A \vdash x : A}{x : A \vdash \mathbf{pure}\, x : \mathbf{K}A}\ \mathbf{K}_I}{\vdash (\lambda x.\mathbf{pure}\, x) : A \to \mathbf{K}A}\ \to_i$$

$$\dfrac{\dfrac{\dfrac{f : \mathbf{K}(A \to B) \vdash f : \mathbf{K}(A \to B) \qquad x : \mathbf{K}A \vdash x : \mathbf{K}A \qquad \dfrac{g : A \to B \qquad y : A}{g : A \to B, y : A \vdash gy : B}}{f : \mathbf{K}(A \to B), x : \mathbf{K}A \vdash \mathbf{let\ pure}\ \langle g, y \rangle = \langle f, x \rangle\ \mathbf{in}\ gy : \mathbf{K}B}}{f : \mathbf{K}(A \to B) \vdash \lambda x.\mathbf{let\ pure}\ \langle g, y \rangle = \langle f, x \rangle\ \mathbf{in}\ gy : \mathbf{K}A \to \mathbf{K}B}}{\vdash \lambda f.\lambda x.\mathbf{let\ pure}\ \langle g, y \rangle = \langle f, x \rangle\ \mathbf{in}\ gy : \mathbf{K}(A \to B) \to \mathbf{K}A \to \mathbf{K}B}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{f : A \to B \vdash f : A \to B}{f : A \to B \vdash \mathbf{pure}\ f : \mathbf{K}(A \to B)} \qquad x : \mathbf{K}A \vdash x : \mathbf{K}A \qquad \dfrac{g : A \to B \qquad y : A}{g : A \to B, y : A \vdash gy : B}}{f : A \to B, x : \mathbf{K}A \vdash \mathbf{let\ pure}\ \langle g, y \rangle = \langle \mathbf{pure}\ f, x \rangle\ \mathbf{in}\ gy : \mathbf{K}B}}{f : A \to B \vdash \lambda x.\mathbf{let\ pure}\ \langle g, y \rangle = \langle \mathbf{pure}\ f, x \rangle\ \mathbf{in}\ gy : \mathbf{K}A \to \mathbf{K}B}}{\lambda f.\lambda x.\mathbf{let\ pure}\ \langle g, y \rangle = \langle \mathbf{pure}\ f, x \rangle\ \mathbf{in}\ gy : (A \to B) \to \mathbf{K}A \to \mathbf{K}B}$$

Now we define free variables and substitutions. β-reduction, multi-step β-reduction and β-equality are defined standardly:

Definition 6. *Set $FV(M)$ of free variables for arbitrary term M:*

1) $FV(x) = \{x\}$;

2) $FV(\lambda x.M) = FV(M) \setminus \{x\}$;

3) $FV(MN) = FV(M) \cup FV(N)$;
4) $FV(\langle M, N\rangle) = FV(M) \cup FV(N)$;
5) $FV(\pi_i M) \subseteq FV(M)$, $i \in \{1, 2\}$;
6) $FV(\textit{pure } M) = FV(M)$;
7) $FV(\textbf{let pure } \vec{N} = \vec{M} \textbf{ in } M) = \bigcup\limits_{i=1}^{n} FV(M)$, *where* $n = |\vec{M}|$.

Definition 7. *Substitution:*
1) $x[x := N] = N$, $x[y := N] = x$;
2) $(MN)[x := N] = M[x := N]N[x := N]$;
3) $(\lambda x.M)[x := N] = \lambda x.M[x := N]$;
4) $(M, N)[x := P] = (M[x := P], N[x := P])$;
5) $(\pi_i M)[x := P] = \pi_i(M[x := P])$, $i \in \{1, 2\}$;
6) $(\textbf{pure } M)[x := P] = \textbf{pure } (M[x := P])$;
7) $(\textbf{let pure } \vec{x} = \vec{M} \textbf{ in } M)[y := P] = \textbf{let pure } \vec{x} = (\vec{M}[y := P]) \textbf{ in } M$.

Definition 8. *β-reduction and η-reduction rules for $\lambda\mathbf{K}$.*
1) $(\lambda x.M)N \to_\beta M[x := N]$;
2) $\pi_1\langle M, N\rangle \to_\beta M$;
3) $\pi_2\langle M, N\rangle \to_\beta N$;
4) $\textbf{let pure } \langle \vec{x}, y, \vec{z}\rangle = \langle \vec{M}, \textbf{let pure } \vec{w} = \vec{N} \textbf{ in } Q, \vec{P}\rangle \textit{ in } R \to_\beta$
$\textbf{let pure } \langle \vec{x}, \vec{w}, \vec{z}\rangle = \langle \vec{M}, \vec{N}, \vec{P}\rangle \textbf{ in } R[y := Q]$
5) $\textbf{let pure } \vec{x} = \textbf{pure } \vec{M} \textbf{ in } N \to_\beta \textbf{pure } N[\vec{x} := \vec{M}]$
6) $\lambda x.fx \to_\eta f$;
7) $\langle \pi_1 P, \pi_2 P\rangle \to_\eta P$;
8) $\textbf{let pure } \_\_ = \_\_ \textbf{ in } N \to_\eta \textbf{pure } N$;
9) $\textbf{let pure } x = M \textbf{ in } x \to_\eta M$;
10) $M \to_{\beta\eta} N \Rightarrow \textbf{pure } \mathbf{M} \to_{\beta\eta} \textbf{pure } \mathbf{N}$

3 Basic lemmas

Now we will prove standard lemmas for contexts in type systems[3]:

Lemma 3. *Generation lemma.*
i) Let $\Gamma \vdash \textbf{pure } M : \mathbf{K}A$, *then* $\Gamma \vdash M : A$;
ii) Let $\Gamma \vdash \textbf{let pure } \vec{x} = \vec{M} \textbf{ in } N : \mathbf{K}B$, *there are some* $A_1, \ldots, A_n \in \mathbb{T}_\mathbf{K}$, *such that* $\Gamma \vdash \vec{M} : \mathbf{K}\vec{A}$ *and* $\vec{x} : \vec{A} \vdash N : B$.

Proof.
Induction on $\Gamma \vdash \textbf{pure } M : \mathbf{K}A$ and $\Gamma \vdash \textbf{let pure } \vec{x} = \vec{N} \textbf{ in } N : \mathbf{K}B$ correspondently. □

Lemma 4. *Weakening.*
Let $\Gamma \vdash M : A$ *and* $\Gamma \subseteq \Delta$, *then* $\Delta \vdash M : A$.

[3]We will not prove cases with $\to$-constructor, they are proved standardly in the same lemmas for simply typed lambda calculus, for example see [11][12][14]. We will consider only modal cases

Proof.

1) Let $\Gamma, x : A \vdash x : A$ and $\Gamma \subseteq \Delta$, then $\Delta, x : A \vdash x : A$ trivially.

2) Let $\Gamma \vdash \textbf{pure}\, M : \mathbf{K}A$. Then $\Gamma \vdash M : A$ by generation and $\Delta \vdash M : A$ by assumption. So $\Delta \vdash \textbf{pure}\, M : \mathbf{K}A$ by $\mathbf{K}_I$.

3) Let $\Gamma \vdash \textbf{let pure}\, \vec{x} = \vec{M}\, \textbf{in}\, N : \mathbf{K}B$ and $\Gamma \subseteq \Delta$. Then $\Gamma \vdash \vec{M} : \mathbf{K}\vec{A}$ and $\vec{x} : \vec{A} \vdash N : B$.

By assumption $\Delta \vdash \vec{M} : \mathbf{K}\vec{A}$. So $\Delta \vdash \textbf{let pure}\, \vec{x} = \vec{N}\, \textbf{in}\, N : \mathbf{K}B$ by $\text{let}_{\mathbf{K}}$. □

Definition 9. *Type substituition*

The substituition of type C for type variable B in type A inductively defined as follows:

1) $B[B := C] = B$ and $D[B := C] = D$, if $B \neq D$;

2) $(A_1 \alpha A_2)[B := C] = (A_1[B := C])\alpha(A_2[B := C])$, where $\alpha \in \{\rightarrow, \times\}$;

3) $(\mathbf{K}A)[B := C] = \mathbf{K}(A[B := C])$.

4) Let Γ be the context, then $\Gamma[B := C] = \{x : (A[B := C]) \mid x : A \in \Gamma\}$

Lemma 5. *Substituition lemma.*

i) Let $\Gamma, x : A \vdash M : B$ and $\Gamma \vdash N : A$, then $\Gamma \vdash M[x := N] : B$.

ii) Let $\Gamma \vdash M : A$, then $\Gamma[B := C] \vdash M : (A[B := C])$.

Proof.

i) For term substitution:

1) Let $\Gamma, x : A \vdash x : A$ and $\Gamma \vdash N : A$, but $x[x := N] = N$, so $\Gamma \vdash N : A$.

2) Let $\Gamma, x : A \vdash \textbf{pure}\, M : \mathbf{K}B$ and $\Gamma \vdash N : A$.

By generation $\Gamma, x : A \vdash M : B$ and by assumption $\Gamma \vdash M[x := N] : B$.

By K_I, $\Gamma \vdash \textbf{pure}\, (M[x := N]) : \mathbf{K}B$.

3) Let $\Gamma, y : A \vdash \textbf{let pure}\, \vec{x} = \vec{M}\, \textbf{in}\, N : \mathbf{K}B$ and $\Gamma \vdash N : A$.

By generation, $\Gamma, y : A \vdash \vec{M} : \mathbf{K}\vec{A}$ and $\vec{x} : \vec{A} \vdash N : B$.

By hypothesis, $\Gamma \vdash \vec{M}[x := N] : \mathbf{K}\vec{A}$.

Hence $\Gamma \vdash \textbf{let pure}\, \vec{x} = \vec{M}[x := N]\, \textbf{in}\, N : \mathbf{K}B$.

ii) For type substitution

1) Let $\Gamma, x : A \vdash x : A$, so $\Gamma[A := C], x : (A[A := C]) \vdash x : (A[A := C])$, or $\Gamma[A := C], x : C \vdash x : C$.

2) Let $\Gamma \vdash \textbf{pure}\, M : \mathbf{K}A$. By generation $\Gamma \vdash M : A$ and by assumption $\Gamma[B := C] \vdash M : A[B := C]$.

By K_I $\Gamma \vdash \textbf{pure}\, \mathbf{L}M : \mathbf{K}(A[B := C])$.

3) $\Gamma \vdash \textbf{let pure}\, \vec{x} = \vec{M}\, \textbf{in}\, N : \mathbf{K}B$. By generation $\Gamma \vdash \vec{M} : \mathbf{K}\vec{A}$ and $\vec{x} : \vec{A} \vdash N : B$.

By assumption $\Gamma[B_1 := C] \vdash \vec{M} : K\vec{A}[B_1 := C]$ and $\vec{x} : \vec{A}[B_1 := C] \vdash N : B[B_1 := C]$.

So by $\text{let}_{\mathbf{K}}$, $\Gamma[B_1 := C] \vdash \textbf{let pure}\, \vec{x} = \vec{M}\, \textbf{in}\, N : \mathbf{K}(B[B_1 := C])$.

□

Theorem 1. *Subject reduction*

Let $\Gamma \vdash M : A$ and $M \twoheadrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : A$

For cases with application, abstraction and pairs see [12] [13].

1) Let $\Gamma \vdash$ **let pure** $\langle \vec{x}, y, \vec{z} \rangle = \langle \vec{M}, \textbf{let pure}\ \vec{w} = \vec{N}\ \textbf{in}\ Q, \vec{P} \rangle\ in\ R : \mathbf{K}B$, then Γ**let pure** $\langle \vec{x}, \vec{w}, \vec{z} \rangle = \langle \vec{M}, \vec{N}, \vec{P} \rangle$ **in** $R[y := Q] : \mathbf{K}B$

2) Let $\Gamma \vdash$ **let pure** $x = M$ **in** $x : \mathbf{K}A$, then $\Gamma \vdash M : \mathbf{K}A$.

See [19].

3) Let $\Gamma \vdash$ **let pure** $\vec{x} =$ **pure** $\vec{M}$ **in** $N : \mathbf{K}B$.

By generation $\Gamma \vdash$ **pure** $\vec{M} : \mathbf{K}\vec{A}$ and $\vec{x} : \vec{A} \vdash N : B$.
Moreover, $\Gamma \vdash \vec{M} : \vec{A}$. By weakening and substitution lemma $\Gamma \vdash N[\vec{x} = \vec{M}] : B$.
By $\mathbf{K}_I$, $\Gamma \vdash$ **pure** $N[\vec{x} := \vec{M}] : \mathbf{K}B$.

4) Let $\vdash$ **let pure** $\_\_ = \_\_$ **in** $N : \mathbf{K}A$

By generation $\vdash N : A$.
So $\vdash$ **pure** $N : \mathbf{K}A$ by $\mathbf{K}_I$.

5) Let $\Gamma \vdash$ **pure** $M : A$ and $M \twoheadrightarrow_{\beta\eta} N$.

By generation $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ by assumption.
So $\Gamma \vdash$ **pure** $N : \mathbf{K}A$.

4 Strong normalization

5 Confluence

6 Categorical semantics

Definition 10. *Lax monoidal functor*

Let $\langle \mathcal{C}, \otimes_1, \mathbb{1} \rangle$ and $\langle \mathcal{D}, \otimes_2, \mathbb{1}' \rangle$ are monoidal categories.

A lax monoidal functor $\mathcal{F} : \langle \mathcal{C}, \otimes_1, \mathbb{1} \rangle \to \langle \mathcal{D}, \otimes_2, \mathbb{1}' \rangle$ is a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ with additional natural transformations:

1) $u : \mathbb{1}' \to \mathcal{F}\mathbb{1}$;

2) $*_{A,B} : \mathcal{F}A \otimes_2 \mathcal{F}B \to \mathcal{F}(A \otimes_1 B)$

Definition 11. *Applicative functor*

An applicative functor is a triple $\langle \mathcal{C}, \mathcal{K}, \eta \rangle$, where $\mathcal{C}$ is a symmetric monoidal category, $\mathcal{K}$ is a lax monoidal endofunctor and η is a natural transformation, such that:

1) $u = \eta_{\mathbb{1}}$;

2) $*_{A,B} \circ (\eta_A \otimes \eta_B) = \eta_{A \otimes B}$;

3) Weak commutativity condition holds:

$$\begin{array}{ccc} A \otimes \mathcal{K}B & \mathcal{K}A \otimes \mathcal{K}B & \mathcal{K}(A \otimes B) \\ & & \\ \mathcal{K}B \otimes A & \mathcal{K}B \otimes \mathcal{K}A & \mathcal{K}(B \otimes A) \end{array}$$

By default we will consider an arbitrary closed functor on some cartersian closed category, which is the special case of an applicative functor.

We identify terminal objects. So $\mathcal{K}(\mathbb{1}) = \mathbb{1}$ and $\eta_{\mathbb{1}} = id_{\mathbb{1}}$ since $\mathcal{K}$ is an endofunctor.

6.1 Soundness

Theorem 2. *Soundness*

Let $\Gamma \vdash M : A$ *and* $M =_{\beta\eta} N$, *then* $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$

Proof.

Definition 12. *Semantical translation from* $\lambda_{\boldsymbol{K}}$ *to CCC with applicative functor* $\mathcal{K}$:

1) Interpretation for types:
$[\![A]\!] := \hat{A}, A \in \mathbb{T}$;
$[\![A \to B]\!] := [\![A]\!] \to [\![B]\!]$;
$[\![A \times B]\!] := [\![A]\!] \times [\![B]\!]$.
2) Interpretation for modal types: $[\![\boldsymbol{K}A]\!] = \mathcal{K}[\![A]\!]$;
3) Interpretaion for contexts:
$[\![\Gamma = \{x_1 : A_1, ..., x_n : A_n\}]\!] := [\![\Gamma]\!] = [\![A_1]\!] \times ... \times [\![A_n]\!]$;
4) Interpretation for typing assignment: $[\![\Gamma \vdash M : A]\!] := [\![M]\!] : [\![\Gamma]\!] \to [\![A]\!]$.
5) Interpretation for typing rules:

$$\frac{}{[\![\Gamma, x : A \vdash x : A]\!] = \pi_2 : [\![\Gamma]\!] \times [\![A]\!] \to [\![A]\!]}$$

$$\frac{[\![\Gamma, x : A \vdash M : B]\!] = f : [\![\Gamma]\!] \times [\![A]\!] \to [\![B]\!]}{[\![\Gamma \vdash (\lambda x.M) : A \to B]\!] = \Lambda(f) : [\![\Gamma]\!] \to [\![B]\!]^{[\![A]\!]}}$$

$$\frac{[\![\Gamma \vdash M : A \to B]\!] = [\![M]\!] : [\![\Gamma]\!] \to [\![B]\!]^{[\![A]\!]} \qquad [\![\Gamma \vdash N : A]\!] = [\![N]\!] : [\![\Gamma]\!] \to [\![A]\!]}{[\![\Gamma \vdash (MN) : B]\!] = [\![\Gamma]\!] \xrightarrow{\langle [\![M]\!], [\![N]\!] \rangle} [\![B]\!]^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\epsilon} [\![B]\!]}$$

$$\frac{[\![\Gamma \vdash M : A]\!] = f : [\![\Gamma]\!] \to [\![A]\!] \qquad [\![\Gamma \vdash N : B]\!] = g : [\![\Gamma]\!] \to [\![B]\!]}{[\![\Gamma \vdash (M, N) : A \times B]\!] = \langle f, g \rangle : [\![\Gamma]\!] \to [\![A]\!] \times [\![B]\!]}$$

$$\frac{[\![\Gamma \vdash p : A_1 \times A_2]\!] = f : [\![\Gamma]\!] \to [\![A_1]\!] \times [\![A_2]\!]}{[\![\Gamma \vdash \pi_i p : A_i]\!] = [\![\Gamma]\!] \xrightarrow{f} [\![A_1]\!] \times [\![A_2]\!] \xrightarrow{\pi_i} [\![A_i]\!]} \; i \in \{1, 2\}$$

$$\frac{[\![\Gamma \vdash M : A]\!] = [\![M]\!] : [\![\Gamma]\!] \to [\![A]\!]}{[\![\Gamma \vdash \textbf{pure}\, M : \boldsymbol{K}A]\!] := [\![\Gamma]\!] \xrightarrow{[\![M]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} \mathcal{K}[\![A]\!]}$$

$$\frac{[\![\Gamma \vdash \vec{M} : \mathbf{K}\vec{A}]\!] = \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle : [\![\Gamma]\!] \to \prod_{i=1}^{n} \mathcal{K}[\![A_i]\!] \qquad [\![\vec{x} : \vec{A} \vdash N : B]\!] = [\![N]\!] : \prod_{i=1}^{n} [\![A_i]\!] \to [\![B]\!]}{[\![\Gamma \vdash \textbf{let pure}\, \vec{x} = \vec{M}\, \textbf{in}\, M : \mathbf{K}B]\!] = \mathcal{K}([\![N]\!]) \circ *_{[\![A_1]\!], \ldots, [\![A_n]\!]} \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle : [\![\Gamma]\!] \to \mathcal{K}[\![B]\!]}$$

Definition 13. *Simultaneous substitution*

Let $\Gamma = \{x_1 : A_1, ..., x_n : A_n\}$, $\Gamma \vdash M : A$ *and for all* $i \in \{1, ..., n\}$, $\Gamma \vdash M_i : A_i$.

We define simultaneous substitution $M[\vec{x} := \vec{M}]$ *recursively by:*
1) $x_i[\vec{x} := \vec{M}] = M_i$;
2) $(\lambda x.M)[\vec{x} := \vec{M}] = \lambda x.(M[\vec{x} := \vec{M}])$;
3) $(MN)[\vec{x} := \vec{M}] = (M[\vec{x} = \vec{M}])(N[\vec{x} := \vec{M}])$;
4) $\langle M, N \rangle = \langle (M[\vec{x} = \vec{M}]), (N[\vec{x} := \vec{M}]) \rangle$;

5) $(\pi_i P)[\vec{x} := \vec{M}] = \pi_i(P[\vec{x} = \vec{M}])$;
6) $(\mathbf{pure}\, M)[\vec{x} := \vec{M}] = \mathbf{pure}\,(M[\vec{x} = \vec{M}])$;
7) $(\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N)[\vec{y} := \vec{P}] = \mathbf{let\ pure}\, \vec{x} = (\vec{M}[\vec{y} := \vec{P}])\ \mathbf{in}\ N$

Lemma 6.
$[\![M[x_1 := M_1, \ldots, x_n := M_n]]\!] = [\![M]\!] \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle$.

Proof.
1) $[\![\Gamma \vdash (\mathbf{pure}\, M)[\vec{x} := \vec{M}] : \mathbf{K}A]\!] = [\![\Gamma \vdash \mathbf{pure}\, M : \mathbf{K}A]\!] \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle$.

$$
\begin{array}{lll}
[\![\Gamma \vdash (\mathbf{pure}\, M)[\vec{x} := \vec{M}] : \mathbf{K}A]\!] & = [\![\Gamma \vdash \mathbf{pure}\,(M[\vec{x} := \vec{M}]) : \mathbf{K}A]\!] & \text{Substitution definition} \\
& = \eta_{[\![A]\!]} \circ [\![(M[\vec{x} := \vec{M}])]\!] & \text{Translation for pure} \\
& = \eta_{[\![A]\!]} \circ ([\![M]\!] \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle) & \text{Induction hypothesis} \\
& = (\eta_{[\![A]\!]} \circ [\![M]\!]) \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle & \text{Associativity of composition} \\
& = [\![\Gamma \vdash \mathbf{pure}\, M : \mathbf{K}A]\!] \circ \langle [\![M_1]\!], \ldots, [\![M_n]\!] \rangle & \text{Translation for pure}
\end{array}
$$

2) $[\![\Gamma \vdash (\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N)[\vec{y} := \vec{P}] : \mathbf{K}B]\!] = [\![\Gamma \vdash \mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N : \mathbf{K}B]\!] \circ \langle [\![P_1]\!], \ldots, [\![P_n]\!] \rangle$

$[\![\Gamma \vdash (\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N)[\vec{y} := \vec{P}] : \mathbf{K}B]\!] =$
Substitution definition
$[\![\Gamma \vdash \mathbf{let\ pure}\, \vec{x} = (\vec{M}[\vec{y} := \vec{P}])\ \mathbf{in}\ N : \mathbf{K}B]\!] =$
Interpretaion for $let_{\mathbf{K}}$
$\mathcal{K}([\![N]\!]) \circ *_{[\![A_1]\!], \ldots, [\![A_n]\!]} \circ [\![\Gamma \vdash (\vec{M}[\vec{y} := \vec{P}]) \vdash : \mathbf{K}\vec{A}]\!] =$
Induction hypothesis
$\mathcal{K}([\![N]\!]) \circ *_{[\![A_1]\!], \ldots, [\![A_n]\!]} \circ ([\![\vec{M}]\!] \circ \langle [\![P_1]\!], \ldots, [\![P_n]\!] \rangle) =$
Associativity of composition
$(\mathcal{K}([\![N]\!]) \circ *_{[\![A_1]\!], \ldots, [\![A_n]\!]} \circ [\![\vec{M}]\!]) \circ \langle [\![P_1]\!], \ldots, [\![P_n]\!] \rangle =$
By interpretation
$[\![\Gamma \vdash (\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N]\!] \circ \langle [\![P_1]\!], \ldots, [\![P_n]\!] \rangle$

□

Lemma 7.
i) Let $\Gamma \vdash M : A$ *and* $M \twoheadrightarrow_\beta N$*, then* $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$;
ii) Let $\Gamma \vdash M : A$ *and* $M \twoheadrightarrow_\eta N$*, then* $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$;

Proof.
i) For β-reduction

Cases with β-reductions for $let_{\mathbf{K}}$ are shown in [20]. Let us consider cases with **pure**.

1) $[\![\Gamma \vdash \mathbf{let\ pure}\, \vec{x} = \mathbf{pure}\, \vec{M}\ \mathbf{in}\ N : \mathbf{K}B]\!] = [\![\Gamma \vdash \mathbf{pure}\, N[\vec{x} := \vec{M}] : \mathbf{K}B]\!]$

$\llbracket \Gamma \vdash \textbf{let pure } \vec{x} = \textbf{pure } \vec{M} \textbf{ in } N : \mathbf{K}B \rrbracket =$
By interpretation
$\mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket,\dots,\llbracket A_n \rrbracket} \circ \langle \eta_{\llbracket A_1 \rrbracket} \circ \llbracket M_1 \rrbracket, \dots, \eta_{\llbracket A_n \rrbracket} \circ \llbracket M_n \rrbracket \rangle =$
By the property of a pair of morphisms
$\mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket,\dots,\llbracket A_n \rrbracket} \circ (\eta_{\llbracket A_1 \rrbracket} \times \dots \times \eta_{\llbracket A_n \rrbracket}) \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle =$
Associativity of composition
$\mathcal{K}(\llbracket N \rrbracket) \circ (*_{\llbracket A_1 \rrbracket,\dots,\llbracket A_n \rrbracket} \circ (\eta_{\llbracket A_1 \rrbracket} \times \dots \eta_{\llbracket A_n \rrbracket})) \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle =$
By the definition of an applicative functor
$\mathcal{K}(\llbracket N \rrbracket) \circ \eta_{\llbracket A_1 \rrbracket \times \dots \times \llbracket A_n \rrbracket} \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle =$
Naturality of η
$\eta_{\llbracket B \rrbracket} \circ \llbracket N \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle =$
Associativity of composition
$\eta_{\llbracket B \rrbracket} \circ (\llbracket N \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle) =$
Simultaneous substitution lemma
$\eta_{\llbracket B \rrbracket} \circ \llbracket N[\vec{x} := \vec{M}] \rrbracket$
By interpetation
$\llbracket \Gamma \vdash \textbf{pure } (N[\vec{x} := \vec{M}]) : \mathbf{K}B \rrbracket$

2)
If $\Gamma \vdash M : A$ and $M \to_\beta N$, then $\llbracket \Gamma \vdash \textbf{pure } M : \mathbf{K}A \rrbracket = \llbracket \Gamma \vdash \textbf{pure } N : \mathbf{K}A \rrbracket$.

If $\Gamma \vdash M : A$ and $M \to_\beta N$, then $\Gamma \vdash N : A$ by subject reduction.
By assumption $\llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \vdash N : A \rrbracket$.
So $\eta_{\llbracket A \rrbracket} \circ \llbracket \Gamma \vdash M : A \rrbracket = \eta_{\llbracket A \rrbracket} \circ \llbracket \Gamma \vdash N : A \rrbracket$.
Hence $\llbracket \Gamma \vdash \textbf{pure } M : \mathbf{K}A \rrbracket = \llbracket \Gamma \vdash \textbf{pure } N : \mathbf{K}A \rrbracket$.

ii) For η-reduction.
1) $\llbracket \vdash \textbf{let pure } \_\_ = \_\_ \textbf{ in } N : KA \rrbracket = \llbracket \vdash \textbf{pure } N : \mathbf{KA} \rrbracket$.

$\llbracket \vdash \textbf{let pure } \_\_ = \_\_ \textbf{ in } N : KA \rrbracket =$ By interpetation
$\mathcal{K}(\llbracket N \rrbracket) \circ \eta_{\mathbb{1}} =$ Naturality for η
$\eta_{\llbracket A \rrbracket} \circ \llbracket N \rrbracket =$ By interpretation
$\llbracket \vdash \textbf{pure } N : \mathbf{K}A \rrbracket$

2)
If $\Gamma \vdash M : A$ and $M \to_\eta N$, then $\llbracket \Gamma \vdash \textbf{pure } M : \mathbf{K}A \rrbracket = \llbracket \Gamma \vdash \textbf{pure } N : \mathbf{K}A \rrbracket$.

Similar to case with β-reduction.

□

□

6.2 Completeness

We will consider term model for simply typed lambda calculus $\times$ and $\to$ standardly described in [22] [23].

Definition 14. *Let us define an endofunctor $\mathcal{K} : \mathcal{C}(\lambda) \to \mathcal{C}(\lambda)$, such that:*

- $\mathbf{K} : A \mapsto \mathbf{K}A$;

- $\mathbf{K} : [x, M] \in Hom_{\mathcal{C}(\lambda)}(A, B) \mapsto fmap\ f = [y, \mathbf{let\,pure}\, x = y \in M] \in Hom_{\mathcal{C}(\lambda)}(\mathbf{K}A, \mathbf{K}B)$.

Lemma 8. *Functoriality*
i) $\mathbf{K}(g \circ f) = \mathbf{K}(g) \circ \mathbf{K}(f)$*;*
ii) $\mathbf{K}(id_A) = id_{\mathbf{K}A}$.

Proof. Easy checking.

□

Definition 15. *Let us define natural transformations:*

- $\eta : Id_{\mathcal{C}(\lambda)} \to \mathcal{K}$*, such that forall* $A \in Ob_{\mathcal{C}(\lambda)}$, $\eta_A = [x, \mathbf{pure}\, x] \in Hom_{\mathcal{C}(\lambda)}(A, \mathbf{K}A)$*;*
- $*_{A,B} : \mathbf{K}A \times \mathbf{K}B \to \mathbf{K}(A \times B)$

Lemma 9. *Naturality for* η *and for* $*$
i) $fmap\ f \circ \eta_A = \eta_B \circ f$*;*
ii) $fmap\ (f \times g) \circ *_{A,B} = *_{C,D} \circ (fmap\ f) \times (fmap\ g)$.
iii) $*_{A,B} \circ (\eta_A \times \eta_B) = \eta_{A \times B}$*;*

Proof.
i) fmap $f \circ \eta_A = \eta_B \circ f$

$\eta_B \circ f =$
$[y, \mathbf{pure}\, y] \circ [x, M] =$
$[x, \mathbf{pure}\, y[y := M]] =$
$[x, \mathbf{pure}\, M]$
On the other hand
fmap $f \circ \eta_A =$
$[z, \mathbf{let\,pure}\, x = z\ \mathbf{in}\ M] \circ [x, \mathbf{pure\,x}] =$
$[x, \mathbf{let\,pure}\, x = z\ \mathbf{in}\ M[z := \mathbf{pure}\, x]]$
$[x, \mathbf{let\,pure}\, x = \mathbf{pure\,x}\ \mathbf{in}\ M]$
$[x, \mathbf{pure}\, M[x := x]]$
$[x, \mathbf{pure}\, M]$

ii) fmap $(f \times g) \circ *_{A,B} = *_{C,D} \circ$ (fmap f) $\times$ (fmap g)

□

Lemma 10. *Tensorial strength*

Lemma 11. $\mathbf{K}$ *is an applicative functor.*

Theorem 3. *Completeness*
Let $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$*, then* $M =_{\beta\eta} N$.

References

[1] Artemov S. and Protopopescu T., "Intuitionistic Epistemic Logic", *The Review of Symbolic Logic*, 2016, vol. 9, no 2. pp. 266-298.

[2] Krupski V. N. and Yatmanov A., "Sequent Calculus for Intuitionistic Epistemic Logic IEL", *Logical Foundations of Computer Science: International Symposium, LFCS 2016, Deerfield Beach, FL, USA, January 4-7, 2016. Proceedings*, 2016, pp. 187-201.

[3] Haskell Language. // URL: https://www.haskell.org. (Date: 1.08.2017)

[4] Idris. A Language with Dependent Types.// URL:https://www.idris-lang.org. (Date: 1.08.2017)

[5] Purescript. A strongly-typed functional programming language that compiles to JavaScript. URL: http://www.purescript.org. (Date: 1.08.2017)

[6] Elm. A delightful language for reliable webapps. // URL: http://elm-lang.org. (Date: 1.08.2017)

[7] Hackage, "The base package" // URL: https://hackage.haskell.org/package/base-4.10.0.0 (Date: 1.08.2017)

[8] Lipovaca M, "Learn you a Haskell for Great Good!". //URL: http://learnyouahaskell.com/chapters (Date: 1.08.2017)

[9] McBride C. and Paterson R., "Applicative programming with effects", *Journal of Functional Programming*, 2008, vol. 18, no 01. pp 1-13.

[10] McBride C. and Paterson R, "Functional Pearl. Idioms: applicative programming with effects", *Journal of Functional Programming*, 2005. vol. 18, no 01. pp 1-20.

[11] R. Nederpelt and H. Geuvers, "Type Theory and Formal Proof: An Introduction". *Cambridge University Press*, New York, NY, USA, 2014. pp. 436.

[12] Sorensen M. H. and Urzyczyn P, "Lectures on the Curry-Howard isomorphism", *Studies in Logic and the Foundations of Mathematics*, vol. 149, *Elsevier Science*, 1998. pp 261.

[13] Pierce B. C., "Types and Programming Languages". *Cambridge, Mass: The MIT Press*, 2002. pp. 605.

[14] Girard J.-Y., Taylor P. and Lafont Y, "Proofs and Types", *Cambridge University Press*, New York, NY, USA, 1989. pp. 175.

[15] Barendregt. H. P., "Lambda calculi with types" // Abramsky S., Gabbay Dov M., and S. E. Maibaum, "Handbook of logic in computer science (vol. 2), Osborne Handbooks Of Logic In Computer Science", Vol. 2. *Oxford University Press, Inc.*, New York, NY, USA, 1993. pp 117-309.

[16] Hindley J. Roger, "Basic Simple Type Theory". *Cambridge University Press*, New York, NY, USA, 1997. pp. 185.

[17] Pfenning F. and Davies R., "A judgmental reconstruction of modal logic", *Mathematical Structures in Computer Science*, vol. 11, no 4, 2001, pp. 511-540.

[18] H.P. Barendregt. The Lambda Calculus — Its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics, vol. 103. Amsterdam: North-Holland, 1985.

[19] Yoshihiko KAKUTANI, A Curry-Howard Correspondence for Intuitionistic Normal Modal Logic, Computer Software, Released February 29, 2008, Online ISSN , Print ISSN 0289-6540.

[20] Kakutani Y. (2007) Call-by-Name and Call-by-Value in Normal Modal Logic. In: Shao Z. (eds) Programming Languages and Systems. APLAS 2007. Lecture Notes in Computer Science, vol 4807. Springer, Berlin, Heidelberg

[21] T. Abe. Completeness of modal proofs in
first-order predicate logic. Computer Software, JSSST Journal, 24:165 – 177, 2007.

[22]

[23]

[24]