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.

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].

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

```
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
```

[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]

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 motivation for using an applicative functor is quite natural. Applicative functor allows to generalize the action of a functor for functions with arbitrary number of arguments, for instance:

```
liftA2 :: Applicative f => (a -> b -> c) -> f a -> f b -> f c
liftA2 f x y = pure f <*> x <*> y
```

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.

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 $\mathbf{K}$-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 $\mathbf{K}$. 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$^-$:

Definition 2. *Natural deduction NIEL for IEL$^-$ is an extenstion of intuitionistic natural deduction with additional derivation rules for modality:*

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

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

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

$\square$

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. $\square$

At the next step we build the typed lambda-calculus based on NIEL$^{-}_{\wedge,\rightarrow}$ 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:

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

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$^{-}_{\wedge,\rightarrow}$:*

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

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \to_i \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \to_e$$

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

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{pure}\ M : \mathbf{K}A} \boldsymbol{K_I} \qquad \frac{\Gamma \vdash \vec{M} : \mathbf{K}\vec{A} \qquad \vec{x} : \vec{A} \vdash N : B}{\Gamma \vdash \mathbf{let\ pure}\ \vec{x} = \vec{M}\ \mathbf{in}\ N : \mathbf{K}B} \mathit{let}_{\mathbf{K}}$$

$\mathbf{K}_I$-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 similar to 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 closed terms:

- $(\lambda x.\mathbf{pure}\, x) : A \to \mathbf{K}A$;
- $\lambda f.\lambda x.\mathbf{let\ pure}\, g, y = f, x\ \mathbf{in}\ gy : \mathbf{K}(A \to B) \to \mathbf{K}A \to \mathbf{K}B$
- $\lambda f.\lambda x.\mathbf{let\ pure}\, g, y = \mathbf{pure}\, f, x\ \mathbf{in}\ gy : (A \to B) \to \mathbf{K}A \to \mathbf{K}B$

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) $FV(pure\ M) = FV(M)$;
7) $FV(\mathbf{let\ pure}\ \vec{N} = \vec{M}\ \mathbf{in}\ M) = \bigcup_{i=1}^{n} FV(M)$, 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) $(\mathbf{pure}\, M)[x := P] = \mathbf{pure}\, (M[x := P])$;
7) $(\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N)[y := P] = \mathbf{let\ pure}\, \vec{x} = (\vec{M}[y := P])\ \mathbf{in}\ M$.

Definition 8. *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* $\alpha \in \{\to, \times\}$*;*

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

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

Definition 9. *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) $\begin{array}{l}\mathbf{let\ pure}\ \langle \vec{x}, y, \vec{z}\rangle = \langle \vec{M}, \mathbf{let\ pure}\ \vec{w} = \vec{N}\ \mathbf{in}\ Q, \vec{P}\rangle\ in\ R \to_\beta \\ \mathbf{let\ pure}\ \langle \vec{x}, \vec{w}, \vec{z}\rangle = \langle \vec{M}, \vec{N}, \vec{P}\rangle\ \mathbf{in}\ R[y := Q]\end{array}$

5) $\mathbf{let\ pure}\ \vec{x} = \mathbf{pure}\ \vec{M}\ \mathbf{in}\ N \to_\beta \mathbf{pure}\ N[\vec{x} := \vec{M}]$

6) $\lambda x.fx \to_\eta f$*;*

7) $\langle \pi_1 P, \pi_2 P\rangle \to_\eta P$*;*

8) $\mathbf{let\ pure}\ x = M\ \mathbf{in}\ x \to_\eta M$*;*

By default we use call-by-name evaluation strategy.

3 Basic lemmas

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

Lemma 3. *Generation for $\mathbf{K}_I$.*

Let $\Gamma \vdash \mathbf{pure}\ M : \mathbf{K}A$*, then* $\Gamma \vdash M : A$*;*

Proof. Induction on the structure of $\mathbf{pure}\ M$. □

Lemma 4. *Basic lemmas .*

i) Let $\Gamma \vdash M : A$ *and* $\Gamma \subseteq \Delta$*, then* $\Delta \vdash M : A$*;*

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

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

Proof.

i-ii-iii) Induction on $\Gamma \vdash M : A$.

□

Theorem 1. *Subject reduction*

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

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

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

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

See [19].

□

[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

Theorem 2.
$\twoheadrightarrow_\beta$ is strongly normalizing;

Proof.
i) Strong normalization for **IK** was proved by Kakutani for call-by-value and for call-by name [19] [20].
$\square$

Theorem 3.
$\twoheadrightarrow_\beta$ is confluent.

Proof. We modify and apply Barendregt's technique with term underlying. We will consider the fragment of the grammar for terms without constructors for pairs for simplicity.

Definition 10. *The set of underlined terms.*

- $x \in \mathbb{V} \Rightarrow x \in \underline{\Lambda}$;
- $M \in \underline{\Lambda} \Rightarrow (\lambda x.M) \in \underline{\Lambda}$;
- $M, N \in \underline{\Lambda} \Rightarrow (MN) \in \underline{\Lambda}$;
- $M \in \underline{\Lambda} \Rightarrow (\mathbf{pure}\, M) \in \underline{\Lambda}$;
- $\vec{x} \in \mathbb{V}, \vec{M}, N \in \underline{\Lambda} \Rightarrow \mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N \in \underline{\Lambda}$;
- $M, N \in \underline{\Lambda} \Rightarrow (\lambda_i x.M)N \in \underline{\Lambda}$, *for all* $i \in \mathbb{N}$.

Definition 11. *Subsitution for term with labelled lambda:*
$((\lambda_i x.M)N)[y := Z] = (\lambda_i x.M[y := Z])(N[y := Z])$

Definition 12. *Index erasing*
Let us define map $|.| : \underline{\Lambda} \to \Lambda$ *as follows:*

- $|x| = x$;
- $|\lambda x.M| = \lambda x.|M|$;
- $|MN| = |M||N|$;
- $|\mathbf{pure}\, M| = \mathbf{pure}\, |M|$;
- $|\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N| = \mathbf{let\ pure}\, \vec{x} = |\vec{M}|\ \mathbf{in}\ |N|$;
- $|(\lambda_i x.M)N| = (\lambda x.M)N$

Definition 13. *Reduction rules:*

- $(\lambda x.M)N \to_{\underline{\beta}} M[x := N]$;
- $$\begin{array}{l}\mathbf{let\ pure}\, \langle \vec{x}, y, \vec{z} \rangle = \langle \vec{M}, \mathbf{let\ pure}\, \vec{w} = \vec{N}\ \mathbf{in}\ Q, \vec{P} \rangle\ in\ R \to_{\underline{\beta}} \\ \mathbf{let\ pure}\, \langle \vec{x}, \vec{w}, \vec{z} \rangle = \langle \vec{M}, \vec{N}, \vec{P} \rangle\ \mathbf{in}\ R[y := Q]\end{array} ;$$
- $\mathbf{let\ pure}\, \vec{x} = \mathbf{pure}\, \vec{M}\ \mathbf{in}\ N \to_{\underline{\beta}} \mathbf{pure}\, N[\vec{x} := \vec{M}]$;
- $(\lambda x_i.M)N \to_{\underline{\beta}} M[x := N]$

$\twoheadrightarrow_{\underline{\beta}}$ is a reflexive-transitive closure of $\to_{\underline{\beta}}$.

Definition 14. *Indexed redex erasing:*
Let us define the next map $\phi : \underline{\Lambda} \to \Lambda$:

- $\phi(x) = x$;
- $\phi(\lambda x.M) = \lambda x.\phi(M)$;
- $\phi(MN) = \phi(M)\phi(N)$;
- $\phi(\mathbf{pure}\, M) = \mathbf{pure}\, \phi(M)$;
- $\phi(\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N) = \mathbf{let\ pure}\, \vec{x} = \phi(\vec{M})\ \mathbf{in}\ \phi(N)$;
- $\phi((\lambda_i x.M)N) = M[x := N]$

Lemma 5. $\forall \underline{M}, \underline{N} \in \underline{\Lambda}\ \forall M, N \in \Lambda$, *if* $|\underline{M}| = M, |\underline{N}| = N$, *then*

- *If* $M \twoheadrightarrow_{\beta} N$, *then* $\underline{M} \twoheadrightarrow_{\underline{\beta}} \underline{N}$
- *Vice versa*

Proof. Induction on the generation $\to_{\beta}$ and $\to_{\underline{\beta}}$ correspondently. The general statement follows from transitivity of multi-step reductions of both types. □

Lemma 6. $\phi(M[x := N]) = \phi(M)[x := \phi(N)]$.

Proof. We treat only cases with **pure** and with **let**. For the rest cases see [15].
1)

$$\begin{array}{ll}
\phi(\mathbf{pure}\,(M[x := N])) = & \text{By the definition of } \phi \\
\mathbf{pure}\,(\phi(M[x := N])) = & \text{Induction hypothesis} \\
\mathbf{pure}\,(\phi(M)[x := \phi(N)]) = & \text{Substitution definition} \\
(\mathbf{pure}\,\phi(M))[x := \phi(N)] &
\end{array}$$

2)

$$\begin{array}{ll}
\phi((\mathbf{let\ pure}\, \vec{x} = \vec{M}\ \mathbf{in}\ N)[y := P]) = & \text{Substitution definition} \\
\phi(\mathbf{let\ pure}\, \vec{x} = (\vec{M}[y := P])\ \mathbf{in}\ N) = & \text{By the definition of } \phi \\
\mathbf{let\ pure}\, \vec{x} = \phi(\vec{M}[y := P])\ \mathbf{in}\ \phi(N) = & \text{Induction hypothesis} \\
\mathbf{let\ pure}\, \vec{x} = (\phi(\vec{M})[y := \phi(P)])\ \mathbf{in}\ \phi(N) = & \text{Substitution definition} \\
(\mathbf{let\ pure}\, \vec{x} = \phi(\vec{M})\ \mathbf{in}\ \phi(N))[y := \phi(P)] &
\end{array}$$

□

Lemma 7.

- *If* $M \twoheadrightarrow_{\underline{\beta}} N$, *then* $\phi(M) \twoheadrightarrow_{\beta} \phi(N)$
- *If* $|M| = N$ *and* $\phi(M) = P$, *then* $N \twoheadrightarrow_{\beta} P$.

Proof.
i) Induction on the generation of $\twoheadrightarrow_{\underline{\beta}}$ using previous lemma.
ii) Induction on the structure of M. □

Lemma 8. *Strip lemma.*

If $M \to_{\beta} N$ *and* $M \twoheadrightarrow_{\beta} P$. *Then there exists some term* Q, *such that* $N \twoheadrightarrow_{\beta} Q$ *and* $P \twoheadrightarrow_{\beta} Q$.

Proof. Proof is similar to [15] [18]. We build the following diagram, which commutes by lemmas 5 and 7.

$$\begin{tikzcd}
M \arrow[rr, two heads] \arrow[dd, "\beta"'] & & P \arrow[dd, dashed, two heads, "\beta"] & \\
& N' \arrow[ul, "|.|"] \arrow[dl, "\phi"] \arrow[rr, dashed, two heads, "\beta"] & & P' \arrow[ul, "|.|"'] \arrow[dl, "\phi"] \\
N \arrow[rr, dashed, two heads, "\beta"'] & & Q &
\end{tikzcd}$$

□

Corollary 1. *If $M \twoheadrightarrow_\beta N$ and $M \twoheadrightarrow_\beta P$. Then there exists some term Q, such that $N \twoheadrightarrow_\beta Q$ and $P \twoheadrightarrow_\beta Q$.*

Proof. Unfold $M \twoheadrightarrow_\beta N$ as the sequence of one-step reduction and apply strip lemma on the every step. □

□

Theorem 4.
Normal form in $\lambda_{\mathbf{K}}$ has the subformula property.

Proof. By induction on the structure of term. Case with **let pure** $\vec{x} = \vec{M}$ **in** N was considered by Kakutani [19] [20]. Similary, if **pure** M is a normal form, so M is a normal form too by hypothesis. □

4 Categorical semantics

Definition 15. *Monoidal functor*
Let $\langle \mathcal{C}, \otimes_1, \mathbb{1} \rangle$ and $\langle \mathcal{D}, \otimes_2, \mathbb{1}' \rangle$ are monoidal categories.
A 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, which satisfy the well-known conditions described in [23]:
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 16. *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 monoidal and $\eta : Id_{\mathcal{C}} \Rightarrow \mathcal{K}$ is a natural transformation (similar to unit in monad), such that:
1) $u = \eta_{\mathbb{1}}$;
2) $*_{A,B} \circ (\eta_A \otimes \eta_B) = \eta_{A \otimes B}$;
3) Weak commutativity condition:

$$\begin{tikzcd}
A \otimes \mathcal{K}B \arrow[r, "\eta_A \otimes id_{\mathcal{K}B}"] \arrow[d, "\sigma_{A,\mathcal{K}B}"'] & \mathcal{K}A \otimes \mathcal{K}B \arrow[r, "*_{A,B}"] & \mathcal{K}(A \otimes B) \arrow[d, "\mathcal{K}(\sigma_{A,B})"] \\
\mathcal{K}B \otimes A \arrow[r, "id_{\mathcal{K}B} \otimes \eta_A"'] & \mathcal{K}B \otimes \mathcal{K}A \arrow[r, "*_{B,A}"'] & \mathcal{K}(B \otimes A)
\end{tikzcd}$$

4.1 Soundness and completeness

Theorem 5. *Soundness*
Let $\Gamma \vdash M : A$ and $M =_{\beta\eta} N$, then $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$

Proof.

Definition 17. *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:

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

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

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

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

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

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

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

Definition 18. *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 9.

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

Proof.

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

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

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 10.

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

Proof.

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

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

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

□

□

Theorem 6. *Completeness*
Let $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$*, then* $M =_{\beta\eta} N$.

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

Definition 19. *Let us define an endofunctor* $\mathcal{K} : \mathcal{C}(\lambda) \to \mathcal{C}(\lambda)$*, such that forall* $[x, M] \in Hom_{\mathcal{C}(\lambda)}(A, B), \mathbf{K}([x, M]) = [y, \textbf{let pure}\ x = y\ \textbf{in}\ M] \in Hom_{\mathcal{C}(\lambda)}(\mathbf{K}A, \mathbf{K}B)$ *(denotation: fmap f for an arbitrary arrow f).*

Lemma 11. *Functoriality*
i) fmap $(g \circ f) =$ *fmap* $(g) \circ$ *fmap* (f)*;*
ii) fmap $(id_A) = id_{\mathbf{K}A}$.

Proof. Easy checking using reduction rules.

□

Definition 20. *Let us define natural transformations:*
1) $\eta : Id \Rightarrow \mathcal{K}$*, s. t.* $\forall A \in Ob_{\mathcal{C}(\lambda)}$, $\eta_A = [x, \textbf{pure}\ x] \in Hom_{\mathcal{C}(\lambda)}(A, \mathbf{K}A)$*;*
2) $*_{A,B} : \mathbf{K}A \times \mathbf{K}B \to \mathbf{K}(A \times B)$*, s. t.* $\forall A, B \in Ob_{\mathcal{C}(\lambda)}, *_{A,B} = [p, \textbf{let pure}\ x, y = \pi_1 p, \pi_2 p\ \textbf{in}\ \langle x, y \rangle] \in Hom_{\mathcal{C}(\lambda)}(\mathbf{K}A \times \mathbf{K}B, \mathbf{K}(A \times B))$.

Implementation for $*$ in our term model is a modification of let$_{\mathbf{K}}$-rule:

$$\dfrac{\dfrac{p : \mathbf{K}A \times \mathbf{K}B \vdash p : \mathbf{K}A \times \mathbf{K}B}{p : \mathbf{K}A \times \mathbf{K}B \vdash \pi_1 p : \mathbf{K}A} \quad \dfrac{p : \mathbf{K}A \times \mathbf{K}B \vdash p : \mathbf{K}A \times \mathbf{K}B}{p : \mathbf{K}A \times \mathbf{K}B \vdash \pi_2 p : \mathbf{K}B} \quad \dfrac{x : A \vdash x : A \qquad y : B \vdash y : B}{x : A, y : B \vdash \langle x, y \rangle : A \times B}}{p : \mathbf{K}A \times \mathbf{K}B \vdash \textbf{let pure}\ \langle x, y \rangle = \langle \pi_1 p, \pi_2 p \rangle\ \textbf{in}\ \langle x, y \rangle : \mathbf{K}(A \times B)}$$

Lemma 12. *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 $f \circ \eta_A = \eta_B \circ f$

$\eta_B \circ f =$	By the definition
$[y, \mathbf{pure}\, y] \circ [x, M] =$	By the definition of composition
$[x, \mathbf{pure}\, y[y := M]] =$	By substitution
$[x, \mathbf{pure}\, M]$	

On the other hand:

fmap $f \circ \eta_A =$	By the definiton
$[z, \mathbf{let\ pure}\, x = z\ \mathbf{in}\, M] \circ [x, \mathbf{pure\ x}] =$	By the definition of composition
$[x, \mathbf{let\ pure}\, x = z\ \mathbf{in}\, M[z := \mathbf{pure}\, x]] =$	By substitution
$[x, \mathbf{let\ pure}\, x = \mathbf{pure\ x\ in}\, M] =$	β-reduction rule
$[x, \mathbf{pure}\, M[x := x]] =$	By substitution
$[x, \mathbf{pure}\, M]$	

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

See [19].

iii) $*_{A,B} \circ (\eta_A \times \eta_B) = \eta_{A \times B}$

$*_{A,B} \circ (\eta_A \times \eta_B) =$
By unfolding
$[q, \mathbf{let\ pure}\, x, y = \pi_1 q, \pi_2 q\ \mathbf{in}\, \langle x, y\rangle] \circ [p, \langle \mathbf{pure}\, (\pi_1 p), \mathbf{pure}\, (\pi_2 p)\rangle] =$
Composition
$[p, \mathbf{let\ pure}\, x, y = \pi_1 q, \pi_2 q\ \mathbf{in}\, \langle x, y\rangle[q := \langle \mathbf{pure}\, (\pi_1 p), \mathbf{pure}\, (\pi_2 p)\rangle]] =$
By substitution
$[p, \mathbf{let\ pure}\, x, y = \pi_1(\langle \mathbf{pure}\, (\pi_1 p), \mathbf{pure}\, (\pi_2 p)\rangle), \pi_2(\langle \mathbf{pure}\, (\pi_1 p), \mathbf{pure}\, (\pi_2 p)\rangle)\ \mathbf{in}\, \langle x, y\rangle] =$
Reduction rules
$[p, \mathbf{let\ pure}\, x, y = \mathbf{pure}\, (\pi_1 p), \mathbf{pure}\, (\pi_2 p)\ \mathbf{in}\, \langle x, y\rangle] =$
Reduction rule
$[p, \mathbf{pure}\, (\langle x, y\rangle[x := \pi_1 p, y := \pi_2 p])] =$
Substitution
$[p, \mathbf{pure}\, \langle \pi_1 p, \pi_2 p\rangle] =$
η-reduction
$[p, \mathbf{pure}\, p] =$
By definition
$\eta_{A \times B}$

□

Tensorial strength is defined as follows:

Definition 21. *Tensorial strength*

Let $[p, \langle \mathbf{pure}\, (\pi_1 p), \pi_2 p\rangle] \in Hom_{\mathcal{C}(\lambda)}(A \times \mathbf{K}B, \mathbf{K}(A \times B))$.

So tensorial strength is defined as $\tau_{A,B} = *_{A,B} \circ [p, \langle \mathbf{pure}\, (\pi_1 p), \pi_2 p\rangle]$.

It is clearly that tensorial strength defined above can be simplified as follows:

$*_{A,B} \circ [p, \langle \textbf{pure}\,(\pi_1 p), \pi_2 p\rangle] =$	By definition
$[p', \textbf{let pure}\, x, y = \pi_1 p', \pi_2 p' \textbf{ in } \langle x, y\rangle] \circ [p, \langle \textbf{pure}\,(\pi_1 p), \pi_2 p\rangle] =$	By composition
$[p, \textbf{let pure}\, x, y = \pi_1 p', \pi_2 p' \textbf{ in } \langle x, y\rangle[p' := \langle \textbf{pure}\,(\pi_1 p), \pi_2 p\rangle]] =$	By substitution
$[p, \textbf{let pure}\, x, y = \pi_1(\langle \textbf{pure}\,(\pi_1 p), \pi_2 p\rangle), \pi_2(\langle \pi_1 p, \textbf{pure}\,(\pi_2 p)\rangle) \textbf{ in } \langle x, y\rangle] =$	By β-reduction rules
$[p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle x, y\rangle]$	

Lemma 13. *Weak commutativity.*
$fmap\ ([p, \langle \pi_2 p, \pi_1 p\rangle]) \circ \tau_{A,B} =$
$*_{B,A} \circ [q, \langle \pi_1 q, \textbf{pure}\,(\pi_2 q)\rangle] \circ [p, \langle \pi_2 p, \pi_1 p\rangle]$

Proof.
$\text{fmap}\ ([r, \langle \pi_2 r, \pi_1 r\rangle]) \circ \tau_{A,B} =$
By the definition of τ
$\text{fmap}\ ([r, \langle \pi_2 r, \pi_1 r\rangle]) \circ [p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle x, y\rangle] =$
By the definition of fmap
$[q, \textbf{let pure}\, r = q \textbf{ in } \langle \pi_2 r, \pi_1 r\rangle] \circ [p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle x, y\rangle] =$
Composition
$[p, \textbf{let pure}\, r = q \textbf{ in } \langle \pi_2 r, \pi_1 r\rangle[q := \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle x, y\rangle]] =$
By β-reduction rules
$[p, \textbf{let pure}\, r = (\textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle x, y\rangle) \textbf{ in } \langle \pi_2 r, \pi_1 r\rangle] =$
By β-reduction rules
$[p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle \pi_2 r, \pi_1 r\rangle[r := \langle x, y\rangle]] =$
By substitution
$[p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle \pi_2\langle x, y\rangle, \pi_1\langle x, y\rangle\rangle] =$
By β-reduction rules
$[p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle y, x\rangle] =$

On the other hand
$*_{B,A} \circ [q, \langle \pi_1 q, \textbf{pure}\,(\pi_2 q)\rangle] \circ [p, \langle \pi_2 p, \pi_1 p\rangle] =$
By the definition of $*$
$[r, \textbf{let pure}\, y, x = \pi_1 r, \pi_2 r \textbf{ in } \langle y, x\rangle] \circ [q, \langle \pi_1 q, \textbf{pure}\,(\pi_2 q)\rangle] \circ [p, \langle \pi_2 p, \pi_1 p\rangle] =$
Composition
$[r, \textbf{let pure}\, y, x = \pi_1 r, \pi_2 r \textbf{ in } \langle y, x\rangle] \circ [p, \langle \pi_1 q, \textbf{pure}\,(\pi_2 q)\rangle[q := \langle \pi_2 p, \pi_1 p\rangle]] =$
By subsitution and by β-reduction rules
$[r, \textbf{let pure}\, y, x = \pi_1 r, \pi_2 r \textbf{ in } \langle y, x\rangle] \circ [p, \langle \pi_2 p, \textbf{pure}\,(\pi_1 p)\rangle]] =$
Composition
$[p, \textbf{let pure}\, y, x = \pi_1 r, \pi_2 r \textbf{ in } \langle y, x\rangle[r := \langle \pi_2 p, \textbf{pure}\,(\pi_1 p)\rangle]] =$
By subsitution and by β-reduction rules
$[p, \textbf{let pure}\, y, x = \pi_2 p, \textbf{pure}\,(\pi_1 p) \textbf{ in } \langle y, x\rangle] =$
By symmetricity of assingment
$[p, \textbf{let pure}\, x, y = \textbf{pure}\,(\pi_1 p), \pi_2 p \textbf{ in } \langle y, x\rangle]$

□

Lemma 14. **K** *is an applicative functor*

Proof. Immediately follows from previous lemmas in the section. □

□

5 Monadic extention

In modern Haskell `Monad` class is an extention of the `Applicative` class, where monadic binding (`(>>=)`) is the single basic class method:

```
class Applicative m => Monad (m :: * -> *) where
  (>>=) :: m a -> (a -> m b) -> m b
  (>>) :: m a -> m b -> m b
  return :: a -> m a
  fail :: String -> m a
```

From a logical point of view we may consider this extention as the extention of IEL$^-$ with the imdepotency axiom.

Definition 22. *"Monadic" 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);*
4) $\mathbf{KK}A \to \mathbf{K}A$
Rule: MP.

Categorically `Monad` class is a Kleisli triple on monoidal category, or, which is the same, strong monoidal monad:

Definition 23. *Strong monoidal monad*

Definition 24. *Monadic* $\lambda_{\mathbf{K}}$
Monadic $\lambda_{\mathbf{K}}$ *is a* $\lambda_{\mathbf{K}}$ *with the next additional typing rule:*

$$\frac{\Gamma \vdash M : \mathbf{KK}A}{\Gamma \vdash \mathbf{join}\, M : \mathbf{K}A}\ \mathbf{K}^2\text{-}e$$

Definition 25.

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] Lambek, J. and Scott P.J. (1986) Introduction to Higher Order Categorical Logic, Cambridge Studies in Advanced Mathematics 7, Cambridge: Cambridge University Press.

[23] Samuel Eilenberg and Max Kelly, Closed categories. Proc. Conf. Categorical Algebra (La Jolla, Calif., 1965).