

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 intuitionistic epistemic logic IEL was proposed by S. Artemov and T. Protopopescu [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) $\mathbf{K}(A \rightarrow B) \rightarrow (\mathbf{K}A \rightarrow \mathbf{K}B)$ (*normality*);
 - 3) $A \rightarrow \mathbf{K}A$ (*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 classes¹ 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
```

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

²Reader 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 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 λ_K) which is Curry-Howard isomorphic to IEL^- . So we will consider K -modality as an arbitrary applicative functor.

λ_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^- :

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

$$\frac{\Gamma \vdash A}{\Gamma \vdash KA} K_I \qquad \frac{\Gamma \vdash K\vec{A} \quad \vec{A} \vdash B}{\Gamma \vdash KB}$$

Where $\Gamma \vdash K\vec{A}$ is a syntax sugar for $\Gamma \vdash KA_1, \dots, \Gamma \vdash KA_n$.

Lemma 1. $\Gamma \vdash_{NIEL_{\wedge, \rightarrow}^-} A \Rightarrow IEL^- \vdash \bigwedge \Gamma \rightarrow A$.

Proof. Induction on the derivation.

Let us consider cases with modality.

- 1) If $\Gamma \vdash_{NIEL_{\wedge, \rightarrow}^-} A$, then $IEL^- \vdash \bigwedge \Gamma \rightarrow KA$.
 - (1) $\bigwedge \Gamma \rightarrow A$ assumption
 - (2) $A \rightarrow KA$ co-reflection
 - (3) $(\bigwedge \Gamma \rightarrow A) \rightarrow ((A \rightarrow KA) \rightarrow (\bigwedge \Gamma \rightarrow KA))$ IPC theorem
 - (4) $(A \rightarrow KA) \rightarrow (\bigwedge \Gamma \rightarrow KA)$ from (1), (3) and MP
 - (5) $\bigwedge \Gamma \rightarrow KA$ 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

□

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. □

At the next step we build the typed lambda-calculus based on $NIEL_{\wedge, \rightarrow}^-$ by proof-assignment 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:

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

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:

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

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

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

$$\begin{array}{c}
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow_i \qquad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \rightarrow_e \\
\\
\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \times_i \qquad \frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i M : A_i} \times_e, i \in \{1, 2\} \\
\\
\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{pure} M : \mathbf{K}A} \mathbf{K}_I \qquad \frac{\Gamma \vdash \vec{M} : \mathbf{K}\vec{A} \quad \vec{x} : \vec{A} \vdash N : B}{\Gamma \vdash \mathbf{let pure} \vec{x} = \vec{M} \mathbf{in} N : \mathbf{K}B} \mathbf{let_K}
\end{array}$$

\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.

$\mathbf{let_K}$ is similar to the \square -rule in typed lambda calculus for intuitionistic normal modal logic \mathbf{IK} , which is described in [19].

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

Here are some examples of closed terms:

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

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) $FV(\mathbf{pure} M) = FV(M)$;
- 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) $(\mathbf{let pure} \vec{x} = \vec{M} \mathbf{in} N)[y := P] = \mathbf{let pure} \vec{x} = (\vec{M}[y := P]) \mathbf{in} N$.

Definition 8. *Type substitution*

The substitution 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 $\Gamma[B := C] = \{x : (A[B := C]) \mid x : A \in \Gamma\}$

Definition 9. *β -reduction and η -reduction rules for $\lambda\mathbf{K}$.*

- 1) $(\lambda x.M)N \rightarrow_\beta M[x := N]$;
- 2) $\pi_1 \langle M, N \rangle \rightarrow_\beta M$;
- 3) $\pi_2 \langle M, N \rangle \rightarrow_\beta N$;
- 4) $\text{let pure } \langle \vec{x}, y, \vec{z} \rangle = \langle \vec{M}, \text{let pure } \vec{w} = \vec{N} \text{ in } Q, \vec{P} \rangle \text{ in } R \rightarrow_\beta \text{let pure } \langle \vec{x}, \vec{w}, \vec{z} \rangle = \langle \vec{M}, \vec{N}, \vec{P} \rangle \text{ in } R[y := Q]$
- 5) $\text{let pure } \vec{x} = \text{pure } \vec{M} \text{ in } N \rightarrow_\beta \text{pure } N[\vec{x} := \vec{M}]$
- 6) $\lambda x.f x \rightarrow_\eta f$;
- 7) $\langle \pi_1 P, \pi_2 P \rangle \rightarrow_\eta P$;
- 8) $\text{let pure } x = M \text{ in } x \rightarrow_\eta M$;

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

3 Basic lemmas

Now we will prove standard lemmas for contexts in type systems³:

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

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

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

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 $M \rightarrow_{\beta\eta} N$, then $\Gamma \vdash N : A$

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

- 1) Let $\Gamma \vdash \text{let pure } \langle \vec{x}, y, \vec{z} \rangle = \langle \vec{M}, \text{let pure } \vec{w} = \vec{N} \text{ in } Q, \vec{P} \rangle \text{ in } R : \mathbf{K}B$, then $\Gamma \text{let pure } \langle \vec{x}, \vec{w}, \vec{z} \rangle = \langle \vec{M}, \vec{N}, \vec{P} \rangle \text{ in } R[y := Q] : \mathbf{K}B$
- 2) Let $\Gamma \vdash \text{let pure } x = M \text{ in } x : \mathbf{K}A$, then $\Gamma \vdash M : \mathbf{K}A$.
- See [19].
- 3) If the derivation ends in

³We will not prove cases with \rightarrow -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

$$\frac{\Gamma \vdash \mathbf{pure} \vec{M} : \mathbf{K}\vec{A} \quad \vec{x} : \vec{A} \vdash N : B}{\Gamma \vdash \mathbf{let pure} \vec{x} = \mathbf{pure} \vec{M} \mathbf{in} N : \mathbf{K}B}$$

So $\Gamma \vdash \vec{M} : \vec{A}$ by generation and $\Gamma \vdash N[\vec{x} := \vec{M}] : B$ by weakening and substitution.

Then we can transform this into the next derivation:

$$\frac{\Gamma \vdash N[\vec{x} := \vec{M}] : B}{\Gamma \vdash \mathbf{pure} N[\vec{x} := \vec{M}] : \mathbf{K}B} \mathbf{K}_I$$

□

Theorem 2.

\rightarrow_β is strongly normalizing;

Proof.

We modify and apply Tait's technique of logical relation for modalities. For strong normalization proof with Tait's method for simply typed lambda calculus see [13].

Definition 10. The set of strongly computable terms:

- $SC_A = \{M : A \mid M \text{ is strongly normalizing}\}$ for $A \in \mathbb{T}$;
- $SC_{A \rightarrow B} = \{M : A \rightarrow B \mid \forall N \in SC_A, MN \in SC_B\}$, for $A, B \in \mathbb{T}_K$
- $SC_{\mathbf{K}A} = \{M : \mathbf{K}A \mid M \text{ is strongly normalizing}\}$ for $A \in \mathbb{T}$;
- $SC_{\mathbf{K}(A \rightarrow B)} = \{M : \mathbf{K}(A \rightarrow B) \mid \forall f \in SC_{A \rightarrow B}, \forall x \in SC_A, \forall N \in SC_{\mathbf{K}A}, \mathbf{let pure} f, x = M, N \mathbf{in} fx \in SC_{\mathbf{K}B}\}$ for $A, B \in \mathbb{T}_K$.

Lemma 5.

- If $M \in SC_A$, then M is strongly normalizing;
- Let $M \in SC_A$ and $M \rightarrow_\beta N$, then $N \in SC_A$;
- Let N is non-introduced, $N \in SC_A$. Then, if $M \rightarrow_\beta N$, then $M \in SC_A$;

Proof. By induction on the structure of A .

1) $A \equiv \mathbf{K}A$, where $A \in \mathbb{T}$.

i) Follows from the definition;

ii) Immediately;

iii) Let N is non-introduced and $N \in SC_A$, such that $M \rightarrow_\beta N$. Any reduction path $M \rightarrow_\beta \dots$ passes through $M \rightarrow_\beta N$.

N is strongly normalizing, so M too.

2) $A \equiv \mathbf{K}(B \rightarrow C)$

i) Suppose $M \in SC_{\mathbf{K}(B \rightarrow C)}$. Let $N \in SC_{\mathbf{K}B}$. So $\mathbf{let pure} f, x = M, N \mathbf{in} fx \in SC_{\mathbf{K}C}$.

So M is strongly normalizing, since $\mathbf{let pure} f, x = M, N \mathbf{in} fx$ is strongly normalizing.

ii) Let $M_1 \in SC_{\mathbf{K}(B \rightarrow C)}$ and $M_1 \rightarrow_\beta M_2$. Fix $N \in SC_{\mathbf{K}B}$.

Then $\mathbf{let pure} f, x = M_1, N \mathbf{in} fx \in SC_{\mathbf{K}C}$.

Hence, **let pure** $f, x = M_1, N$ **in** $fx \in SC_{\mathbf{K}C} \rightarrow_\beta$ **let pure** $f, x = M_2, N$ **in** fx .
 So **let pure** $f, x = M_2, N$ **in** $fx \in SC_{\mathbf{K}C}$. Then $M_2 \in SC_{\mathbf{K}(B \rightarrow C)}$.
 iii) Let M_2 be non-introduced, $M_2 \in SC_A$ and $M_1 \rightarrow_\beta M_2$ \square

\square

Theorem 3.

\rightarrow_β 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 11. The set of underlined terms.

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

Definition 12. Substitution for term with labelled lambda:

$$((\lambda_i x.M)N)[y := Z] = (\lambda_i x.M[y := Z])(N[y := Z])$$

Definition 13. Index erasing

Let us define map $|\cdot| : \underline{\Lambda} \rightarrow \Lambda$ as follows:

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

Definition 14. Reduction rules:

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

$\rightarrow_{\underline{\beta}}$ is a reflexive-transitive closure of \rightarrow_β .

Definition 15. *Indexed redex erasing:*

Let us define the next map $\phi : \underline{\Lambda} \rightarrow \Lambda$:

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

Lemma 6. $\forall \underline{M}, \underline{N} \in \underline{\Lambda} \forall M, N \in \Lambda$, if $|\underline{M}| = M, |\underline{N}| = N$, then

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

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

Lemma 7. $\phi(M[x := N]) = \phi(M)[x := \phi(N)]$.

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

- 1)

$\phi(\text{pure } (M[x := N])) =$	By the definition of ϕ
$\text{pure } (\phi(M[x := N])) =$	Induction hypothesis
$\text{pure } (\phi(M)[x := \phi(N)]) =$	Substitution definition
$(\text{pure } \phi(M))[x := \phi(N)]$	
- 2)

$\phi((\text{let pure } \vec{x} = \vec{M} \text{ in } N)[y := P]) =$	Substitution definition
$\phi(\text{let pure } \vec{x} = (\vec{M}[y := P]) \text{ in } N) =$	By the definition of ϕ
$\text{let pure } \vec{x} = \phi(\vec{M}[y := P]) \text{ in } \phi(N) =$	Induction hypothesis
$\text{let pure } \vec{x} = (\phi(\vec{M})[y := \phi(P)]) \text{ in } \phi(N) =$	Substitution definition
$(\text{let pure } \vec{x} = \phi(\vec{M}) \text{ in } \phi(N))[y := \phi(P)]$	

\square

Lemma 8.

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

Proof.

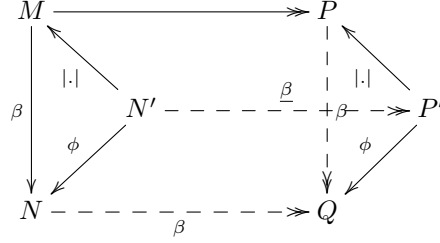
- i) Induction on the generation of $\rightarrow_{\underline{\beta}}$ using previous lemma.
- ii) Induction on the structure of M .

\square

Lemma 9. *Strip lemma.*

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

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



□

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 reductions and apply strip lemma on the every step.

□

□

Theorem 4.

Normal form in λ_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]. Similarly, if **pure** M is a normal form, so M is a normal form too by hypothesis.

□

4 Categorical semantics

Definition 16. *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 $\mathcal{F} : \langle \mathcal{C}, \otimes_1, \mathbb{1} \rangle \rightarrow \langle \mathcal{D}, \otimes_2, \mathbb{1}' \rangle$ is a functor $\mathcal{F} : \mathcal{C} \rightarrow \mathcal{D}$ with additional natural transformations, which satisfy the well-known conditions described in [23]:

- 1) $u : \mathbb{1}' \rightarrow \mathcal{F}\mathbb{1}$;
- 2) $*_{A,B} : \mathcal{F}A \otimes_2 \mathcal{F}B \rightarrow \mathcal{F}(A \otimes_1 B)$.

Definition 17. *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:*

$$\begin{array}{ccccc}
 A \otimes \mathcal{K}B & \xrightarrow{\eta_A \otimes id_{\mathcal{K}B}} & \mathcal{K}A \otimes \mathcal{K}B & \xrightarrow{*_{A,B}} & \mathcal{K}(A \otimes B) \\
 \sigma_{A,\mathcal{K}B} \downarrow & & & & \downarrow \mathcal{K}(\sigma_{A,B}) \\
 \mathcal{K}B \otimes A & \xrightarrow{id_{\mathcal{K}B} \otimes \eta_A} & \mathcal{K}B \otimes \mathcal{K}A & \xrightarrow{*_{B,A}} & \mathcal{K}(B \otimes A)
 \end{array}$$

4.1 Soundness and completeness

Theorem 5. Soundness

Let $\Gamma \vdash M : A$ and $M =_{\beta\eta} N$, then $\llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \vdash N : A \rrbracket$

Proof.

Definition 18. Semantical translation from λ_K to CCC with applicative functor \mathcal{K} :

1) Interpretation for types:

$$\llbracket A \rrbracket := \hat{A}, A \in \mathbb{T};$$

$$\llbracket A \rightarrow B \rrbracket := \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket;$$

$$\llbracket A \times B \rrbracket := \llbracket A \rrbracket \times \llbracket B \rrbracket.$$

2) Interpretation for modal types: $\llbracket \mathbf{K}A \rrbracket = \mathcal{K}\llbracket A \rrbracket$;

3) Interpretation for contexts:

$$\llbracket \Gamma = \{x_1 : A_1, \dots, x_n : A_n\} \rrbracket := \llbracket \Gamma \rrbracket = \llbracket A_1 \rrbracket \times \dots \times \llbracket A_n \rrbracket;$$

4) Interpretation for typing assignment: $\llbracket \Gamma \vdash M : A \rrbracket := \llbracket M \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$.

5) Interpretation for typing rules:

$$\begin{array}{c} \frac{}{\llbracket \Gamma, x : A \vdash x : A \rrbracket = \pi_2 : \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket \rightarrow \llbracket A \rrbracket} \\ \frac{\llbracket \Gamma, x : A \vdash M : B \rrbracket = f : \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket}{\llbracket \Gamma \vdash (\lambda x.M) : A \rightarrow B \rrbracket = \Lambda(f) : \llbracket \Gamma \rrbracket \rightarrow \llbracket B \rrbracket^{\llbracket A \rrbracket}} \\ \frac{\llbracket \Gamma \vdash M : A \rightarrow B \rrbracket = \llbracket M \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket B \rrbracket^{\llbracket A \rrbracket} \quad \llbracket \Gamma \vdash N : A \rrbracket = \llbracket N \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket}{\llbracket \Gamma \vdash (MN) : B \rrbracket = \llbracket \Gamma \rrbracket \xrightarrow{\langle \llbracket M \rrbracket, \llbracket N \rrbracket \rangle} \llbracket B \rrbracket^{\llbracket A \rrbracket} \times \llbracket A \rrbracket \xrightarrow{\epsilon} \llbracket B \rrbracket} \\ \frac{\llbracket \Gamma \vdash M : A \rrbracket = f : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket \quad \llbracket \Gamma \vdash N : B \rrbracket = g : \llbracket \Gamma \rrbracket \rightarrow \llbracket B \rrbracket}{\llbracket \Gamma \vdash (M, N) : A \times B \rrbracket = \langle f, g \rangle : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket \times \llbracket B \rrbracket} \\ \frac{\llbracket \Gamma \vdash p : A_1 \times A_2 \rrbracket = f : \llbracket \Gamma \rrbracket \rightarrow \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket}{\llbracket \Gamma \vdash \pi_i p : A_i \rrbracket = \llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket \xrightarrow{\pi_i} \llbracket A_i \rrbracket} \quad i \in \{1, 2\} \\ \frac{\llbracket \Gamma \vdash M : A \rrbracket = \llbracket M \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket}{\llbracket \Gamma \vdash \mathbf{pure} M : \mathbf{K}A \rrbracket := \llbracket \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket A \rrbracket \xrightarrow{\eta_{\llbracket A \rrbracket}} \mathcal{K}\llbracket A \rrbracket} \\ \frac{\llbracket \Gamma \vdash \vec{M} : \mathbf{K}\vec{A} \rrbracket = \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle : \llbracket \Gamma \rrbracket \rightarrow \prod_{i=1}^n \mathcal{K}\llbracket A_i \rrbracket \quad \llbracket \vec{x} : \vec{A} \vdash N : B \rrbracket = \llbracket N \rrbracket : \prod_{i=1}^n \llbracket A_i \rrbracket \rightarrow \llbracket B \rrbracket}{\llbracket \Gamma \vdash \mathbf{let pure} \vec{x} = \vec{M} \mathbf{in} M : \mathbf{K}B \rrbracket = \mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket, \dots, \llbracket A_n \rrbracket} \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle : \llbracket \Gamma \rrbracket \rightarrow \mathcal{K}\llbracket B \rrbracket} \end{array}$$

Definition 19. Simultaneous substitution

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

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) $(\mathbf{let} \mathbf{pure} \vec{x} = \vec{M} \mathbf{in} N)[\vec{y} := \vec{P}] = \mathbf{let} \mathbf{pure} \vec{x} = (\vec{M}[\vec{y} := \vec{P}]) \mathbf{in} N$

Lemma 10.

$$\llbracket M[x_1 := M_1, \dots, x_n := M_n] \rrbracket = \llbracket M \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle.$$

Proof.

$$1) \llbracket \Gamma \vdash (\mathbf{pure} M)[\vec{x} := \vec{M}] : \mathbf{KA} \rrbracket = \llbracket \Gamma \vdash \mathbf{pure} M : \mathbf{KA} \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle.$$

$$\begin{aligned} \llbracket \Gamma \vdash (\mathbf{pure} M)[\vec{x} := \vec{M}] : \mathbf{KA} \rrbracket &= \llbracket \Gamma \vdash \mathbf{pure} (M[\vec{x} := \vec{M}]) : \mathbf{KA} \rrbracket && \text{Substitution definition} \\ &= \eta_{\llbracket A \rrbracket} \circ \llbracket (M[\vec{x} := \vec{M}]) \rrbracket && \text{Translation for pure} \\ &= \eta_{\llbracket A \rrbracket} \circ (\llbracket M \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle) && \text{Induction hypothesis} \\ &= (\eta_{\llbracket A \rrbracket} \circ \llbracket M \rrbracket) \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle && \text{Associativity of composition} \\ &= \llbracket \Gamma \vdash \mathbf{pure} M : \mathbf{KA} \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle && \text{Translation for pure} \end{aligned}$$

$$2) \llbracket \Gamma \vdash (\mathbf{let} \mathbf{pure} \vec{x} = \vec{M} \mathbf{in} N)[\vec{y} := \vec{P}] : \mathbf{KB} \rrbracket = \llbracket \Gamma \vdash \mathbf{let} \mathbf{pure} \vec{x} = \vec{M} \mathbf{in} N : \mathbf{KB} \rrbracket \circ \langle \llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket \rangle$$

$$\begin{aligned} &\llbracket \Gamma \vdash (\mathbf{let} \mathbf{pure} \vec{x} = \vec{M} \mathbf{in} N)[\vec{y} := \vec{P}] : \mathbf{KB} \rrbracket = \\ &\text{Substitution definition} \\ &\llbracket \Gamma \vdash \mathbf{let} \mathbf{pure} \vec{x} = (\vec{M}[\vec{y} := \vec{P}]) \mathbf{in} N : \mathbf{KB} \rrbracket = \\ &\text{Interpretation for } \mathbf{let}_{\mathbf{K}} \\ &\mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket, \dots, \llbracket A_n \rrbracket} \circ \llbracket \Gamma \vdash (\vec{M}[\vec{y} := \vec{P}]) : \mathbf{KA} \rrbracket = \\ &\text{Induction hypothesis} \\ &\mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket, \dots, \llbracket A_n \rrbracket} \circ (\llbracket \vec{M} \rrbracket \circ \langle \llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket \rangle) = \\ &\text{Associativity of composition} \\ &(\mathcal{K}(\llbracket N \rrbracket) \circ *_{\llbracket A_1 \rrbracket, \dots, \llbracket A_n \rrbracket} \circ \llbracket \vec{M} \rrbracket) \circ \langle \llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket \rangle = \\ &\text{By interpretation} \\ &\llbracket \Gamma \vdash (\mathbf{let} \mathbf{pure} \vec{x} = \vec{M} \mathbf{in} N) \rrbracket \circ \langle \llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket \rangle \end{aligned}$$

□

Lemma 11.

Let $\Gamma \vdash M : A$ and $M \rightarrow_{\beta\eta} N$, then $\llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \vdash N : A \rrbracket$;

Proof.

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

$$1) \llbracket \Gamma \vdash \mathbf{let} \mathbf{pure} \vec{x} = \mathbf{pure} \vec{M} \mathbf{in} N : \mathbf{KB} \rrbracket = \llbracket \Gamma \vdash \mathbf{pure} N[\vec{x} := \vec{M}] : \mathbf{KB} \rrbracket$$

$$\begin{aligned}
& \llbracket \Gamma \vdash \text{let pure } \vec{x} = \text{pure } \vec{M} \text{ in } N : \mathbf{KB} \rrbracket = \\
& \quad \text{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 = \\
& \quad \text{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 = \\
& \quad \text{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 \times \eta_{\llbracket A_n \rrbracket})) \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle = \\
& \quad \text{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 = \\
& \quad \text{Naturality of } \eta \\
& \eta_{\llbracket B \rrbracket} \circ \llbracket N \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle = \\
& \quad \text{Associativity of composition} \\
& \eta_{\llbracket B \rrbracket} \circ (\llbracket N \rrbracket \circ \langle \llbracket M_1 \rrbracket, \dots, \llbracket M_n \rrbracket \rangle) = \\
& \quad \text{Simultaneous substitution lemma} \\
& \eta_{\llbracket B \rrbracket} \circ \llbracket N[\vec{x} := \vec{M}] \rrbracket \\
& \quad \text{By interpretation} \\
& \llbracket \Gamma \vdash \text{pure } (N[\vec{x} := \vec{M}]) : \mathbf{KB} \rrbracket
\end{aligned}$$

□

□

Theorem 6. Completeness

Let $\llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \vdash N : A \rrbracket$, then $M =_{\beta\eta} N$.

Proof.

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

Definition 20. Let us define an endofunctor $\mathcal{K} : \mathcal{C}(\lambda) \rightarrow \mathcal{C}(\lambda)$, such that for all $[x, M] \in \text{Hom}_{\mathcal{C}(\lambda)}(A, B)$, $\mathbf{K}([x, M]) = [y, \text{let pure } x = y \text{ in } M] \in \text{Hom}_{\mathcal{C}(\lambda)}(\mathbf{KA}, \mathbf{KB})$ (denotation: $fmap$ f for an arbitrary arrow f).

Lemma 12. Functoriality

- i) $fmap (g \circ f) = fmap (g) \circ fmap (f)$;
- ii) $fmap (id_A) = id_{\mathbf{KA}}$.

Proof. Easy checking using reduction rules.

□

Definition 21. Let us define natural transformations:

- 1) $\eta : Id \Rightarrow \mathcal{K}$, s. t. $\forall A \in \text{Ob}_{\mathcal{C}(\lambda)}$, $\eta_A = [x, \text{pure } x] \in \text{Hom}_{\mathcal{C}(\lambda)}(A, \mathbf{KA})$;
- 2) $*_{A,B} : \mathbf{KA} \times \mathbf{KB} \rightarrow \mathbf{K}(A \times B)$, s. t. $\forall A, B \in \text{Ob}_{\mathcal{C}(\lambda)}$, $*_{A,B} = [p, \text{let pure } x, y = \pi_1 p, \pi_2 p \text{ in } \langle x, y \rangle] \in \text{Hom}_{\mathcal{C}(\lambda)}(\mathbf{KA} \times \mathbf{KB}, \mathbf{K}(A \times B))$.

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

$$\frac{\frac{p : \mathbf{KA} \times \mathbf{KB} \vdash p : \mathbf{KA} \times \mathbf{KB}}{p : \mathbf{KA} \times \mathbf{KB} \vdash \pi_1 p : \mathbf{KA}} \quad \frac{p : \mathbf{KA} \times \mathbf{KB} \vdash p : \mathbf{KA} \times \mathbf{KB}}{p : \mathbf{KA} \times \mathbf{KB} \vdash \pi_2 p : \mathbf{KB}} \quad \frac{x : A \vdash x : A \quad y : B \vdash y : B}{x : A, y : B \vdash \langle x, y \rangle : A \times B}}{p : \mathbf{KA} \times \mathbf{KB} \vdash \text{let pure } \langle x, y \rangle = \langle \pi_1 p, \pi_2 p \rangle \text{ in } \langle x, y \rangle : \mathbf{K}(A \times B)}$$

Lemma 13. *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) $fmap f \circ \eta_A = \eta_B \circ f$

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

On the other hand:

$$\begin{aligned}
 fmap f \circ \eta_A &= && \text{By the definition} \\
 [z, \mathbf{let} \mathbf{pure} x = z \mathbf{in} M] \circ [x, \mathbf{pure} x] &= && \text{By the definition of composition} \\
 [x, \mathbf{let} \mathbf{pure} x = z \mathbf{in} M[z := \mathbf{pure} x]] &= && \text{By substitution} \\
 [x, \mathbf{let} \mathbf{pure} x = \mathbf{pure} x \mathbf{in} M] &= && \beta\text{-reduction rule} \\
 [x, \mathbf{pure} M[x := x]] &= && \text{By substitution} \\
 [x, \mathbf{pure} M] &= &&
 \end{aligned}$$

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

See [19].

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

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

□

Tensorial strength is defined as follows:

Definition 22. *Tensorial strength*

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

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:

$$\begin{aligned}
& *_{A,B} \circ [p, \langle \mathbf{pure}(\pi_1 p), \pi_2 p \rangle] = && \text{By definition} \\
& [p', \mathbf{let pure } x, y = \pi_1 p', \pi_2 p' \mathbf{ in } \langle x, y \rangle] \circ [p, \langle \mathbf{pure}(\pi_1 p), \pi_2 p \rangle] = && \text{By composition} \\
& [p, \mathbf{let pure } x, y = \pi_1 p', \pi_2 p' \mathbf{ in } \langle x, y \rangle [p' := \langle \mathbf{pure}(\pi_1 p), \pi_2 p \rangle]] = && \text{By substitution} \\
& [p, \mathbf{let pure } x, y = \pi_1(\langle \mathbf{pure}(\pi_1 p), \pi_2 p \rangle), \pi_2(\langle \pi_1 p, \mathbf{pure}(\pi_2 p) \rangle) \mathbf{ in } \langle x, y \rangle] = && \text{By } \beta\text{-reduction rules} \\
& [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle x, y \rangle]
\end{aligned}$$

Lemma 14. *Weak commutativity.*

$$\begin{aligned}
& fmap ([p, \langle \pi_2 p, \pi_1 p \rangle]) \circ \tau_{A,B} = \\
& *_{B,A} \circ [q, \langle \pi_1 q, \mathbf{pure}(\pi_2 q) \rangle] \circ [p, \langle \pi_2 p, \pi_1 p \rangle]
\end{aligned}$$

Proof.

$$\begin{aligned}
& fmap ([r, \langle \pi_2 r, \pi_1 r \rangle]) \circ \tau_{A,B} = \\
& \text{By the definition of } \tau \\
& fmap ([r, \langle \pi_2 r, \pi_1 r \rangle]) \circ [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle x, y \rangle] = \\
& \text{By the definition of } fmap \\
& [q, \mathbf{let pure } r = q \mathbf{ in } \langle \pi_2 r, \pi_1 r \rangle] \circ [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle x, y \rangle] = \\
& \text{Composition} \\
& [p, \mathbf{let pure } r = q \mathbf{ in } \langle \pi_2 r, \pi_1 r \rangle [q := \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle x, y \rangle]] = \\
& \text{By } \beta\text{-reduction rules} \\
& [p, \mathbf{let pure } r = (\mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle x, y \rangle) \mathbf{ in } \langle \pi_2 r, \pi_1 r \rangle] = \\
& \text{By } \beta\text{-reduction rules} \\
& [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle \pi_2 r, \pi_1 r \rangle [r := \langle x, y \rangle]] = \\
& \text{By substitution} \\
& [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle \pi_2 \langle x, y \rangle, \pi_1 \langle x, y \rangle \rangle] = \\
& \text{By } \beta\text{-reduction rules} \\
& [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle y, x \rangle] =
\end{aligned}$$

On the other hand

$$\begin{aligned}
& *_{B,A} \circ [q, \langle \pi_1 q, \mathbf{pure}(\pi_2 q) \rangle] \circ [p, \langle \pi_2 p, \pi_1 p \rangle] = \\
& \text{By the definition of } * \\
& [r, \mathbf{let pure } y, x = \pi_1 r, \pi_2 r \mathbf{ in } \langle y, x \rangle] \circ [q, \langle \pi_1 q, \mathbf{pure}(\pi_2 q) \rangle] \circ [p, \langle \pi_2 p, \pi_1 p \rangle] = \\
& \text{Composition} \\
& [r, \mathbf{let pure } y, x = \pi_1 r, \pi_2 r \mathbf{ in } \langle y, x \rangle] \circ [p, \langle \pi_1 q, \mathbf{pure}(\pi_2 q) \rangle [q := \langle \pi_2 p, \pi_1 p \rangle]] = \\
& \text{By substitution and by } \beta\text{-reduction rules} \\
& [r, \mathbf{let pure } y, x = \pi_1 r, \pi_2 r \mathbf{ in } \langle y, x \rangle] \circ [p, \langle \pi_2 p, \mathbf{pure}(\pi_1 p) \rangle]] = \\
& \text{Composition} \\
& [p, \mathbf{let pure } y, x = \pi_1 r, \pi_2 r \mathbf{ in } \langle y, x \rangle [r := \langle \pi_2 p, \mathbf{pure}(\pi_1 p) \rangle]] = \\
& \text{By substitution and by } \beta\text{-reduction rules} \\
& [p, \mathbf{let pure } y, x = \pi_2 p, \mathbf{pure}(\pi_1 p) \mathbf{ in } \langle y, x \rangle] = \\
& \text{By symmetricity of assingment} \\
& [p, \mathbf{let pure } x, y = \mathbf{pure}(\pi_1 p), \pi_2 p \mathbf{ in } \langle y, x \rangle]
\end{aligned}$$

□

Lemma 15. *K is an applicative functor*

Proof. Immediately follows from previous lemmas in the section.

□

□

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).