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 $\Gamma \vdash \mathbf{K}A_1, \dots, \Gamma \vdash \mathbf{K}A_n$.

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. □

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:

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

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

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

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

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

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

$\mathbf{K}_I$-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.

$\text{let}_{\mathbf{K}}$ 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.

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

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

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

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) $FV(\textbf{let pure } \vec{N} = \vec{M} \textbf{ in } M) = \bigcup_{i=1}^{n} FV(M)$*, 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) $\begin{array}{l}\textbf{let pure}\, \langle \vec{x}, y, \vec{z}\rangle = \langle \vec{M}, \textbf{let pure}\, \vec{w} = \vec{N} \,\textbf{in}\, Q, \vec{P}\rangle \textit{ in } R \to_\beta \\ \textbf{let pure}\, \langle \vec{x}, \vec{w}, \vec{z}\rangle = \langle \vec{M}, \vec{N}, \vec{P}\rangle \,\textbf{in}\, R[y := Q]\end{array}$
5) $M \to_\beta N \Rightarrow \textbf{pure}\, \mathbf{M} \to_\beta \textbf{pure}\, \mathbf{N}$
6) $\lambda x.fx \to_\eta f$*;*
7) $\langle \pi_1 P, \pi_2 P\rangle \to_\eta P$*;*
10) $\textbf{let pure}\, \_\_ = \_\_ \,\textbf{in}\, N \to_\eta \textbf{pure}\, N$*;*
11) $\textbf{let pure}\, x = M \,\textbf{in}\, x \to_\eta M$*;*
12) $M \to_\beta N \Rightarrow \textbf{pure}\, \mathbf{M} \to_\eta \textbf{pure}\, \mathbf{N}$

Let us show the next simple observation, which immeadelty follows from the previous definition.

Lemma 3.
If $M \to_{\beta\eta} N$*, then* $\textbf{pure}\, M \to_{\beta\eta} \textbf{pure}\, N$.

3 Basic lemmas

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

4 Strong normalization

We modify and apply Tait's technique of logical relation for modalities. Strong normalization proof with Tait's method for simply typed lambda calculus is described here [13].

Strong normalization for **IK** is proved in [21] [19]. So we consider simply typed lambda calculus with $\mathbf{K}_I$ rule and show that $\lambda_{\to,\times} + \mathbf{K}_I$ is strongly normalizable.

[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 1. *Let $M \in \Lambda_{\mathbf{K}}$, then any sequence of reduction $M \to_\beta M_1 \dots$ terminates.*

Proof.
We build the subset of strongly normalazing terms and show that an arbitrary term belongs to this subset.

Definition 9. *The set of strongly computable terms for every type $T \in \mathbb{T}_{\mathbf{K}}$.*

- *Let $A \in \mathbb{T}$, then $SC_A = \{M : A \mid M$ is strongly normalizing$\}$;*
- *$SC_{A \to B} = \{M : A \to B \mid \forall A \in SC_A, MN \in SC_B\}$;*
- *$SC_{A_1 \times A_2} = \{M : A \times B \mid \pi_i M \in SC_{A_i}, i \in \{1, 2\}\}$;*
- *$SC_{\mathbf{K}A} = \{\mathbf{pure}\, M : \mathbf{K}A \mid M \in SC_A\}$*

Strong normalization proof reduces to the proof of the next lemma:

Lemma 4.
i) If $M \in SC_A$, then M is stronly normalizing;
ii) If $M \to_\beta M^{'}$ and $M \in SC_A$, then $M^{'}$;
iii) Let $M \to_\beta M^{'}$ and $M^{'} \in SC_A$, then, if M is a neutral term, then $M \in SC_A$.
iv) Let $x_1 : A_1, \dots, x_n : A_n \vdash M : B$ and $\forall i \in \{1, \dots, n\}, N_i \in SC_{A_i}$, then $M[\vec{x} := \vec{N}] \in SC_B$.

Proof.
i)
The base case follows from the definition.
Let us consider case with $SC_{\mathbf{K}A}$. If $\mathbf{pure}\, M \in SC_{\mathbf{K}A}$, then $M \in SC_A$ and M is strongly normalizible. So $\mathbf{pure}\, M$ is strongly normalizible, otherwise there would be an infinite reduction path in $\mathbf{pure}\, M$.

ii)
The base case is trivial.
Let $\mathbf{pure}\, M \to_\beta \mathbf{pure}\, M^{'}$ and $\mathbf{pure}\, M \in SN_{\mathbf{K}A}$. By assumption, $M \in SN_A$ and $M \to_\beta M^{'}$, so $M^{'} \in SN_A$. Hence $\mathbf{pure}\, M^{'} \in SC_{\mathbf{K}A}$ by the first statement of the lemma.

iii)
The base case is trivial.
Let $\mathbf{pure}\, M \to_\beta \mathbf{pure}\, M^{'}$ and $\mathbf{pure}\, M^{'} \in SN_{\mathbf{K}A}$.
$\mathbf{pure}\, M^{'}$ is a neutral by the definition. By assumption M is a strongly normalizing. So $\mathbf{pure}\, M$ is a strongly normalizing by the first part of the current lemma.

iv)
Let $x_1 : A_1, \dots, x_n : A_n \vdash \mathbf{pure}\, M : \mathbf{K}A$ and $\forall i \in \{1, \dots, n\}, N_i \in SC_{A_i}$.
By generation $x_1 : A_1, \dots, x_n : A_n \vdash M : A$ and by assumption $M[\vec{x} := \vec{N}] \in SC_B$.
Hence, by the first part of lemma, $\mathbf{pure}\,(M[\vec{x} := \vec{N}]) \in SC_{\mathbf{K}B}$. □

Corollary 1.
Let $\vdash N : A$, then N is strongly normalizing.

Proof.
If $\vdash N : A$, then $N \in SC_A$, hence N is strongly normalizing. □

□

5 Confluence

6 Categorical semantics

Definition 10. *Lax monoidal functor*
Let $\langle \mathcal{C}, \oplus_1, \mathbb{1} \rangle$ and $\langle \mathcal{D}, \oplus_2, \mathbb{1}' \rangle$ are monoidal categories.
A lax monoidal functor $\mathcal{F} : \langle \mathcal{C}, \oplus_1, \mathbb{1} \rangle \to \langle \mathcal{D}, \oplus_2, \mathbb{1}' \rangle$ 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

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:

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

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

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

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

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

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

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

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 5.

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

Proof.

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

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

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

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

□

Lemma 6.
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) $[\![\Gamma \vdash \textbf{pure}\ ((\lambda x.M)N) : \mathbf{K}B]\!] = [\![\Gamma \vdash \textbf{pure}\ (M[x := N]]) : \mathbf{K}B]\!]$

$[\![\Gamma \vdash \textbf{pure}\ (\lambda x.M)N : \mathbf{K}B]\!] =$	By interpretation
$\eta_{[\![B]\!]} \circ (\epsilon \circ \langle \Lambda([\![M]\!]), [\![N]\!] \rangle) =$	Property of $\times$
$\eta_{[\![B]\!]} \circ (\epsilon \circ (\Lambda([\![M]\!]) \times id_{[\![A]\!]}) \circ \langle id_{[\![\Gamma]\!]}, [\![N]\!] \rangle) =$	Associativity of composition
$\eta_{[\![B]\!]} \circ ((\epsilon \circ (\Lambda([\![M]\!]) \times id_{[\![A]\!]})) \circ \langle id_{[\![\Gamma]\!]}, [\![N]\!] \rangle) =$	Exponentiation property
$\eta_{[\![B]\!]} \circ ([\![M]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![N]\!] \rangle) =$	Substitution lemma
$\eta_{[\![B]\!]} \circ [\![M[\vec{x}, x := \vec{x}, N]]\!] =$	By interpretation
$[\![\Gamma \vdash \textbf{pure}\ (M[x := N]]) : \mathbf{K}B]\!]$	

2) $[\![\Gamma \vdash \textbf{pure}\ (\pi_i \langle [\![M_1]\!], [\![M_2]\!] \rangle) : \mathbf{K}A_i]\!] = [\![\Gamma \vdash\ \textbf{pure} M_i : \mathbf{K}A_i]\!]$

$[\![\Gamma \vdash \textbf{pure}\ (\pi_i \langle M_1, M_2]\rangle) : \mathbf{K}A_i]\!] =$	By interpretation
$\eta_{[\![A_i]\!]} \circ \pi_i \circ \langle [\![M_1]\!], [\![M_2]\!] \rangle =$	Property of $\times$
$\eta_{[\![A_i]\!]} \circ [\![M_i]\!] =$	By interpetation
$[\![\Gamma \vdash \textbf{pure}\ M_i : \mathbf{K}A_i]\!]$	

ii) For η-reduction.

1) $[\![\Gamma \vdash \textbf{pure}\ (\lambda x.Mx) : \mathbf{K}(A \to B)]\!] = [\![\Gamma \vdash \textbf{pure}\ M : \mathbf{K}(A \to B)]\!]$.

$[\![\Gamma \vdash \textbf{pure}\ (\lambda x.Mx) : \mathbf{K}(A \to B)]\!] =$	By interpetation
$\eta_{[\![B]\!]^{[\![A]\!]}} \circ \Lambda(\epsilon \circ [\![M]\!] \times id_{[\![A]\!])}$	Exponentiation property
$\eta_{[\![B]\!]^{[\![A]\!]}} \circ [\![M]\!] =$	By interpretation
$[\![\Gamma \vdash \textbf{pure}\ M : \mathbf{K}(A \to B)]\!]$	

2) $[\![\Gamma \vdash \textbf{pure}\ \langle \pi_1 M, \pi_2 M \rangle : \mathbf{K}(A \times B)]\!] = [\![\Gamma \vdash \textbf{pure}\ M : \mathbf{K}(A \times B)]\!]$

$$
\begin{array}{ll}
[\![\Gamma \vdash \textbf{pure}\ \langle \pi_1 M, \pi_2 M \rangle : \mathbf{K}(A \times B)]\!] = & \text{By interpetation} \\
\eta_{[\![A]\!] \times [\![B]\!]} \circ \langle \pi_1 \circ [\![M]\!], \pi_2 \circ [\![M]\!] \rangle = & \text{By the property of a product of morphisms} \\
\eta_{[\![A]\!] \times [\![B]\!]} \circ [\![M]\!] = & \text{By interpetation} \\
[\![\Gamma \vdash \textbf{pure}\ M : \mathbf{K}(A \times B)]\!] &
\end{array}
$$

3) $[\![\vdash \textbf{let pure}\ \_\_ = \_\_\ \textbf{in}\ N : KA]\!] = [\![\vdash \textbf{pure}\ N : \mathbf{KA}]\!]$.

$$
\begin{array}{ll}
[\![\vdash \textbf{let pure}\ \_\_ = \_\_\ \textbf{in}\ N : KA]\!] = & \text{By interpetation} \\
\mathcal{K}([\![N]\!]) \circ \eta_{\mathbb{1}} = & \text{Naturality for } \eta \\
\eta_{[\![A]\!]} \circ [\![N]\!] = & \text{By interpretation} \\
[\![\vdash \textbf{pure}\ N : \mathbf{K}A]\!] &
\end{array}
$$

□

Theorem 2. *Soundness*
Let $\Gamma \vdash M : A$ *and* $M =_{\beta\eta} N$, *then* $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$

Proof. Straightforwardly follows from two previous lemmas. □

7 Acknowledgement.

Author would like to thank his supervisior V.L.Vasukov, V.N. Krupski for general idea and wise advice, V. de Paiva, V.I. Shalack, A.V. Rodin and M. Taldykin for discussing, critics and consulting.

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.