POLYMORPHIC TYPE THEORY AS A BASE FOR CATEGORICAL LOGIC

Valerie NOVITZKÁ, Daniel MIHÁLYI
Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University Košice, Letná 9, 042 10 Košice
e-mail: Valerie.Novitzka@tuke.sk, Daniel.Mihalyi@tuke.sk

SUMMARY

In this paper we present construction of polymorphic type theory and its model in terms of category theory. We extend the notion of many-typed signature to polymorphic type signature. We define polymorphic type calculi for first-, second- and higher-order types and set-theoretical semantics of first-order polymorphism. Semantics for higher-order polymorphic type calculi we construct in categorical terms. Then we can build a logical system over polymorphic type theory as composed polymorphic fibration with double indexing.

***Keywords*:** *polymorphic type theory, fibration, categorical logic*

1. INTRODUCTION

In our previous papers [6,7,8] we worked out a part of mathematical theory of programming where we consider programming as logical reasoning over types. We started with many-typed signature $\Sigma=(T,F)$ for a solved problem consisting of basic types $\sigma, \tau, \ldots$ and function symbols of the form $f: \sigma_1,\ldots,\sigma_n \rightarrow \tau$. Using constructors '×', '+' and '→' we built Church's types, namely product types $\sigma \times \tau$, coproduct (sum) types $\sigma+\tau$ and function (exponent) types $\sigma \rightarrow \tau$. We denoted by a sequent

$$\Gamma \mid\!- t:\tau$$

a term t of type τ in which can occur typed variables declared in type context

$$\Gamma = (v_1:\sigma_1,\ldots,v_n:\sigma_n).$$

We enclosed Church's types over a signature Σ into classifying category $Cl(\Sigma)$ consisting of type contexts as category objects and terms $\Gamma \mid\!- t:\tau$ as category morphisms $t: \Gamma \rightarrow \Delta$ between contexts. The classifying category $Cl(\Sigma)$ has finite products $\Gamma \times \Delta$, finite coproducts $\Gamma + \Delta$, exponent objects Γ^{Δ} and terminal object 1, (empty product, empty type context). Therefore $Cl(\Sigma)$ is bicartesian closed category (biccc). We consider logic always over types. Therefore we constructed logical system of equational first-order logic $\mathcal{L}(\Sigma,\Pi,\mathcal{A})$ as a preordered fibration over base category $Cl(\Sigma)$ as in Fig.1.

The symbol Π in Fig.1 denotes predicate symbols, $\mathcal{A}$ is a set of equational axioms. A fibration is a special functor expressing indexing and substitution in categorical terms [1,5]. Since $Cl(\Sigma)$ is biccc, every object $\Gamma = (v_1:\sigma_1,\ldots,v_n:\sigma_n)$ can be regarded as the product type

$$\sigma = \sigma_1 \times \ldots \times \sigma_n$$

and it indexes fibre subcategory $\mathcal{L}(\Sigma,\Pi,\mathcal{A})_\sigma$ of total category over σ containing equational logic over σ.

Fig. 1 Logic over type theory

Objects of this subcategory are contexts

$$\Gamma \mid \Theta,$$

where

$$\Theta = (\varphi_1, \ldots, \varphi_m)$$

is a propositional context consisting of formulae (assumptions) in type context Γ. For every category morphisms $\Gamma \rightarrow \Delta$ in $Cl(\Sigma)$ between type contexts there is a cartesian morphism in total category $\mathcal{L}(\Sigma,\Pi,\mathcal{A})$ that can be regarded as an entailment

$$\Gamma \mid \Theta \mid\!- \psi.$$

In this paper we follow our approach by introducing higher-order types called *polymorphic types*, we construct *polymorphic type theory* in categorical terms and we build logical system as composition of polymorphic fibration.

2. POLYMORPHIC TYPES

In polymorphic types we can use *type variables* $\alpha, \beta, \ldots$ to construct types and form type terms. This leads to new level of indexing that we introduce later by fibration. There are many publications about polymorphic types that deal with polymorphic types

without using category theory, we mention only [3,9,12] in our approach we follow mainly the results of [2,5]. First, we assume a family $\mathcal{T}ype$ consisting of all types.
Polymorphic types can be divided into three versions:

- *first-order polymorphic types* enable type variables α substitutable by specific types σ;

- *second-order polymorphic types* enable abstraction by type variable, second-order products Π and sums Σ, e.g. polymorphic conditional term has the form

$$if: \Pi\alpha : \mathcal{T}ype\,.\,(bool \times \alpha \times \alpha) \to \alpha;$$

- *higher-order polymorphic types* that introduce kinds of types and enables to form higher-order finite products and exponents of kinds.

Type theory for first-order polymorphic types (PTT1) is the most simple polymorphic type theory. We can only use *type variables* and instantiate them by types. For example, an identity function in PTT1 can be defined by

$$\lambda\, id{:}\alpha\,.id : \alpha \to \alpha$$

and instantiated by substituting type variable α by a type σ and get

$$\lambda\, id : \sigma\,.\, id{:}\, \sigma \to \sigma.$$

In type theory for second-order polymorphic types (PTT2) we can form *type terms* with type variables, e.g. by abstraction. For example an identity in PTT2 is the following type term

$$I = \lambda\alpha : \mathcal{T}ype\,.\,\lambda id : \alpha\,.\, id : \Pi\alpha : \mathcal{T}ype\,.(\alpha \to \alpha)$$

and we can instantiate it by substitution and β-reduction as

$$I\,\sigma = \lambda\, id : \sigma\,.\, id{:}\, \sigma \to \sigma\,.$$

Second-order product is impredicative, it involves quantification over all types. Impredictability makes PTT2 very popular but introduces some semantical problems [3].

Type theory for higher order polymorphic types (HPTT) introduces *kinds of types*. In Church's type theory we assume a countable set of *term variables*

$$Var = \{ v_1, v_2, ..., x, y, z, ...\},$$

in PTT1 and PTT2 we need a set of *type variables*

$$Tvar = \{ \alpha_1,\ \alpha_2\,, ..., \beta, ...\}\,.$$

In HPTT we assume kinds of types from the set

$$Kinds = \{ K_1, K_2, ..., L, M, ...\}.$$

So every type variable has associated exactly one kind, e.g. $\alpha : K$. We can see that while PTT1 and PTT2 use only one kind, namely $\mathcal{T}ype$, in HPTT we may have more kinds and form finite products and exponents of them.

3. POLYMORPHIC SIGNATURE

As in the case of Church's type theory we start to build polymorphic type theory (PTT) with defining signature. For PTT we need to introduce more complex structure reflecting kinds of types, so called polymorphic signature.

A *polymorphic signature* $(\Sigma\,,(\Sigma_k)\,)$ consists of

- a higher-order *kind signature*

$$\Sigma = (\mathcal{K},\ \mathcal{F}),$$

where $\mathcal{K}$ is a finite set of kinds and $\mathcal{F}$ is a finite set of function symbols $F: K_1,..., K_n \to L$ between kinds;

- for every finite sequence $k = K_1,...,K_n$ of kinds from $\mathcal{K}$ a *type signature*

$$\Sigma_k = (T,F)$$

containing a finite set T of Σ-terms

$$\alpha_1{:}K_1,\ ...,\ \alpha_n{:}K_n \mid\!- \sigma : \mathcal{T}ype$$

constructed with the kind signature and a finite set F of function symbols.

In PTT1 and PTT2 we have just one kind $\mathcal{T}ype$, therefore kind signature contains only this kind and function symbols in $\mathcal{F}$ can be, for instance

$$List : \mathcal{T}ype \to \mathcal{T}ype$$
$$Stack: \mathcal{T}ype \to \mathcal{T}ype\,.$$

Then in type signatures can be constants and function symbols as

$$empty: \to Stack(\alpha)$$
$$push: \alpha,\ Stack(\alpha) \to Stack(\alpha)$$
$$nil: \to List(\alpha)$$
$$add: \alpha,\ List(\alpha) \to List\,(\alpha)$$

where $\alpha{:}\mathcal{T}ype$ is a type variable.

4. POLYMORPHIC TYPE CALCULUS

Let $(\Sigma, (\Sigma_k))$ be a polymorphic signature defined in the previous section. Let Var and $TVar$ be sets of term and type variables, respectively. We denote by

$$\Phi = (\alpha_1{:}\,K_1,\ ...\ ,\ \alpha_n{:}\,K_n)$$

a kind context over polymorphic signature, where $K_1, \ldots, K_n \in \mathcal{K}$. Let $\Gamma = (v_1{:}\sigma_1, \ldots, v_m{:}\sigma_m)$ be a type context as in Church's type theory. A well-formed term $t{:}\tau$ with free type variables $\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n$ and free term variables from Γ has the form

$$\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n \mid v_1{:}\sigma_1, \ldots, v_m{:}\sigma_m \mid\!- = t{:}\tau . \qquad (1)$$

The vertical bar '|' separates kind context and type context. This sequent determines that σ_i, for $i = 1, \ldots, m$ and τ are types in kind context $\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n$, i.e.

$$\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n \mid\!- \sigma_1 : \mathcal{T}ype .$$

We use for types the notation $\sigma(\alpha_1, \ldots, \alpha_n)$ expressing corresponding kinds in type and $t(\alpha_1, \ldots \alpha_n, v_1, \ldots, v_m)$ to express that a term t has kind and type contexts as in (1). From such notation we see what type and term variables are free in types and terms, respectively.

Now we can define polymorphic type calculi. Firstly, we introduce rules for product types $\sigma \times \tau$, with empty product 1 and for function types $\sigma \to \tau$.

$$\frac{}{\Phi \mid\!- 1 : \mathcal{T}ype} \qquad \frac{\Phi \mid\!- \sigma{:}\mathcal{T}ype \quad \Phi \mid\!- \tau{:}\mathcal{T}ype}{\Phi \mid\!- \sigma \times \tau{:}\mathcal{T}ype}$$

$$\frac{\Phi \mid\!- \sigma{:}\mathcal{T}ype \quad \Phi \mid\!- \tau{:}\mathcal{T}ype}{\Phi \mid\!- \alpha \to \tau : \mathcal{T}ype}$$

We note that these rules are for all PTT1, PTT2 and HPTT calculi because we handle types and we can construct their products and functions in arbitrary kind context. The same situation is in the rules of abstraction and applications:

$$\frac{\Phi \mid \Gamma, x{:}\sigma \mid\!- t{:}\tau}{\Phi \mid \Gamma \mid\!- \lambda x{:}\sigma . t{:}\sigma \to \tau} \qquad \frac{\Phi \mid \Gamma \mid\!- t{:}\sigma \to \tau \quad \Phi \mid \Gamma \mid\!- s{:}\sigma}{\Phi \mid \Gamma \mid\!- t\, s : \tau}$$

These are only terms of PTT1 calculus. In PTT2 we use two second-order constructors: product $\mathbf{\Pi}$ and sum $\mathbf{\Sigma}$ that enable to form second-order product and sum types

$$\mathbf{\Pi}\alpha{:}\mathcal{T}ype . \sigma \quad \text{and} \quad \mathbf{\Sigma}\alpha{:}\mathcal{T}ype . \sigma.$$

Both these constructors bind the type variable α and are formed by the rules

$$\frac{\Phi, \alpha{:}\mathcal{T}ype \mid\!- \sigma{:}\mathcal{T}ype}{\Phi \mid\!- \mathbf{\Pi}\alpha{:}\mathcal{T}ype . \sigma : \mathcal{T}ype} \qquad \frac{\Phi, \alpha{:}\mathcal{T}ype \mid\!- \sigma{:}\mathcal{T}ype}{\Phi \mid\!- \mathbf{\Sigma}\alpha{:}\mathcal{T}ype . \sigma{:}\mathcal{T}ype}$$

To these rules correspond abstraction and application rules that can be found in [7]. Here we present introduction and elimination rules for second-order product. The rules for second-order sums can be formulated in similar way using term constructor *unp* as in Church's type theory. Rules for $\mathbf{\Pi}$ are introduction

$$\frac{\Phi, \alpha{:}\mathcal{T}ype \mid \Gamma \mid\!- t{:}\sigma}{\Phi \mid \Gamma \mid\!- \lambda\alpha{:}\mathcal{T}ype . t{:}\, \mathbf{\Pi}\alpha{:}\, \mathcal{T}ype . \sigma} \qquad (2)$$

where α is not in Γ and elimination

$$\frac{\Phi \mid \Gamma \mid\!- t{:}\Pi\alpha{:}\, \mathcal{T}ype . \sigma \quad \Phi \mid\!- \tau{:}\mathcal{T}ype}{\Phi \mid \Gamma \mid\!- t\, \tau : \sigma[\tau/\alpha]} \qquad (3)$$

Introduction rule allows to abstract over types by polymorphic function $\lambda\alpha : \mathcal{T}ype . t$, elimination rule allows substitution of type variable by a type τ. The associated conversions are

$$(\lambda\alpha{:}\, \mathcal{T}ype . t)\, \tau = t[\tau/\alpha] \quad (\beta\text{- reduction})$$
$$\lambda\alpha{:}\mathcal{T}ype . t\, \alpha = t \quad (\eta\text{- reduction})$$

In HPTT we have more atomic kinds $K \in \mathcal{K}$ from kind signature. HPTT has rules for forming finite kind products, i.e. $K \times L \in \mathcal{K}$ with empty product kind $\mathbf{1} \in \mathcal{K}$ and exponents of kinds $K \to L \in \mathcal{K}$. The rules are similar as for product and function types in Church's type theory where we write kinds K,L instead of types σ,τ. Higher-order products $\mathbf{\Pi}\alpha{:}K.\sigma$ can be constructed over all kinds from $\mathcal{K}$, not only for the unique kind $\mathcal{T}ype$ as in rule (2) and (3) for PTT2.

We can extend these polymorphic type calculi with *equality types* denoted by $eq_K(\sigma,\tau){:}\ \mathcal{T}ype$. The equality types have real sense only in HPTT.

5. SET THEORETIC SEMANTICS OF PTT

In PTT we have type variables that introduce new level of indexing. Firstly, we define set-theoretic semantics of PTT and discuss the problems appearing in it for PTT2 and HPTT. Let $\mathcal{U}$ be some set of sets consisting of interpretations of kinds of kind signature Σ. Then representation $[\![K]\!]$ of a kind $K \in \mathcal{K}$ is an element of $\mathcal{U}$, $[\![K]\!] \in \mathcal{U}$.

We define a model of polymorphic signature $(\Sigma, (\Sigma_k))$ as follows. We interpret

- every kind K as an element $[\![K]\!] \in \mathcal{K}$ as above;

- every type in a sequent $\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n \mid\!- \sigma{:}\mathcal{T}ype$ we interpret as a function

$$[\![\sigma]\!] : [\![K_1]\!] \times \ldots \times [\![K_n]\!] \to \mathcal{U}$$

• if we denote by $l \in [\![K_1]\!] \times \ldots \times [\![K_n]\!]$ a product of kind representations, we can write

$$[\![\sigma]\!](l) \in \mathcal{U}$$

for the representation of type σ in kind context l as an element of $\mathcal{U}$, i.e. a set.

• for every sequence $k = K_1, \ldots, K_n$ of kinds we interpret a function symbol $f: \sigma_1 \ldots \sigma_n \to \tau$ in Σ_k as a family of functions

$$[\![f]\!](l): [\![\sigma_1]\!](l) \times \ldots \times [\![\sigma_m]\!](l) \to [\![\tau]\!](l).$$

In other words f is interpreted as an element of product

$$[\![f]\!] \in \Pi_l ([\![\sigma_1]\!](l) \times \ldots \times [\![\sigma_m]\!](l) \to [\![\tau]\!](l)).$$

Let assume that $\mathcal{U}$ is closed under finite products and exponents. If it does not lead to confusion, we omit kind and type contexts to simplify a bit our notation. We denote by $[\![\sigma]\!]$ an interpretation of any type $\Phi \vdash \sigma: \mathcal{T}ype$, by $[\![t]\!]$ an interpretation of any term $\Phi \mid \Gamma \vdash t: \tau$ and by K the product $K_1 \times \ldots \times K_n$.

We extend defined interpretation to exponent types $\sigma \to \tau$. If

$$[\![\sigma]\!]: [\![K]\!] \to \mathcal{U} \quad \text{and} \quad [\![\tau]\!]: [\![K]\!] \to \mathcal{U}$$

then interpretation of exponent types is defined by pointwise function space:

$$[\![\sigma \to \tau]\!] =_{def} \lambda l. [\![\tau]\!](l)^{[\![\sigma]\!](l)}: [\![K]\!] \to \mathcal{U}.$$

Let $\alpha: K \mid x: \sigma, y: \rho \vdash t: \tau$ be a term interpreted as

$$[\![t]\!] \in \Pi_l ([\![\sigma]\!](l) \times [\![\rho]\!](l) \to [\![\tau]\!](l)).$$

Then abstraction is interpreted as

$$[\![\lambda y: \rho. t]\!] =_{def} \lambda l. \lambda x. \lambda y. [\![t]\!](l)(x,y) \in \Pi_l ([\![\sigma]\!](l) \to [\![\rho \to \tau]\!](l)).$$

For application let

$$\alpha: K \mid x: \rho \vdash t: \sigma \to \tau \quad \text{and} \quad \alpha: K \mid x: \rho \vdash s: \sigma$$

be terms interpreted as functions

$$[\![t]\!] \in \Pi_l ([\![\rho]\!](l)) \to [\![\sigma \to \tau]\!](l), \text{ and}$$
$$[\![s]\!] \in \Pi_l ([\![\rho]\!](l) \to [\![\sigma]\!](l)).$$

Then application is interpreted as

$$[\![t\ s]\!] =_{def} \lambda l. \lambda x. [\![t]\!](l)(x)([\![s]\!](l)(x) \in \Pi_l ([\![\rho]\!](l) \to [\![\tau]\!](l)).$$

This set-theoretic approach works well for PTT1. But for PTT2 and HPTT we need to interpret second and higher-order products. In [13] is proved that such set-theoretic interpretation is not possible because $\mathcal{U}$ is not closed under second and higher-order products over itself. However there are models containing 'sets' suitably closed under exponents and products to allow interpretation of PTT2. One of them is internal category of partial equivalence relations in the category of effective toposes [5]. In [10] is argued that the impossibility of model of HPTT in ***Set*** category of sets and functions is because of considering classical logic. In [11] is shown that it is not the classical nature of logic that cause problems but rather the nature of type $\mathcal{P}rop$ of propositions.

6. FIBRATIONS

A *fibration* $p: \boldsymbol{E} \to \boldsymbol{B}$ from total category $\boldsymbol{E}$ to base category $\boldsymbol{B}$ is a special functor p such that for any object I in $\boldsymbol{B}$ there is a subcategory, a *fibre* $\boldsymbol{E}_I$ of $\boldsymbol{E}$, where

$$p(\boldsymbol{E}_I) = I$$

and for any morphism $u: I \to J$ there is a *cartesian morphism* $X \to Y$ in $\boldsymbol{E}$ such that

$$p(X) = I, \qquad p(Y) = J$$

and it has universal property [5]. Every object I in base category indexes fibre subcategory $\boldsymbol{E}_I$.

A morphism $X \to Y$ in $\boldsymbol{E}$ is *cartesian lifting* over $u: I \to J$ if for any Y,such that $p(Y) = J$, there is unique object X in $\boldsymbol{E}$, such that $X \to Y$ is cartesian. A fibration is *split* if it comes together with a choice of cartesian lifting and substitution functor $u^*: Y \to X$.

We define polymorphic fibration suitable for *all* kind of PTT.We assume $\mathcal{P}rop: \mathcal{T}ype$, a special atomic type, such that predicates on type σ correspond to characteristic terms $\sigma \to \mathcal{P}rop$. Categorically we describe this correspondence by generic object.

Let $\boldsymbol{E} \to \boldsymbol{B}$ be a split fibration. A *generic object* Ω is an object in the category $\boldsymbol{B}$ together with a family of isomorphisms

$$\phi_I: Hom(I, \Omega) \to \mathrm{Obj}(\boldsymbol{E}_I)$$

i.e. the set $Hom(I,\Omega)$ of all morphisms from an object I in $\boldsymbol{B}$ to Ω is isomorphic to the set of objects of fibre subcategory $\boldsymbol{E}_I$ over I, with the property

$$\phi_J (u \circ v) = v^* (\phi (u))$$

where $u: I \to \Omega$, $v: J \to I$ for any object J in $\boldsymbol{B}$ and $v^*: \boldsymbol{E}_I \to \boldsymbol{E}_J$ is substitution functor between corresponding fibre categories in $\boldsymbol{E}$ induced by v.

6.1. PTT as fibration

In PTT we have two indexing: by type variables and by term variables. Therefore it is appropriate to consider fibration as a suitable concept for expressing the nature of PTT. We construct PTT as *polymorphic fibration*, i.e. a fibration with generic object, fibred finite products in total category and finite products in base category.

We note here that fibration enables us to build also logic over PTT similarly as in the case of Church's type theory by composed fibration. Objects of base categories should be type contexts Γ and kind contexts Φ. To construct logic we introduce propositional context $\Theta = (\varphi_1, ..., \varphi_p)$, over Φ and Γ, of propositions (assumptions) [7]. We construct logic over PTT in the next section. Now we need to construct PTT as split fibration.

In the following let $(\Sigma, (\Sigma_k))$ be a polymorphic signature for solved problem. We construct *classifying category* $Cl(\Sigma)$ for kind signature as follows:

- objects are kinds contexts $\Phi = (\alpha_1:K_1, ..., \alpha_n:K_n)$;

- morphisms $\Phi \to \Phi'$ are sequences of terms $(t_1, ..., t_n)$ with $\Phi' \mid\!- t_i:K_i$, $i=1, ..., n$.

The special kind $\mathcal{T}ype$ is an object in $Cl(\Sigma)$. Finite products in this category are given by concatenation of kind contexts.

We construct split indexed category $Cl(\Sigma, (\Sigma_k))$ over $Cl(\Sigma)$ by the functor

$$q: Cl(\Sigma)^{op} \to \mathbf{Cat}$$

from the dual (opposite) category of classifying category $Cl(\Sigma)$ to the category ***Cat*** of small categories which assigns:

- to every object (kind context) Φ in $Cl(\Sigma)$ a category of types $\Phi \mid\!- \sigma:\mathcal{T}ype$ and of morphisms $\sigma \to \tau$, i.e. terms s,

$$\Phi \mid x:\Sigma \mid\!- s(x):\tau;$$

- to every morphism $\Phi \to \Phi'$ in $Cl(\Sigma)$ it assigns morphisms between corresponding subcategories in ***Cat*** by substitution. If

$$\Phi = (\alpha_1:K_1, ..., \alpha_n:K_n)$$
$$\Phi' = (\beta_1:L_1, ..., \beta_n:L_n)$$

are kind contexts and

$$\Phi \mid\!- \sigma_1:L_1, ..., \Phi \mid\!- \sigma_m:L_m$$

are terms then we transfer terms and types in Φ' by substituting types $\sigma_1, ..., \sigma_m$ for $\beta_1, ..., \beta_m$ by

$$\tau (\beta_1, ..., \beta_m) \mapsto \tau [\sigma_1/\beta_1, ..., \sigma_m/\beta_m]$$

and

$$t (\beta_1, ..., \beta_m, x) \mapsto t [\sigma_1/\beta_1, ..., \sigma_m/\beta_m, x].$$

We note that because domain of q is dual category, the direction of morphisms is reversed.

Then by Grothendieck construction [4] we get a split polymorphic fibration in Fig.2

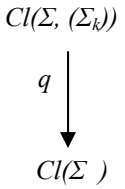

Fig. 2 Polymorphic fibration of PTT

This fibration has generic object $\mathcal{T}ype$ in $Cl(\Sigma)$ because objects over Φ are morphisms $\Phi \to \mathcal{T}ype$ in base category. Because $Cl(\Sigma)$ has finite products, this fibration has fibred finite products. Therefore we have split polymorphic fibration for all PTT.

A *model* of PTT is *functor of fibrations (H,M)* to a split polymorphic fibration $\mathbf{E} \to \mathbf{B}$, where

$$M: Cl(\Sigma) \to \mathbf{B}$$

is a functor from kind classifying category to base category ***B*** and

$$H: Cl(\Sigma,(\Sigma_k)) \to \mathbf{E}$$

is a functor from total category of PTT to total category ***E,*** such that the diagram in Fig.3 commutes. This fibration functor preserves the structure of polymorphic fibration. This fibration morphism can be a model of first-, second- and higher-order polymorphic type theory by putting some further structure on it. In PTT1 there are exponent types $\sigma \to \tau$ modelled as fibred exponents. In the PTT2 are polymorphic products and sums. These are modelled categorically by quantification along projection

$$\pi: I \times \Omega \to I,$$

where Ω is interpretation of $\mathcal{T}ype$.

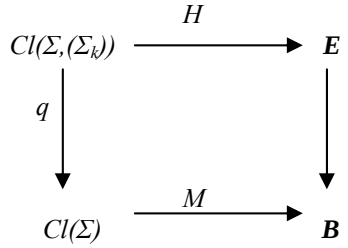

Fig. 3 Model of PTT

7. LOGIC OVER PTT

We constructed logic over Church's type theory as a preorder fibration over classifying category [8]. Here we construct logic over PTT as polymorphic fibration. Let in the following

$$q: Cl(\Sigma, (\Sigma_k)) \rightarrow Cl(\Sigma)$$

be a polymorphic fibration of PTT defined in the previous section. The objects of the base category $Cl(\Sigma)$ are kinds and the objects of the fibres in total category $Cl(\Sigma,(\Sigma_k))_\Phi$ are types $\sigma:\mathcal{T}ype$ over a kind Φ

$$\alpha:K \mid\!- \sigma(\alpha): \mathcal{T}ype\,.$$

We assume a new syntactic notion of type $\mathcal{P}rop$ of propositions such that a formula over kind context K is of type $\mathcal{P}rop$

$$\alpha:K \mid\!- \varphi(\alpha): \mathcal{T}ype\,.$$

Such propositions are the objects in the total category $\mathcal{L}((\Sigma,(\Sigma_k)),\mathcal{A})$ of polymorphic fibration over polymorphic type theory, where $\mathcal{A}$ is a corresponding set of axioms. Therefore we can construct logic over PTT as composed split polymorphic fibration in Fig.4.

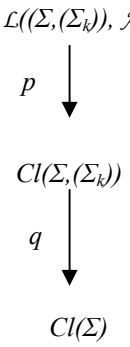

Fig. 4 Logic over PTT

In this figure we can see double indexing by kinds and by types. The top fibration p from logic to PTT is of *propositions-over-types* and the bottom one q is *types-over kinds*. The total category of this composed polymorphic fibration has

- as objects propositions of the form

$$\alpha:K \mid x:\sigma(\alpha) \mid\!- \varphi(\alpha, x): \mathcal{P}rop$$

- as morphisms $\varphi \rightarrow \psi$ entailments of the form

$$\alpha:K \mid x:\sigma(\alpha) \mid \varphi(\alpha, x) \mid\!- \psi(\alpha,x)$$

over σ.

In the entailment we use three contexts,

- a *kind context* $\Phi = (\alpha:K)$, an object of $Cl(\Sigma)$ that serves as the first index for types in fibre category $Cl(\Sigma,(\Sigma_k))_\Phi$;

- a *type context* $\Gamma = (x:\sigma(\alpha))$, an object of fibre subcategory $Cl(\Sigma,(\Sigma_k))_\Phi$ over kind context Φ that serves as the second index for propositions in fibre category $\mathcal{L}((\Sigma,(\Sigma_k)), \mathcal{A})_{\Gamma;\Phi}$;
- a *proposition context* $\Theta = (\varphi_1, ..., \varphi_p)$ that contains assumptions in entailment

$$\Phi \mid \Gamma \mid \Theta \mid\!- \psi.$$

In similar manner as for Church's types we can introduce also logical connectives and quantifiers.

8. CONCLUSION

Inour approach we consider programming as logical reasoning in logical system over type theory. Fibrations enable precise means how to construct entailments of logical system over types. Because polymorphic types are very useful in programming our aim was extend our approach for polymorphic types and construct in exact way logical system over it. We follow our research with considering about another interesting area, dependent types and we investigate how to embedd them into our approach.

This work was supported by VEGA Grant No.1/2181/05: Mathematical Theory of Programming and Its Application in the Methods of Stochastic Programming

REFERENCES

[1] S.Awodey, A.Bauer: Introduction to categorical logic, draft, Carnegie-Mellon University, 2004

[2] H.P.Barendregt: Lambda calculi with types, In: S.Abramsky, D.M.Gabbai, T.S.E.Maibaum (eds.): Handbook of Logic in Computer Science, Vol.2, Oxford Univ.Press, 1992,pp.117-309

[3] Z.Csörnyei: Lambda calculus, Typotex, Budapest, 2007

[4] A Grothendieck: Categories fibrees et descente, In: A.Grothendieck(ed.): Revetement Etales et Groupe Fondamental, Springer, Berlin, 1970, pp.145-194

[5] B.Jacobs: Categorical logic and type theory, Elsevier, Amsterdam, 1999

[6] V.Novitzká: Logical reasoning about programming of mathematical machines, Acta Electrotechnica et Informatica, 5,3,2005, Košice, pp.50-55

[7] V.Novitzká: Church's types in logical reasoning on programming, Acta Electrotechnica et Informatica 6,2, Košice, 2006, pp.27-31

[8] V.Novitzká, D.Mihályi, V.Slodičák: Categorical models of logical systems in the mathematical theory of programming, 6th Joint Conference on Mathematics and Computer Science, July 12-15, 2006, Pécs, Hungary
[9] B.C.Pierce: Types and programming languages, MIT Press, Cambridge, 2002
[10] A.M.Pitts: Polymorphism is set-theoretic constructively, In: D.H.Pitt, A.Poigne, D.E.Rydeheard (eds.): Category and Computer Science, LNCS 283, Springer, Berlin, 1987, pp.12-39.
[11] A.M.Pitts: Non-trivial power types can't be subtypes of polymorphic types, Logic in Computer Science, IEEE, Computer Science Press, 1989, pp.6-13
[12] I. Zólyomi, Z. Porkoláb, T. Kozsik: An extension to the subtype relationship in C++ implemented with template metaprogramming, generative programming and component engineering, , LNCS 2830, 2003, pp. 209-227.
[13] J.C.Reynolds: Polymorphism is not set-theoretic, In: G.Kajn, D.B.Mac Queen, G.D.Plotkin(eds.): Semantics of Data Types, LNCS 173, Springer, Berlin, 1981, pp.145-156
[14] L.Vokorokos, A.Kleinová, O.Látka: Network Security on the Intrusion Detection System Level, Proceedings of IEEE 10th International Conference on Intelligent Engineering Systems, London, Jun 26th – 28th, 2006, pp. 270-275, ISBN 1-4244-9708-8.

BIOGRAPHIES

Valerie Novitzká defended her PhD Thesis: On semantics of specification languages at Hungarian Academy of Sciences in 1989. She works at Department of Computers and Informatics from 1998, firstly as Assistent Professor, from 2004 as Associate Professor. Her research areas covers category theory, categorical logic, type theory, classical and linear logic and theoretical foundations of program development.

Daniel Mihályi works as a researcher at the Department of Computers and Informatics since 1989. Now he works on his PhD. Thesis. His main areas of research activities and interests are categorical logic, linear logic and logical reasoning in applied mathematics and programming. He is interested also in analysis and methodology design of program systems security in Unix operating systems environment and internet services technologies.