JAM CM Journal of Applied Mathematics and Computational Mechanics 2015, 14(1), 27-42
www.amcm.pcz.pl p-ISSN 2299-9965
DOI: 10.17512/jamcm.2015.1.03 e-ISSN 2353-0588

A CATEGORICAL MODEL OF PREDICATE LINEAR LOGIC

Emília Demeterová, Daniel Mihályi, Valerie Novitzká

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice
Košice, Slovak Republic
emilia.demeterova@tuke.sk, daniel.mihalyi@tuke.sk, valerie.novitzka@tuke.sk

Abstract. Linear logic is one of the logical systems with special properties suitable for describing real processes used in computer science. It enables one to specify dynamics, non determinism, consecutive processes and important resources as memory and time on syntactic level. Moreover, its deduction system enables one to verify specified properties. Constructing an appropriate model based on categories can serve for modeling various program systems in the wide spectrum of computer science. Mainly, propositional linear logic is used for these purposes. The expression power of linear logic significantly grows by extending propositional logic with predicates and quantifiers. Our paper concerns itself with defining predicate linear logic together with its deduction system and our main aim is to construct a categorical model of predicate linear logic as a symmetric monoidal closed category.

Keywords: *linear type theory, predicate linear logic, symmetric monoidal closed category*

Introduction

Linear logic was introduced by Jean-Yves Girard in 1987 [1]. Linear logic is the only logic from the existing logical systems which is able to describe processes as they behave in real world. It can describe dynamics of processes, external or internal non determinism, consecutive processes manage with resources on a syntactic level. Linear logic can be considered as a bridge between computer science and logic.

Propositional linear logic is often used for describing program systems [2, 3], their behavior [4] and its extension with modal operators enables the modeling of knowledge achievement [5].

In spite of these useful properties of propositional linear logic, its expressive power is insufficient for describing properties of some objects and relations between them. For these purposes are appropriate predicates together with quantifiers as it provides predicate logic. Therefore in this paper we formulate predicate linear logic, its syntax, deduction system and categorical model. Calculations can be expressed by linear terms. Properties and relations of calculations can be

expressed by predicates. Quantifiers are able to specify a group of objects for which some feature or relation is valid. In the future we plan to use predicate linear logic for specifying component based systems [6], especially for describing interactions and dependencies between components. In the next section we shortly introduce propositional linear logic, its syntax and deduction calculus. In section 2 we define the linear type theory and its model as a symmetric monoidal closed category. Section 3 contains syntax and deduction calculus of a multiplicative fragment of predicate linear logic and in section 4 we construct a categorical model of predicate linear logic.

1. Propositional linear logic

Linear logic has much greater expressive power than classical logic thanks to more connectives with special properties. Besides, every formula of classical propositional logic can be expressed in linear logic, too.

One of the most important properties of linear logic is its ability to describe dynamics of processes. By linear implication $\multimap$ it is able to express changes, sequentiality and causality of processes.

Another important property of linear logic is its ability to handle resources - with logical space and logical time [3]. It allows one to express the internal structure of the resources, their consumption together with a continuance of processes by the incrementation of time. Resources used in linear logic, logical space and logical time are the most important resources used in computer science, too. These resources are used at calculating a running program.

In same cases it is advisable to use only some fragments [7] of linear logic, e.g. multiplicative, additive. An interesting fragment is intuitionistic linear logic which satisfies Curry-Howard correspondence [8, 9] with the computing system such as λ - calculus.

Linear logic defines new logical connectives [10]. Depending on the fragments of linear logic there exist multiplicative conjunction $\otimes$, multiplicative disjunction $\wp$, additive conjunction &, additive disjunction $\oplus$, linear implication $\multimap$. Inexhaustibility of resources can be described by modal operators ! *of course* and ? *why not*.

1.1. Syntax of propositional linear logic

In this section we introduce the syntax of propositional linear logic. Let

$$Prop = \{p_1, p_2, \dots, p_n\}$$

be a countable set of elementary sentences denoted by $p_1, p_2, \dots$. Every formula can be understood either as a resource or as an action. A linear formula φ has one of the following forms defined by the BNF rule:

$$\varphi ::= p \mid 1 \mid 0 \mid \top \mid \bot \mid \varphi \otimes \varphi \mid \varphi \oplus \varphi \mid \varphi \& \varphi \mid \varphi \parr \varphi \mid \varphi \multimap \varphi \mid \varphi^{\perp} \mid !\varphi \mid ?\varphi$$

- Formula $\varphi \otimes \psi$ expresses a *multiplicative conjunction* and it has the neutral element *1*. This formula expresses that both actions are performed simultaneously or both resources are available at once.
- Formula $\varphi \ \& \ \psi$ expresses an *additive conjunction* and it has the neutral element $\top$. This formula expresses that only one of the actions is performed but we can anticipate or deduce which one from the environment. We can call this external non determinism. In the case of resources only one of them is available.
- Formula $\varphi \oplus \psi$ expresses *additive disjunction* and it has the neutral element 0. This formula expresses that only one of the actions is performed but we cannot anticipate which one. We can call this internal non determinism.
- Formula $\varphi \parr \psi$ expresses *multiplicative disjunction* and it has the neutral element $\bot$. This formula expresses that if the first action is not performed then the second one is performed or vice versa.
- Expression $\varphi^{\perp}$ is called *linear negation* and it denotes a reaction of an action φ or a consumption of a resource φ. Linear negation is involutive

$$\varphi^{\perp\perp} \equiv \varphi.$$

- Formula $!\varphi$ is a modal formula with the modal operator *of course*. It expresses the potential inexhaustible resource φ.
- Formula $?\varphi$ uses the modal operator *why not* and it expresses the actuality of potential resource inexhaustibility. Modal operators *of course* and *why not* are dual

$$(!\varphi)^{\perp} \equiv ?(\varphi^{\perp}).$$

- Formula $\varphi \multimap \psi$ expresses *linear implication*. This formula expresses that the first action is a cause of the second action or in the case of resources, it expresses that the first resource is consumed after linear implication.
  Linear implication expressed by the modal operator *of course !*

$$!\varphi \multimap \varphi$$

  is an analogy to classical implication $\varphi \Rightarrow \varphi$.

1.2. Deduction calculus for linear logic

We describe the deduction system of linear logic by sequent calculus defined by Gentzen. A sequent has the form

$$\varphi_1, \ldots, \varphi_n \vdash \psi_1, \ldots, \psi_m,$$

where formulae $\varphi_1, \ldots, \varphi_n$ are assumptions implying at least one of the formulae $\psi_1, \ldots, \psi_m$.
The deduction system of linear logic consists of the rules for the connectives, constants and modal operators of linear logic.

The deduction rules have the form

$$\frac{S_1, \ldots, S_n}{S}$$

where the sequence of sequents $S_1,\ldots,S_n$ contains conditions that have to be valid in order to deduce the conclusion S.

The deduction system of propositional linear logic has the following deduction rules:

$$\frac{}{\varphi \vdash \varphi}\,(\mathbf{id}) \quad (1) \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma', \varphi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\mathbf{cut}) \quad (2)$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \otimes \psi \vdash \Delta}\,(\otimes_{\mathbf{L}}) \quad (3) \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma' \vdash \psi, \Delta'}{\Gamma, \Gamma' \vdash \varphi \otimes \psi, \Delta, \Delta'}\,(\otimes_{\mathbf{R}}) \quad (4)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta}\,(\top_{\mathbf{L}}) \quad (5) \qquad \frac{}{\vdash \top}\,(\top_{\mathbf{R}}) \quad (6)$$

$$\frac{\Gamma, \varphi \vdash \Delta \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \varphi \wp \psi \vdash \Delta, \Delta'}\,(\wp_{\mathbf{L}}) \quad (7) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \wp \psi, \Delta}\,(\wp_{\mathbf{R}}) \quad (8)$$

$$\frac{}{\bot \vdash}\,(\bot_{\mathbf{L}}) \quad (9) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\,(\bot_{\mathbf{R}}) \quad (10)$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \varphi \multimap \psi \vdash \Delta, \Delta'}\,(\multimap_{\mathbf{L}}) \quad (11) \qquad \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \multimap \psi. \Delta}\,(\multimap_{\mathbf{R}}) \quad (12)$$

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\,(\mathbf{promotion_L}) \quad (13) \qquad \frac{!\Gamma \vdash \varphi, ?\Delta}{!\Gamma \vdash !\varphi, ?\Delta}\,(\mathbf{promotion_R}) \quad (14)$$

$$\frac{!\Gamma, \varphi \vdash ?\Delta}{!\Gamma, ?\ \varphi \vdash ?\Delta}\,(\mathbf{dereliction_L}) \quad (15) \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash ?\varphi, \Delta}\,(\mathbf{dereliction_R}) \quad (16)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\,(\mathbf{w_L}) \quad (17) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?\varphi, \Delta}\,(\mathbf{w_R}) \quad (18)$$

$$\frac{\Gamma, !\varphi, !\varphi \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\,(\mathbf{c_L}) \quad (19) \qquad \frac{\Gamma \vdash ?\varphi, ?\varphi, \Delta}{\Gamma \vdash ?\varphi, \Delta}\,(\mathbf{c_R}) \quad (20)$$

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \varphi \& \psi \vdash \Delta}\,(\&_{\mathbf{L1}}) \quad (21) \qquad \frac{\Gamma, \psi \vdash \Delta}{\Gamma, \varphi \& \psi \vdash \Delta}\,(\&_{\mathbf{L2}}) \quad (22)$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \& \psi, \Delta}\,(\&_{\mathbf{R}}) \quad (23) \qquad \frac{}{\Gamma \vdash \mathbf{1}, \Delta}\,(\mathbf{1_R}) \quad (24)$$

$$\frac{\Gamma,\varphi\Delta \quad \Gamma,\psi\vdash\Delta}{\Gamma,\varphi\oplus\psi\vdash\Delta}(\oplus_{\mathbf{L}}) \qquad (25) \qquad \frac{\Gamma\vdash\varphi,\Delta}{\Gamma\vdash\varphi\oplus\psi,\Delta}(\oplus_{\mathbf{R1}}) \qquad (26)$$

$$\frac{\Gamma\vdash\psi,\Delta}{\Gamma\vdash\varphi\oplus\psi,\Delta}(\oplus_{\mathbf{R2}}) \qquad (27) \qquad \frac{}{\Gamma,\mathbf{0}\vdash\Delta}(\mathbf{0}_{\mathbf{L}}) \qquad (28)$$

$$\begin{aligned}\boldsymbol{\varphi} \multimap \boldsymbol{\psi} \quad &\equiv \quad \boldsymbol{\varphi}^{\perp} \wp\, \boldsymbol{\psi} \\ \boldsymbol{\varphi} \multimap \boldsymbol{\psi} \quad &\equiv \quad (\boldsymbol{\varphi} \otimes\, \boldsymbol{\psi}^{\perp})^{\perp}\end{aligned} \qquad (29)$$

Many approaches exist to express the semantics of predicate linear logic. Primarily the semantics of linear logic was expressed by coherent spaces [1-10] or by quantales [11]. Phase semantics [13] does not say anything about connectives and expresses only the truth of the statements. Heyting's semantics [12] is not interested in the truth of the expressions, but their sense. This semantics is important only if there exists a provable formula.

In constructing a model of predicate linear logic, we follow the idea that every elementary linear formula can be represented as a type. First, we define linear type theory and then we construct a categorical model of predicate linear logic.

2. Linear type theory

To introduce predicates into linear logic, we have to define types and linear terms. Every programming language includes some predefined basic types. Let $\boldsymbol{B} = \{X, Y, Z\}$ be a set of basic types and I the unit type. The syntax of linear types [14] are defined by the following BNF grammar:

$$\sigma ::= X \mid I \mid \sigma\otimes\sigma \mid \sigma\multimap\sigma$$

where

- X is a basic type;
- I is the unit type;
- $\sigma\otimes\sigma$ is a linear product type;
- $\sigma\multimap\sigma$ is a linear function type.

2.1. Linear terms

Linear terms are expressed by operations and variables. First, we define linear preterms.

Let $Var(\sigma)$ be a set of variables of type σ. Let $Preterm\ \sigma$ be a set of preterms types of σ constructed as

- $() \in Preterm\ (I)$ is an empty linear preterm;
- $x \in Preterm\ (\sigma)$ is a linear preterm of types σ, if $x \in var(\sigma)$ is a variable of type σ;

- $(s,t) \in Preterm(\sigma \otimes \tau)$ is a linear preterm of product type $\sigma \otimes \tau$, if $s \in Preterm(\sigma)$ and $t \in Preterm(\tau)$ are linear preterms;
- $\alpha(s) \in Preterm(\tau)$ is a linear preterm of type τ if $s \in Preterm(\sigma)$ is a linear preterm and $\alpha: \sigma \multimap \tau$ is a function.

Linear terms have the following syntax:

$$t ::= x \mid f(t,...,t)$$

i.e. a term is either a variable or application of function. Every term has associated a unique type and we denote typed terms in sequent form as

$$\Gamma \vdash t: \sigma$$

where Γ is a finite sequence of typed variables $x_1:\sigma_1,...,x_n:\sigma_n$.

Linear term s of type σ is a preterm $s \in Preterm(\sigma)$ in which every variable occurs only once. A linear combinator is a closed linear term, i.e. all variables occuring in the term are bounded. We introduce several linear combinators expressing special properties needed for modeling the linear type theory:

- linear combinator Id_σ expresses identity on the type σ

$$Id_\sigma : \sigma \to \sigma;$$

- linear combinator *assl* expresses left associativity on types σ, τ, χ

$$assl_{\sigma,\tau,\chi} : \sigma \otimes (\tau \otimes \chi) \to (\sigma \otimes \tau) \otimes \chi;$$

- linear combinator *assr* expresses right associativity on types σ, τ, χ

$$assr_{\sigma,\tau,\chi} : (\sigma \otimes \tau) \otimes \chi \to \sigma \otimes (\tau \otimes \chi);$$

- linear combinator *swap* expresses commutativity on type

$$swap_{\sigma,\tau} : \sigma \otimes \tau \to \tau \otimes \sigma;$$

- linear combinator *open* expresses left neutral element

$$open_\sigma : \sigma \to 1 \otimes \sigma;$$

- linear combinator *close* expresses right neutral element

$$close_\sigma : 1 \otimes \sigma \to \sigma;$$

- linear combinator *eval* expresses evaluation over types

$$eval_{\sigma \multimap \tau} : (\sigma \multimap \tau) \otimes \sigma \to \tau.$$

We can construct generalized combinators with the rules of composition, product and abstraction [9] from linear combinators and function symbols where α, β, γ denote either function or combinator:

$$\frac{\alpha: \sigma \longrightarrow \tau \quad \beta: \tau \longrightarrow \theta}{\beta \circ \alpha: \sigma \longrightarrow \theta}(composition)$$

Linear combinator *composition* expresses composition of functions α and β.

$$\frac{\alpha: \sigma \longrightarrow \tau \quad \beta: \theta \longrightarrow \chi}{\alpha \otimes \beta: \sigma \otimes \theta \longrightarrow \tau \otimes \chi}(product)$$

Linear combinator *product* expresses tensor product between α and β, and the result is a tensor product of corresponding types.

$$\frac{\alpha: \sigma \otimes \tau \longrightarrow \theta}{\Lambda(\alpha): \sigma \longrightarrow (\tau \multimap \chi)}(abstraction)$$

Linear combinator *abstraction* expresses a map from the argument σ on the function type $\tau \multimap \theta$.

In order to define equivalence between terms, we introduce relation of equivalence of terms $s \equiv_\sigma t$. Equivalence means that terms will have the same value after the evaluation, so for any functions $f, g \in term(\sigma \multimap \tau)$ and basic term $x{:}\sigma$ hold

$$\text{if } eval_{\sigma,\tau}(f,x) \equiv_\tau eval_{\sigma,\tau}(g,x), \text{ then } f \equiv_\sigma g.$$

Properties of defined linear combinators are defined by the following axioms:

$$\begin{array}{ccc} Id_\sigma\ (s) & \equiv & s \\ (\gamma \circ\ \alpha)(s) & \equiv & \gamma(\alpha(s)) \\ assl_{\sigma,\tau,\theta}\ (s,(t,u)) & \equiv & (s,t),u) \\ assr_{\sigma,\tau,\theta}\ ((s,t,)u) & \equiv & (s,(t,u)) \\ swap_{\sigma,\tau}\ (s,t) & \equiv & (t,u) \\ open_\sigma\ (s) & \equiv & ((),s) \\ close_\sigma\ ((),s) & \equiv & s \\ eval_{\sigma,\tau}\ (\Lambda(\alpha)(\ s),t) & \equiv & \alpha(s,t) \end{array}$$

where *s, t, u* are terms with same types and variables, α, β, γ are combinators.

Free variables can be substituted by terms of same type in terms. Evaluation of terms are gained by their substitution. If s is a linear term of type σ, $s \in term(\sigma)$, and x is a variable of type σ, $x \in var(\sigma)$, then $t[s/x]$ expresses linear term t, where every free occurrence of variable x is substituted with a linear term s. Substitution is defined as following:

$$
\begin{aligned}
()[s/x] = (); \quad &= \quad (); \\
y[s/x] \quad &= \quad \begin{cases} s \ \ iff \ x = y; \\ y \ \ iff \ x \neq y; \end{cases} \\
(t,u)[s/x] \quad &= \quad (t[s/x], u[s/x]); \\
\alpha(t)[s/x] \quad &= \quad \alpha(t[s/x]);
\end{aligned}
$$

where t and u are linear terms of optional type, α is functional symbol or combinator between linear types.

We call the expression $t[s/x]$ *linear term*, if terms t and s do not have the same variables.

2.2. Categorical model of linear type theory

As a basis for a categorical model of linear type theory we use symmetric monoidal closed category [15, 16]. The reason for this choice was formulated by the following facts:

- a type can be represented as an object in Cartesian closed category [2]. Symmetric monoidal closed category is a generalization of it.
- For arbitrary monoidal closed category it exists a linear type theory [14, 17], whose model is this category.

Terms can be represented by the categorical morphisms [18].

Symmetric monoidal closed category [17, 19] is defined by a sixtuple

$$(\mathfrak{C}, \otimes, I, a, l, c, Hom(-,-)),$$

where

- $\mathfrak{C}$ is a category;
- $\otimes : \mathfrak{C} \ x \ \mathfrak{C} \rightarrow \mathfrak{C}$ is a tensor product;
- I is the neutral element of the tensor product, it is an object in $\mathfrak{C}$. This object serves as terminal object of the category $\mathfrak{C}$;
- $a_{X,Y,Z}$:$(X \otimes Y) \otimes Z \rightarrow X \otimes (Y \otimes Z)$ is a natural isomorphism which expresses left associativity of tensor product, X, Y, Z are objects in $\mathfrak{C}$;
- l: $I \otimes X \rightarrow X$ is a natural isomorphism expressing left neutral element of the tensor product;
- $c_{X,Y}$: $X \otimes Y \rightarrow Y \otimes X$ is a natural isomorphism expressing commutativity of the tensor product;
- for every object X in $\mathfrak{C}$ the functor $- \otimes X$ has a right adjoint Hom-functor $Hom(X,-)$ with natural transformations

$$\varepsilon_{X,Y} : Hom(X,Y) \otimes X \rightarrow Y$$
$$\delta_{X,Y} : X \rightarrow Hom(Y, X \otimes Y).$$

Now we construct symmetric monoidal closed category $\mathfrak{C}$ as a model of linear type theory. Category objects of the $\mathfrak{C}$ are type contexts Γ, Δ,..., which can be realized as finite products of types

$$\sigma_1 \otimes \ \ldots \ \otimes \ \sigma_n.$$

Morphisms in $\mathfrak{C}$ are linear terms $x_1{:}\sigma_1,\ldots, x_n{:}\ \sigma_n \vdash t{:}\tau$ expressed as morphisms:

$$\sigma_1 \otimes \ \ldots \ \otimes \sigma_n \overset{t}{\rightarrow} \tau;$$

In order to express the semantics of linear type theory we have to define interpretation functions for objects and morphisms. The interpretation function for the objects is

$$i{:}\ \mathfrak{T} \rightarrow \mathfrak{C}_{obj}$$

and it assigns an object in $\mathfrak{C}_{obj}$ to every type from $\mathfrak{T}$ as follows:

$$\begin{array}{lcllcl} i(\sigma) & = & [\![\sigma]\!] & \qquad & i(\sigma \otimes \tau) & = & [\![\sigma]\!] \otimes [\![\tau]\!] \\ i(I) & = & [\![I]\!] & & i(\sigma \multimap \tau) & = & Hom([\![\sigma]\!], [\![\tau]\!]) \end{array}$$

The interpretation function for morphism

$$j{:}\ \mathfrak{T}erm \rightarrow \mathfrak{C}_{morp}$$

assigns a morphism in $\mathfrak{C}_{morp}$ for every typed linear term from a set $\mathfrak{T}erm$.

$$j{:}\ \tau \vdash \mathrm{t}{:}\ \sigma \longrightarrow [\![t]\!]{:}\ [\![\sigma]\!] \ \longrightarrow [\![\tau]\!]$$

Combinators of linear type theory are interpreted in $\mathfrak{C}$ as follows:

$$\begin{array}{rcl} j(Id_\sigma) & = & id_{[\![\sigma]\!]} \\ j(assr_{\sigma,\tau,\chi}) & = & a_{[\![\sigma]\!],[\![\tau]\!],[\![\chi]\!]} \\ j(assl_{\sigma,\tau,\chi}) & = & a^{-1}_{[\![\sigma]\!],[\![\tau]\!],[\![\chi]\!]} \\ j(swap_{\sigma,\tau}) & = & c_{[\![\sigma]\!],[\![\tau]\!]} \\ j(open_\sigma) & = & l_{[\![\sigma]\!]} \\ j(close_\sigma) & = & l^{-1}_{[\![\sigma]\!]} \\ j(eval_{\sigma,\tau}) & = & \varepsilon_{[\![\sigma]\!],[\![\tau]\!]} \end{array}$$

We denote the composition of combinators α and β, where α: $\sigma \rightarrow \tau$ and β: $\tau \rightarrow \theta$, as $\beta \circ \alpha$. We interpret that as a composition of morphism in category $\mathfrak{C}$

$$j(\beta \circ \alpha) = j(\beta) \circ j(\alpha)$$

The semantics of the linear type theory is a pair of functions

$$(i,j).$$

3. Multiplicative fragment of predicate linear logic

After defining linear type theory we can introduce predicate linear logic. We use only multiplicative fragment of linear logic that can be modelled in symmetric monoidal closed category. The multiplicative fragment of predicate linear

logic [7] contains the multiplicative connectives with their neutral elements from the propositional linear logic. A linear formula φ has a form defined by the following BNF rule:

$$\varphi ::= p \mid 1 \mid \bot \mid \varphi \otimes \varphi \mid \varphi \parr \varphi \mid \varphi \multimap \varphi \mid \varphi^{\bot} \mid !\varphi \mid ?\varphi \mid P(t_1,\ldots,t_n) \mid \forall x\, \varphi \mid \exists x\, \varphi,$$

where:

- formula $P(t_1,\ldots,t_n)$ is a predicate expressing relations between terms $(t_1,\ldots,t_n)$;
- formula $\forall x\, \varphi$ expresses universal quantifier applied on the formula and $\exists x\, \varphi$ expresses existential quantifier applied on the formula. Quantifiers bind occurrence of variables.

In accordance to arity we can distinguish predicate symbols as:

- unary $P(t)$ which expresses some property of a term t;
- binary $P(t_1,t_2)$ which expresses some binary relation between terms t_1 and t_2;
- n-ary $P(t_1,\ldots,t_n)$ which expresses n-ary relation between terms $(t_1,\ldots,t_n)$.

Quantifiers have the same meaning as in the predicate logic and De Morgan's rules are valid:

$$\begin{aligned} (\forall \boldsymbol{x}\, \boldsymbol{\varphi})^{\bot} &\equiv \exists \boldsymbol{x}\, \boldsymbol{\varphi}^{\bot} \\ (\exists \boldsymbol{x}\, \boldsymbol{\varphi})^{\bot} &\equiv \forall \boldsymbol{x}\, \boldsymbol{\varphi}^{\bot} \end{aligned} \tag{30}$$

De Morgan's rules are also valid for predicates:

$$\begin{aligned} (\boldsymbol{P}(\boldsymbol{t}))^{\bot} &\equiv \boldsymbol{P}^{\bot}(\boldsymbol{t}) \\ (\boldsymbol{P}^{\bot}(\boldsymbol{t}))^{\bot} &\equiv \boldsymbol{P}(\boldsymbol{t}) \end{aligned} \tag{31}$$

Because we added new forms for the formulae, we have to extend deduction system with new deduction rules. For the quantifiers we have the following deduction rules:

$$\frac{\boldsymbol{\Gamma},\boldsymbol{\varphi}[\boldsymbol{t}/\boldsymbol{x}]\vdash\boldsymbol{\Delta}}{\boldsymbol{\Gamma},(\forall \boldsymbol{x})\boldsymbol{\varphi}\vdash\boldsymbol{\Delta}}(\forall \boldsymbol{x}_{\mathbf{L}}) \tag{32}$$

$$\frac{\boldsymbol{\Gamma}\vdash\boldsymbol{\varphi},\boldsymbol{\Delta}}{\boldsymbol{\Gamma}\vdash(\forall \boldsymbol{x})\boldsymbol{\varphi},\boldsymbol{\Delta}}(\forall \boldsymbol{x}_{\mathbf{R}}) \tag{33}$$

$$\frac{\boldsymbol{\Gamma},\boldsymbol{\varphi}\vdash\boldsymbol{\Delta}}{\boldsymbol{\Gamma},(\exists \boldsymbol{x})\boldsymbol{\varphi}\vdash\boldsymbol{\Delta}}(\exists \boldsymbol{x}_{\mathbf{L}}) \tag{34}$$

$$\frac{\boldsymbol{\Gamma}\vdash\boldsymbol{\varphi}[\boldsymbol{t}/\boldsymbol{x}],\boldsymbol{\Delta}}{\boldsymbol{\Gamma}\vdash(\exists \boldsymbol{x})\boldsymbol{\varphi},\boldsymbol{\Delta}}(\exists \boldsymbol{x}_{\boldsymbol{R}}) \tag{35}$$

where Γ is a finite sequence of linear formulae $\varphi_1,\ldots,\varphi_n$ and in the rules $\forall x_L$ and $\exists x_R$ x have no free occurrence in Γ and Δ. These rules express how to introduce the universal and the existential quantifier into formula φ.

4. Categorical model of multiplicative fragment of predicate linear logic

In the previous section we constructed the semantics of linear type theory as symmetric monoidal closed category. Next we construct a categorical model of the multiplicative fragment of predicate linear logic using the symmetric monoidal closed category, defined in previous sections.

Any elementary sentence is interpreted as an object, basic type in category $\mathfrak{C}$.

Neutral element

$$1 \equiv I$$

is interpreted as terminal object I of the category $\mathfrak{C}$. The neutral element $\perp$ is dual to 1

$$1^{\perp} \equiv \perp,$$

therefore from the properties of category $\perp$ is interpreted as initial object of $\mathfrak{C}$.

For the interpretation of negation we use the following equivalence:

$$[\![\varphi^{\perp}]\!] \equiv [\![\varphi \multimap \perp]\!]$$

According to [20] we interpret every sequent

$$\varphi_1,\ldots,\varphi_n \vdash \psi$$

as morphism

$$[\![\varphi_1]\!] \otimes \cdots \otimes [\![\varphi_n]\!] \longrightarrow [\![\psi]\!]$$

in $\mathfrak{C}$, where $[\![\varphi]\!]$ expresses object, a representation of the formulae φ in $\mathfrak{C}$.

Connectives are interpreted as morphisms in category $\mathfrak{C}$ as follows:

$$\begin{array}{ccc} [\![\varphi]\!] \otimes [\![\psi]\!] & \longrightarrow & [\![\varphi \otimes \psi]\!] \\ [\![\varphi]\!] \parr [\![\psi]\!] & \longrightarrow & [\![\varphi \parr \psi]\!] \\ [\![\varphi]\!] \multimap [\![\psi]\!] & \longrightarrow & [\![\psi^{\varphi}]\!] \end{array}$$

Because the symmetric monoidal closed category is Cartesian closed category, the existence of objects $[\![\varphi \otimes \psi]\!]$, $[\![\varphi \parr \psi]\!]$ and exponential object $[\![\psi^{\varphi}]\!]$ arises from its properties [19].

Now we define interpretation of unary predicate $P(t)$. A predicate $P(t)$ is a property of the value of a term t of type σ. Because $[\![\sigma]\!]$ is an object in $\mathfrak{C}$, the interpretation of the predicate $P(t)$

$$[\![P(t)]\!] \subseteq [\![\sigma]\!]$$

is a subset of the object $[\![\sigma]\!]$ in the $\mathfrak{C}$.

We define functors and adjoint functors in order to express the semantics of the modal operator *of course !*, the *existential* and *universal quantifiers*.

A functor is morphism between categories. A functor $F: \mathfrak{C} \longrightarrow \mathfrak{D}$ is a pair of functions (F_0,F_1) [17]

$$\begin{array}{ccc} F_0: \mathfrak{C}_{obj} & \longrightarrow & \mathfrak{D}_{obj} \\ F_1: \mathfrak{C}_{morp} & \longrightarrow & \mathfrak{D}_{morp}, \end{array}$$

for which holds:

- if $f : A \to B$ is morphism in $\mathfrak{C}$ then $F_1(f) : F_0(A) \to F_0(B)$ in $\mathfrak{D}$;
- for any object A in $\mathfrak{C}$ holds $F_1(id_A\}) = id_{F0(A)}$;
- if the composition $g \circ f$ is in $\mathfrak{C}$ then the composition $F_1(g) \circ F_1(f)$ is defined in $\mathfrak{D}$ and holds $F_1(g \circ f) = F_1(g) \circ F_1(f)$.

A functor $F: \mathfrak{C} \to \mathfrak{C}$ is called endofunctor over category $\mathfrak{C}$. We define adjoint endofunctors over category $\mathfrak{C}$.

Let $F: \mathfrak{C} \to \mathfrak{C}$ and $G: \mathfrak{C} \to \mathfrak{C}$ be endofunctors [17] and Id be identity functor. We say that:

- F is a left adjoint to the endofunctor G, $F \dashv G$ and
- G is a right adjoint to the endofunctor F, $G \vdash F$

if there is a natural transformation

$$\eta: Id \to G \circ F$$

such that for any objects A, B in $\mathfrak{C}$ and any morphism $f: A \to G(B)$ it exists unique morphism

$$g: F(A) \to B$$

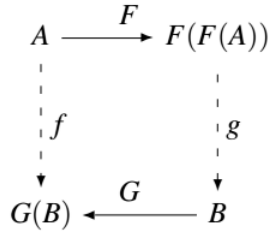
in $\mathfrak{C}$, for which holds

$$f = G(g) \circ \eta A.$$

Adjunction means that there exact correspondence exists between morphisms $A \to G(B)$ and $B \to F(A)$, i.e. the Homsets

$$\boldsymbol{Hom}(\boldsymbol{F}(\boldsymbol{A}), \boldsymbol{B}) \cong \boldsymbol{Hom}(\boldsymbol{A}, \boldsymbol{G}(\boldsymbol{B})) \tag{36}$$

are isomorphic. Adjunction can be illustrated also by the following commuting diagram.



The property (36) is useful in defining semantics of modal operator *of course !* as follows. Let F and G be a pair of adjoint endofunctors

$$F \dashv G$$

in $\mathfrak{C}$. We define this modal operator as a composition

$$[\![!]\!]: G \circ F$$

such that for any object $[\![\varphi]\!]$

$$(G(F[\![\varphi]\!])) = [\![!\,\varphi]\!]$$

it returns an object isomorphic with $[\![\varphi]\!]$, i.e. we can model unexhaustible resource $[\![\varphi]\!]$ by composition of adjoint functors as it is illustrated in Figure 1.

To interpret quantifiers we use adjoint functors, too.

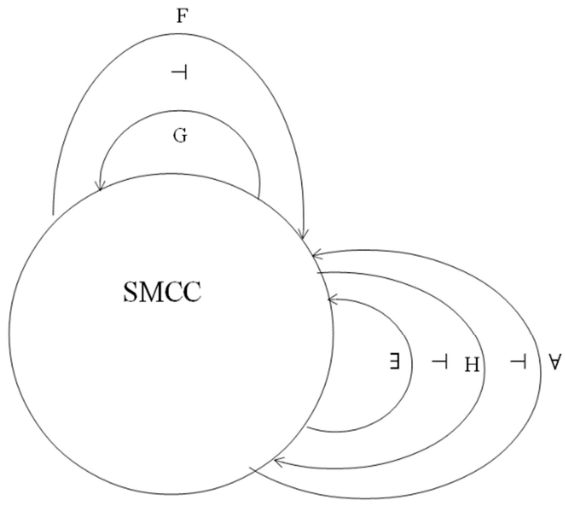


Fig. 1. Model of predicate linear logic

Let $[\![P(t)]\!] \subseteq [\![\sigma]\!]$ be an interpretation of the unary predicate symbol, where $[\![\sigma]\!]$ is an object in $\mathfrak{C}$. We consider a variable $y{:}\tau$ and we construct a predicate $P(t,y)$ interpreted as

$$[\![P(t,y)]\!] \subseteq [\![\sigma]\!] \times [\![\tau]\!],$$

where y has no free occurrence in t. We construct an auxiliary endofunctor H as follows:

$$\mathrm{H}\colon \mathcal{P}([\![\sigma]\!]) \longrightarrow \mathcal{P}([\![\sigma]\!] \times [\![\tau]\!]),$$

where $\mathcal{P}([\![\sigma]\!])$ is a power set over $[\![\sigma]\!]$.

Now we define a left adjoint functor of H that is interpretation of existential quantifier ∃

$$[\![\exists]\!] \dashv \mathrm{H}$$

as

$$[\![\exists]\!]\colon \mathcal{P}([\![\sigma]\!] \times [\![\tau]\!]) \longrightarrow \mathcal{P}([\![\sigma]\!]),$$

that for quantified formula $\exists y.P(t,y)$ returns a value of type τ (if it exists) satisfying predicate $P(t,y)$:

$$[\![\exists y.P(t,y)]\!] = \{[\![t]\!] \in [\![\sigma]\!] |\ \text{exists a value in } [\![\tau]\!] \vDash [\![P(t,y)]\!]\}.$$

Because of duality between existential and universal quantifiers in (31), we interpret universal quantifier as a right adjoint to the auxiliary functor H

$$\mathrm{H} \dashv [\![\forall]\!].$$

Interpretation of quantifiers by adjoint functors is illustrated in the Figure 1.

In the following text we explain how deduction and proofs can be interpreted by morphisms in our model.
The identity rule

$$\frac{}{\varphi \vdash \varphi}\,(\mathrm{id})$$

is interpreted as identical morphism:

$$id_{[\![\varphi]\!]}\colon [\![\varphi]\!] \longrightarrow [\![\varphi]\!].$$

The proofs

$$\frac{\begin{array}{c}\tau_1\\ \vdots\end{array}}{\Gamma \vdash \varphi, \Delta} \qquad\qquad \frac{\begin{array}{c}\tau_2\\ \vdots\end{array}}{\Gamma' \vdash \psi, \Delta'}$$

are interpreted as morphisms

$$f\colon [\![\Gamma]\!] \longrightarrow [\![\varphi \& \Delta]\!] \qquad\qquad g\colon [\![\Gamma']\!] \longrightarrow [\![\psi \& \Delta']\!]$$

in category $\mathfrak{C}$. The proof of the multiplicative conjunction

$$\frac{\begin{array}{c}\tau_1\\ \vdots\end{array}\quad \begin{array}{c}\tau_2\\ \vdots\end{array}}{\Gamma \vdash \varphi, \Delta \quad \Gamma' \vdash \psi, \Delta'}{\Gamma, \Gamma' \vdash \varphi \otimes \psi, \Delta, \Delta'}\,(\otimes_{\mathrm{R}})$$

is interpreted as morphism

$$[\![\Gamma]\!] \otimes [\![\Gamma']\!] \xrightarrow{\ f \otimes g\ } [\![\varphi \otimes \psi \& \Delta \& \Delta']\!]$$

in category $\mathfrak{C}$.

Our model of predicate linear logic is constructed as a symmetrical monoidal closed category of types together with appropriate adjoint functors for modal operator and quantifiers. A proof of a formula is modeled as a finite path of category morphism.

Conclusions

Linear logic has many important properties useful for describing and verifying various program systems. Its dynamic nature, expressing causality, non determinism and handling resources make it the most appropriate logical system for computer science. In this paper we defined predicate linear logic, its deduction system and we constructed categorical model based on symmetric monoidal closed category. This category with types as objects enables direct connection with computing, where types and typed data structures play an important role. Categories provide many useful structures that we can also use in constructing models of logical systems. In this paper we used special properties of adjunct endofunctors for modeling the modal operators expressing non exhaustibility of resources and for modeling quantifiers. The model of predicate linear logic defined in this paper will serve for our further research, either for specifying contracts and dependencies between components in modeling component based systems, or in modeling observable behavior of such systems on the base of coalgebras.

Acknowledgment

This work has been supported by APVV-0008 0 Grant: Modelling, simulation and implementation of GPGPU-enabled architectures of high-throughput network security tools.

References

[1] Girard J.-Y., Linear logic, Theoretical Computer Science 1987, 50, 1-102.

[2] Novitzká V., Mihályi D., Slodičák V., Categorical models of logical systems in the mathematical theory of programming, P. U. M. A 2006, 17, 3-4, 367-378.

[3] Novitzká V., Mihályi D., Resource-oriented programming based on linear logic, Acta Polytechnica Hungarica 2007, 4(2), 157-166.

[4] Mihályi D., Novitzká V., Prazňák P., Popovec P., Network routing modelled by game semantics, Studia Universitatis Babes-Bolyai, Informatica 2012, 57, 4, 19-29.

[5] Mihályi D., Novitzká V., Towards the knowledge in coalgebraic model of IDS, Computing and Informatics 2014, 33, 1, 61-78.

[6] Steingartner W., Novitzká V., Benčková M., Prazňák P., Considerations and ideas in component programming - towards to formal specification, Proc. of 25th Central European Conference on Information and Intelligent Systems CECIIS, Varaždin, Sept. 17-19, 2014, 332-339.

[7] Mihályi D., Novitzká V., What about linear logic in computer science? Department of Computers and Informatics, Technical University of Košice, Acta Polytechnica Hungarica 2013, 10, 4.

[8] Lincoln P., Linear Logic, SRI and Stanford University, 1992.

[9] Novitzká V., Mihályi D., Slodičák V., Linear logical reasoning on programming, Acta Electrotechnica et Informatica 2006, 3, 6.

[10] Girard J.-Y., Linear Logic: Its Syntax and Semantics, Cambridge University Press, 2003.

[11] Yetter D.N., Quantales and (noncommutative) linear logic, Journal of Symbolic Logic 1990, 55(1), 41-64.

[12] Girard J.-Y., Taylor P., Lafont Y., Proofs and Types, Cambridge University Press, New York 1989.

[13] Girard J.-Y., On the Meaning of Logical Rules I: Syntax vs. Semantics, Institut de Mathematiques de Luminy, UPR 9016- CNRS 163, Avenue de Luminy, Case 930, F-13288 Marseille Cedex 09, 1998.

[14] Ambler S.J., First order linear logic in symmetric monoidal closed categories, PhD. Thesis, University of Edinburgh, 1991.

[15] Abramsky S., Computational interpretations of linear logic, Technical Report 90/20, Department of Computing, Imperial College, 1990, 1-15.

[16] Hasegawa M., Categorical Glueing and Logical Predicates for Models of Linear Logic, Kyoto University, Research Institute for Mathematical Sciences, 1999

[17] Novitzká V., Slodičák V., Kategorické štruktúry a ich aplikácie v informatike 2010, ISBN 978-80-89284-67-2.

[18] de Paiva V., Categorical Semantics of Linear Logic for All, Palo Alto Research Center, Palo Alto 2006.

[19] Barr M., Wells Ch., Category Theory for Computing Science, Prentice Hall International Ltd., Hertfordshire 1990.

[20] Melliès P.-A., Categorical Semantics of Linear Logic, Panoramas et Syntheses 27, Citeseer, 2009.

[21] Mihályi D., Novitzká V., Ľaľová M., Intrusion detection system episteme, Central European Journal of Computer Science 2012, 2, 3, 214-220.

[22] Slodičák V., Some Useful Structures for Categorical Approach for Program Behavior, Journal of Information and Organizational Sciences 2011, 35, 1, 99-109.