



**synrc** research center s.r.o.  
ROHÁČOVA 141/18, PRAHA 3 13000, CZECH REPUBLIC

**Categorical semantic of general purpose functional language  
that compiles to normalized lambda terms  
of non-recursive intermediate language with dependent types**

Technical Article

Maxim Sokhatsky, Synrc Research Center

December 2015

# Содержание

<b>1</b>	<b>Вступ</b>	<b>3</b>
1.1	Functional Language Compiler Pipeline . . . . .	3
1.1.1	General Purpose Functional Language Exe . . . . .	3
1.1.2	Intermediate Language Om . . . . .	3
1.1.3	Target Erlang VM and LLVM platforms . . . . .	3
1.2	Basic Theories . . . . .	4
1.2.1	Category Theory . . . . .	4
1.2.2	Лямбда числення . . . . .	6
1.2.3	Числення процесів . . . . .	7
1.2.4	Інтуїтіоністична теорія типів Мартіна Льюфа . . . . .	8
1.2.5	Логіка та квантори . . . . .	8

# 1 Вступ

## 1.1 Functional Language Compiler Pipeline

This article describes the frontend language along with its categorical semantic and Categorical Abstract Machine, but that compiles not to CAM, but a non-recursive subset of dependent type theory from which you can extract LLVM code, untyped lambda code with erased type information.

### 1.1.1 General Purpose Functional Language Exe

General purpose function language with functors (lambdas on types), recursive algebraic types, higher order functions and embedded process calculus with corecursion. This language will be called Om and dedicated to be high level general purpose functional programming language frontend to small core of dependent type system without recursion called Exe. This language intended to be useful enough to encode KVS, N2O and BPE applications.

### 1.1.2 Intermediate Language Om

An intermediate Exe language is based on Henk [?] languages described first by Erik Meyer and Simon Peyton Jones in 1997. Later on in 2015 Morte implementation of Henk design appeared in Haskell, using Boem-Berrarducci encoding of non-recursive lambda terms. It is based only on  $\pi$ ,  $\lambda$  and *apply* constructions, one axiom and four deduction rules. The design of Exe language resembles Henk and Morte both design and implementation. This language intended to be small, concise, easy provable and clean and produce verifiable pieces of code that can be distributed over the networks and compiled at target with safe linkage.

### 1.1.3 Target Erlang VM and LLVM platforms

This work expects to compile to limited target platforms. For now Erlang, Haskell and LLVM is awaiting. Erlang version is expected to be useful both on LING and BEAM Erlang virtual machines.

## 1.2 Basic Theories

### 1.2.1 Category Theory

Category theory is widely used as an instrument for mathematicians for software analysis. Category theory could be treated as an abstract algebra of functions. Let's define a Category formally: **Category** consists of two lists: the one is morphisms (arrows) and the second is objects (domains and codomains of arrows) along with associative operation of composition and unit morphism that exists for all objects in category.

The formation axioms of objects and arrows are not given here and autopostulating yet. Formation axioms will be introduced during exponential definition. Objects  $A$  and  $B$  of an arrow  $f : A \rightarrow B$  are called **domain** and **codomain** respectively.

Intro axioms – associativity of composition and left/right unit arrow compositions show that categories are actually typed monoids, which consist of morphisms and operation of composition. There are many language to show the semantic of categories such as commutative diagrams and string diagrams however here we define here in proof-theoretic manner:

$$\begin{array}{c}
 \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash g : B \rightarrow C}{\Gamma \vdash g \circ f : A \rightarrow C} \qquad \frac{}{\Gamma \vdash id_A : A \rightarrow A} \\
 \\
 \frac{\Gamma \vdash f : B \rightarrow A \quad \Gamma \vdash g : C \rightarrow B \quad \Gamma \vdash h : D \rightarrow C}{\Gamma \vdash (f \circ g) \circ h = f \circ (g \circ h) : D \rightarrow A} \qquad \frac{\Gamma \vdash f : A \rightarrow B}{\Gamma \vdash f \circ id_A = f : A \rightarrow B} \\
 \\
 \frac{}{\Gamma \vdash id_B \circ f = f : A \rightarrow B}
 \end{array}$$

Composition shows an ability to connect the area of values of the previous evaluation (codomain) and are of definition the next evaluation (domain). Composition is fundamental property of morphisms.

1.  $A : *$
2.  $A : *, B : * \implies f : A \rightarrow B$
3.  $f : B \rightarrow C, g : A \rightarrow B \implies f \circ g : A \rightarrow C$
4.  $(f \circ g) \circ h = f \circ (g \circ h)$
5.  $A \implies id : A \rightarrow A$
6.  $f \circ id = f$
7.  $id \circ f = f$

## Algebraic Datatypes

Після операції композиції, як способу конструювання нових об'єктів за допомогою морфізмів далі йде операція конструювання добутка двох об'єктів певної категорії, разом з добутком морфізмів зі спільним доменом, необхідних для визначення декартового добутка  $A \times B$ .

Це є внутрішня мова декартової категорії, у якій для будь яких двох доменів існує їх декартова сума (кодобутку) та декартовий добуток (косума, кортеж), за допомогою яких конструюються суми-протоколи та добутки-повідомлення, а також існує  $\perp$  тип-термінал, та  $\top$  тип-котермінал. Термінальними типами зручно термінувати рекурсивні типи даних, такі як списки. Ми будемо розглядати тільки категорії які мають добутки та суми.

Добуток має природні елімінатори  $\pi$  зі спільним доменом, які є морфізмами-проекціями об'єктів добутку. Сума має обернені елімінатори  $\sigma$  зі спільним кодоменом. Як видно добуток є дуальний до суми з точністю до напрямлення стрілок, таким чином елімінатори  $\pi$  та  $\sigma$  є оберненими, тобто  $\pi \circ \sigma = \sigma \circ \pi = id$ .

$$\begin{array}{c}
 \frac{\Gamma x : A \times B}{\Gamma \vdash \pi_1 : A \times B \rightarrow A; \Gamma \vdash \pi_2 : A \times B \rightarrow B} \quad \frac{}{\Gamma \vdash \top} \\
 \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash a \mid b : A \otimes B} \\
 \frac{}{\Gamma \vdash \perp} \quad \frac{\Gamma x : A \otimes B}{\Gamma \vdash \sigma_1 : A \rightarrow A \otimes B; \Gamma \vdash \sigma_2 : B \rightarrow A \otimes B}
 \end{array}$$

Також додамо тут аксіому множення морфізмів, яка впливає з визначення добутку, яка необхідна для забезпечення аплікативного програмування.

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash g : A \rightarrow C \quad \Gamma \vdash B \times C}{\Gamma \vdash \langle f, g \rangle : A \rightarrow B \times C}$$

$$\begin{aligned}
 \pi_1 \circ \langle f, g \rangle &= f \\
 \pi_2 \circ \langle f, g \rangle &= g \\
 \langle f \circ \pi_1, f \circ \pi_2 \rangle &= f \\
 \langle f, g \rangle \circ h &= \langle f \circ h, g \circ h \rangle \\
 \langle \pi_1, \pi_2 \rangle &= id
 \end{aligned}$$

### 1.2.2 Exponential

Будучи внутрішньою мовою декартово-замкненої категорії лямбда числення окрім змінних та констант у вигляді термів пропонує операції абстракції та аплікації, що визначає достатньо лаконічну та потужну структуру обчислень з функціями вищих порядків, та метатипизаціями, такими як System F, яка була запропонована вперше Робіном Мілнером в мові ML, та зараз присутня в більш складних, таких як System F $\omega$ , системах Haskell та Scala.

Щоб пояснити функції з категоріальної точки зору потрібно пояснити категоріальні експоненти  $f : A^B$ , які є аналогами функціональних просторів  $f : A \rightarrow B$ . Так як ми вже визначили добутки та термінали, то ми можемо визначити і експоненти, опускаючи усі категоріальні подробиці ми визначимо конструювання функції (операція абстракції), яка параметризується змінною  $x$  у середовищі  $\Gamma$ ; та її елімінатора – операції аплікації функції до аргументу. Так визначається декартово-замкнена категорія. Визначається також рекурсивний механізм виклику функції з довільною кількістю аргументів.

$$\begin{array}{c}
 \frac{\Gamma x : A \vdash M : B}{\Gamma \vdash \lambda x . M : A \rightarrow B} \\
 \\
 \frac{\Gamma f : A \rightarrow B \quad \Gamma a : A}{\Gamma \vdash apply\ f\ a : (A \rightarrow B) \times A \rightarrow B} \\
 \\
 \frac{\Gamma \vdash f : A \times B \rightarrow C}{\Gamma \vdash curry\ f : A \rightarrow (B \rightarrow C)}
 \end{array}
 \qquad
 \begin{array}{l}
 apply \circ \langle (curry\ f) \circ \pi_1, \pi_2 \rangle = f \\
 curry\ apply \circ \langle g \circ \pi_1, \pi_2 \rangle = g \\
 apply \circ \langle curry\ f, g \rangle = f \circ \langle id, g \rangle \\
 (curry\ f) \circ g = curry\ (f \circ \langle g \circ \pi_1, \pi_2 \rangle) \\
 curry\ apply = id \\
 \\
 \text{Об'єкти : } \perp \mid \rightarrow \mid \times \\
 \text{Морфізми : } id \mid f \circ g \mid \langle f, g \rangle \mid apply \mid \lambda \mid curry
 \end{array}$$

### 1.2.3 Process Calculus

Теорія  $\pi$ -числення процесів Роберта Мілнера є основним формалізмом обчислювальної теорії розподілених систем та її імплементації. З часів виникнення CSP числення розробленого Хоаром, Мілнеру вдалося значно розширити та адаптувати теорію до сучасних телекомунікаційних вимог, як наприклад хендовери в мобільних мережах. Основні теореми в моделі  $\pi$ -числення стосуються непротиворечивості та неблокованості у синхронному виконанні мобільних процесів. Так як сучасний Web можна розглядати як телекомунікаційну систему, тому у розробці додатків можна покладатися у тому числі і на такі моделі як  $\pi$ -числення.

#### Процеси і протоколи

Також ми анонсуємо процес як фундаментальний тип даних, подібний до функції але який здатний тримати певний стан у вигляді типа коротежа та є морфізмом-одиницею типу свого стану.

$$\begin{array}{c}
 \frac{\Gamma \vdash E, \Sigma, X \quad \Gamma \vdash action : \Sigma \times X \rightarrow \Sigma \times X}{\Gamma \vdash spawn\ action : \pi_\Sigma} \\
 \\
 \frac{\Gamma \vdash pid : \pi_\Sigma \quad \Gamma \vdash msg : \Sigma}{\Gamma \vdash join\ msg\ pid : \Sigma \times \pi_\Sigma \xrightarrow{\bullet} \Sigma; \Gamma \vdash send\ msg\ pid : \Sigma \times \pi_\Sigma \rightarrow \Sigma} \\
 \\
 \frac{\Gamma \vdash L : A + B, R : X + Y \quad \Gamma \vdash M : A \rightarrow X, N : B \rightarrow Y}{\Gamma \vdash receive\ L\ M\ N : L \xrightarrow{\bullet} R}
 \end{array}$$

#### Алгебра процесів

Алгебра процесів визначає базові операції мультиплексування двох чи декількох протоколів в рамках одного процесу (добуток), а також паралельного та повністю ізольованого запуску включно зі стеком та областю пам'яті (сума) на віртуальній машині.

$$\begin{array}{lcl}
 \oplus & : & \pi \parallel \pi \\
 \otimes & : & \pi \mid \pi
 \end{array}$$

#### Типи процесів

$$\begin{array}{lcl}
 action & : & proc_{Proc} (BPE \mid \Sigma) \times process \rightarrow process \\
 event & : & proc_{App} (N2O \mid \Sigma) \times cx \rightarrow cx \\
 operation & : & proc_{Store} (KVS \mid \Sigma) \times container \rightarrow container
 \end{array}$$

### 1.2.4 Інтуїціоністична теорія типів Мартіна Льюфа

Системи з залежними типами як верифікаційні математичні формальні моделі для доведення коректності. Система  $\Sigma$  та  $\Pi$  типів, як кванторів існування та узагальнення. Системи Mizar, Coq, Agda, Idris, F\*, Lean. Ми будемо використовувати автоматизовану систему теорем Lean від Microsoft Research.

Розбудовуючи певний фреймворк чи систему конструктивними методами так чи інакше доведеться зробити певний вибір у мові та способі кодування. Так при розробці теорії абстрактної алгебри в Coq були використані поліморфні індуктивні структури [?]. Однак Agda та Idris використовують для побудови алгебраїчної теорії типи класів, а у Idris взагалі відсутні поліморфічні індуктивні структури та коіндуктивні структури. В Lean теж відсутні коіндуктивні структури проте повністю реалізована теорія HoTT на нерекурсивних поліморфних структурах що об'єднує основні чотири класи математичних теорій: логіка, топологія, теорія множин, теорія типів. Як було показано Стефаном Касом [?], одна з стратегій імплементації типів класів — це використання поліморфних структур. Хоча в Lean також підтримуються типи класів нами була вибрана стратегія імплементації нашої теорії з використанням нерекурсивних індуктивних структур, що дозволить нам оперувати з персистентними структурами на низькому рівні. Крім того такий спосіб кодування ієрархій повністю відповідає семантиці Erlang, де немає типів класів, а дані передаються запаковані в кортежі-структури.

### 1.2.5 Logic and Quantification

Далі йдуть квантори  $\forall$  та  $\exists$  які теж виражаються як конструкції типів:

$$\frac{\Gamma x : A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash \Pi(x : A)B} \qquad \frac{\Gamma \vdash a : A \quad \Gamma x : A \vdash B \quad \Gamma b : B(x = a)}{\Gamma \vdash (a, b) : \Pi(x : A)B}$$

$$\frac{\Gamma x : A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash \Sigma(x : A)B} \qquad \frac{\Gamma \vdash a : A \quad \Gamma x : A \vdash B \quad \Gamma b : B(x = a)}{\Gamma \vdash (a, b) : \Sigma(x : A)B}$$

$$\frac{\Gamma \vdash x : A \quad \Gamma \vdash x' : A}{\Gamma \vdash Id_A(x, x')}$$

рефлексивність	:	$Id_A(a, a)$
підстановка	:	$Id_A(a, a') \rightarrow B(x = a) \rightarrow B(x = a')$
симетричність	:	$Id_A(a, b) \rightarrow Id_A(b, a)$
транзитивність	:	$Id_A(a, b) \rightarrow Id_A(b, c) \rightarrow Id_A(a, c)$
конгруентність	:	$(f : A \rightarrow B) \rightarrow Id_A(x, x') \rightarrow Id_B(f(x), f(x'))$



## Список литературы

- [1] Per Martin-Löf *Intermediate Language* 1984
- [2] P.Wadler *Recursive types for free!* 2014
- [3] Per Martin-Löf *Intuitionistic Type Theory*. 1984
- [4] A.Baer *Programming with Algebraic Effects and Handlers* 2012
- [5] P.Wadler *Monads for functional programming*
- [6] PL. Curien *Category theory: a programming language-oriented introduction* 2008
- [7] Robin Milner. *A Calculus of Communicating Systems*. 1986.
- [8] William Lawvere. *Conceptual Mathematics*. 1997.
- [9] Сандерс Мак Лейн. *Категории для работающего математика*. 2004.
- [10] И.Бакур, А.Деляну. *Введение в теорию категори и функторов*. 1972.
- [11] Robin Milner. *Communicating and Mobile Systems: The  $\pi$ -calculus*. 1999.
- [12] Robin Milner. *The Polyadic  $\pi$ -Calculus: A Tutorial*. 1993.
- [13] T.Coquand, G.Huet *The Calculus of Constructions*. 1988
- [14] A.Chlipala *Certified Programming with Dependent Types* 2015
- [15] E.Brady *Programming in IDRIS: A Tutorial* 2015
- [16] C.McBride, R.Patterson *Applicative programming with effects* 2002