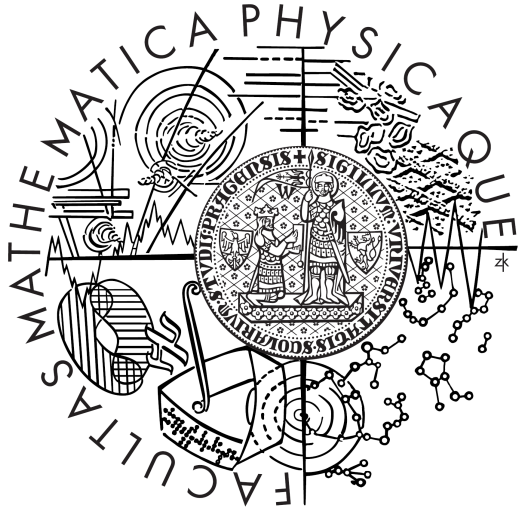
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Tomáš Křen

Typed Functional Genetic Programming

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. Petr Pudlák, Ph.D.
Study programme: Theoretical Computer Science
Specialization: Non-Procedural Programming and Artificial Intelligence

Prague 2013

Contents

1. Introduction

2. Definitions

Let us first say some basic definitions.

2.1 Lambda term

Let V be set of *variable names*.
Let C be set of *constant names*.
Then Λ is set of λ-*terms* inductively defined as follows:

$$x \in V \Rightarrow x \in \Lambda$$
$$c \in C \Rightarrow c \in \Lambda$$
$$M, N \in \Lambda \Rightarrow (MN) \in \Lambda$$
$$x \in V, M \in \Lambda \Rightarrow (\lambda x.M) \in \Lambda$$

2.2 Type

Let A be set of *atomic type names*.
Then $\mathbb{T}$ is set of *types* inductively defined as follows:

$$\alpha \in A \Rightarrow \alpha \in \mathbb{T}$$
$$\sigma, \tau \in \mathbb{T} \Rightarrow (\sigma \to \tau) \in \mathbb{T}$$

2.3 Statement of a form $M : \sigma$

Let Λ be set of λ-*terms*.
Let $\mathbb{T}$ be set of *types*.
A *statement* $M : \sigma$ is a pair $(M, \sigma) \in \Lambda \times \mathbb{T}$.
$M : \sigma$ is vocalized as *"M has type σ"*.[1]
The type σ is the *predicate* and the term M is the *subject* of the statement.

2.4 Context

Let $\Gamma \in \mathfrak{P}(\Lambda \times \mathbb{T})$. ($\Gamma$ is a set of *statements* of a form $M : \sigma$.)
Then Γ is *context* if it obeys following conditions[2]:

$$\forall (x, \sigma) \in \Gamma : x \in V \cup C$$
$$\forall s_1, s_2 \in \Gamma : s_1 \neq s_2 \Rightarrow \pi_1(s_1) \neq \pi_1(s_2)$$

In other words context is a set of statements with distinct variables or constants as subjects.

[1] $M : \sigma$ can be also imagined as $M \in \sigma$

[2] The π_1 corresponds to the projection of the first component of the Cartesian product.

2.5 Statement of a form $\Gamma \vdash M : \sigma$

By writing $\Gamma \vdash M : \sigma$ we say *statement* $M : \sigma$ *is derivable from context* Γ .

We construct valid statements of form $\Gamma \vdash M : \sigma$ by using inference rules.

2.6 Inference rule

Basically speaking, inference rules are used for deriving statements of a form $\Gamma \vdash M : \sigma$ from yet derived statements of such a form. Those inference rules are written in the following form:

$$\frac{\Gamma_1 \vdash M_1 : \sigma_1 \qquad \Gamma_2 \vdash M_2 : \sigma_2 \quad \cdots \quad \Gamma_n \vdash M_n : \sigma_n}{\Gamma_{n+1} \vdash M_{n+1} : \sigma_{n+1}}$$

Suppose we have yet derived statements $\Gamma_1 \vdash M_1 : \sigma_1, \Gamma_2 \vdash M_2 : \sigma_2, \ldots, \Gamma_n \vdash M_n : \sigma_n$. It allows as to use the inference rule to derive statement $\Gamma_{n+1} \vdash M_{n+1} : \sigma_{n+1}$.

For deriving statements including types of a form $(\sigma \to \tau)$ are essential those two inference rules:

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau}$$

$$\frac{\Gamma \cup \{(x, \sigma)\} \vdash M : \tau}{\Gamma \vdash (\lambda x.M) : \sigma \to \tau}$$

This kind of inference rules allows us to derive new statements from yet derived statements, but what if we do not have any statement yet? For this purpose we have other kinds of inference rules such as *axiom* inference rule:

$$\frac{(x, \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

Let us consider an example statement of a form $\Gamma \vdash M : \sigma$:

$$\{\} \vdash (\lambda f.(\lambda x.(fx))) : (\sigma \to \tau) \to (\sigma \to \tau)$$

This statement is derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{(f, \sigma \to \tau) \in \{(f, \sigma \to \tau), (x, \sigma)\}}{\{(f, \sigma \to \tau), (x, \sigma)\} \vdash f : \sigma \to \tau} \qquad \dfrac{(x, \sigma) \in \{(f, \sigma \to \tau), (x, \sigma)\}}{\{(f, \sigma \to \tau), (x, \sigma)\} \vdash x : \sigma}}{\{(f, \sigma \to \tau), (x, \sigma)\} \vdash (fx) : \tau}}{\{(f, \sigma \to \tau)\} \vdash (\lambda x.(fx)) : \sigma \to \tau}}{\{\} \vdash (\lambda f.(\lambda x.(fx))) : (\sigma \to \tau) \to (\sigma \to \tau)}$$

2.7 Term generating grammar

Inference rules are good for deriving statements of a form $\Gamma \vdash M : \sigma$, but our goal is slightly different; we would like to generate many λ-terms M for a given type σ and context Γ.

Our approach will be to take each inference rule and transform it to a rule of term generating grammar. With this term generating grammar it will be much easier to reason about generating λ-terms.

It won't be a grammar in classical sense because we will be operating with infinite sets of nonterminal symbols and rules. [3]

Let $Non = Type \times Context$ be our *nonterminal* set. So for every $i \in Non$ is $i = (\sigma_i, \Gamma_i)$.

Let's consider each relevant inference rule and its corresponding grammar rule.

First inference rule is *implication elimination* also known as *modus ponens*:

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau}$$

For every $\sigma, \tau \in \mathbb{T}$ and for every *context* $\Gamma \in \mathfrak{P}(\Lambda \times \mathbb{T})$ there is a grammar rule of a form[4]:

$$(\tau, \Gamma) \longmapsto \Big((\sigma \to \tau, \Gamma) \ \_ \ (\sigma, \Gamma) \Big)$$

Second inference rule is *implication introduction*:

$$\frac{\Gamma \cup \{(x, \sigma)\} \vdash M : \tau}{\Gamma \vdash (\lambda x.M) : \sigma \to \tau}$$

$\forall \sigma, \tau \in \mathbb{T}$ $\forall$*context* $\Gamma \in \mathfrak{P}(\Lambda \times \mathbb{T})$ $\forall x \in V$ such that there is no $(x, \rho) \in \Gamma$ there is a grammar rule:

$$(\sigma \to \tau, \Gamma) \longmapsto \Big(\ \lambda \ \mathrm{X} \ . \ (\tau, \Gamma \cup \{(x, \sigma)\}) \ \Big)$$

Third inference rule is *axiom*:

$$\frac{(x, \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

$\forall \sigma \in \mathbb{T}$ $\forall$*context* $\Gamma \in \mathfrak{P}(\Lambda \times \mathbb{T})$ $\forall x \in V \cup C$ such that $(x, \sigma) \in \Gamma$ there is a grammar rule:

$$(\sigma, \Gamma) \longmapsto \mathrm{X}$$

[3] TODO : mention terminal symbols - situation around variables and their construction with ' symbol.

[4] Terminal symbols for parenthesis and normally *space* now $\_$ (for *function application* operator) are visually highlighted.

We will demonstrate λ-term generation on example. Again on $(\lambda f.(\lambda x.(fx)))$. We would like to generate λ-term of a type $(\sigma \to \tau) \to (\sigma \to \tau)$ with $\Gamma = \{\}$.

$$
\begin{aligned}
&((\sigma \to \tau) \to (\sigma \to \tau), \{\}) \\
\longmapsto &\Big(\lambda\ \mathrm{f}\ .\ (\sigma \to \tau, \{(f, \sigma \to \tau)\}) \Big) \\
\longmapsto &\Big(\lambda\ \mathrm{f}\ .\ \Big(\lambda\ \mathrm{x}\ .\ (\tau, \{(f, \sigma \to \tau), (x, \sigma)\}) \Big) \Big) \\
\longmapsto &\Big(\lambda\ \mathrm{f}\ .\ \Big(\lambda\ \mathrm{x}\ .\ \Big((\sigma \to \tau, \{(f, \sigma \to \tau), (x, \sigma)\}) \_\_ (\sigma, \{(f, \sigma \to \tau), (x, \sigma)\})\Big) \Big) \Big) \\
\longmapsto &\Big(\lambda\ \mathrm{f}\ .\ \Big(\lambda\ \mathrm{x}\ .\ \Big(\mathrm{f} \_\_ (\sigma, \{(f, \sigma \to \tau), (x, \sigma)\})\Big) \Big) \Big) \\
\longmapsto &\Big(\lambda\ \mathrm{f}\ .\ \Big(\lambda\ \mathrm{x}\ .\ \Big(\mathrm{f} \_\_ \mathrm{x}\Big) \Big) \Big)
\end{aligned}
$$

2.7.1 "Barendregt-like" inference and grammar rules

Inference rule 1:

$$\frac{\Gamma \cup \{(x_1, \tau_1), \ldots, (x_n, \tau_n)\} \vdash M : \alpha}{\Gamma \vdash (\lambda x_1 \ldots x_n.M) : \tau_1 \to \cdots \to \tau_n \to \alpha}$$

... there is a grammar rule:

$$(\tau_1 \to \cdots \to \tau_n \to \alpha, \Gamma) \longmapsto \Big(\lambda x_1 \ldots x_n.\ (\alpha, \Gamma \cup \{(x_1, \tau_1), \ldots, (x_n, \tau_n)\}) \Big)$$

Inference rule 2:

$$\frac{(f, \tau_1 \to \cdots \to \tau_n \to \alpha) \in \Gamma \qquad \Gamma \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma \vdash M_n : \tau_n}{\Gamma \vdash (fM_1 \ldots M_n) : \alpha}$$

... there is a grammar rule:

$$(\alpha, \Gamma) \longmapsto \Big(\mathrm{f}\ \_\_ (\tau_1, \Gamma) \_\_ \ldots \_\_ (\tau_n, \Gamma) \Big)$$

2.8 Inhabitation tree

2.8.1 Inhabitation Machine

2.9 Roadmap

2.10 Conversion to SKI combinators

2.11 Genetic Programming

2.11.1 Term generating

2.11.2 Crossover

2.11.3 Mutation

3. Designed system

3.1 Top level view

3.1.1 Comments about main source files

Eva.hs

3.2 Term generating

3.2.1 A* algorithm

3.3 Crossover

3.3.1 Finding same types

3.3.2 Two basic options

Resolve problems with free variables or avoid variables completely.

3.4 Mutation

3.4.1 Using term generation

4. Problems

In this section will be presented usage of the system in order to solve specific problems.

4.1 Even Parity Problem

4.2 Big Context

4.3 Fly

4.4 Simple Symbolic Regression

4.5 Artificial Ant

4.6 Boolean Alternate

5. Conclusion