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

Dedication.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In .... date .... signature of the author

Název práce: Název práce

Autor: Jméno a příjmení autora

Katedra: Název katedry či ústavu, kde byla práce oficiálně zadána

Vedoucí diplomové práce: Jméno a příjmení s tituly, pracoviště

Abstrakt:

Klíčová slova:

Title:

Author: Jméno a příjmení autora

Department: Název katedry či ústavu, kde byla práce oficiálně zadána

Supervisor: Jméno a příjmení s tituly, pracoviště

Abstract:

Keywords:

Contents

Introduction

1. Definitions

Let us first say some basic definitions.

1.1 Genetic Programming

1.1.1 Term generating

1.1.2 Crossover

1.1.3 Mutation

1.2 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:

$$\begin{aligned} x \in V \cup C &\Rightarrow x \in \Lambda \\ M, N \in \Lambda &\Rightarrow (MN) \in \Lambda \\ x \in V, M \in \Lambda &\Rightarrow (\lambda x.M) \in \Lambda \end{aligned}$$

TODO

- TALK ABOUT "parenthesis" conventions (and packing of lambda abstractions).
- BETTER SPECIFICATION V is infinite spočetná (?countable)

1.3 Type

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

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

TODO

- TALK ABOUT $\tau_1 \to \cdots \to \tau_n \to \alpha$
- TALK ABOUT arrow/parenthesis conventions.

1.4 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.

1.5 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.

TODO: TALK ABOUT Context represents library/building blocks.

1.6 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.

1.7 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:

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

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

[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.

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:

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

1.8 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 us 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]:

$$(\tau, \Gamma) \longmapsto \Big((\sigma \to \tau, \Gamma) \,\llcorner\!\lrcorner\, (\sigma, \Gamma) \Big)$$

[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.

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:

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

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 $\Gamma = \{\}$.

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

1.8.1 "Barendregt-like" inference and grammar rules

Inference rule 1:

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

Proof of correctness:

$$\cfrac{\cfrac{\cfrac{\Gamma \cup \{(x_1,\tau_1), \dots, (x_n,\tau_n)\} \vdash M : \alpha}{\Gamma \cup \{(x_1,\tau_1), \dots, (x_{n-1},\tau_{n-1})\} \cup \{(x_n,\tau_n)\} \vdash M : \alpha}}{\Gamma \cup \{(x_1,\tau_1), \dots, (x_{n-1},\tau_{n-1})\} \vdash (\lambda x_n.M) : \tau_n \to \alpha}}{\cfrac{\vdots}{\Gamma \vdash (\lambda x_1 \dots x_n.M) : \tau_1 \to \dots \to \tau_n \to \alpha}}$$

... there is a grammar rule:

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

Inference rule 2:

$$\frac{(f, \rho_1 \to \cdots \to \rho_m \to \alpha) \in \Gamma \qquad \Gamma \vdash M_1 : \rho_1 \quad \cdots \quad \Gamma \vdash M_m : \rho_m}{\Gamma \vdash (f M_1 \ldots M_m) : \alpha}$$

Proof of correctness (**TODO REPAIR** Conceptually it is ok but there is sazbabug somewhere):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boxed{(f, \rho_1 \to \cdots \to \rho_m \to \alpha) \in \Gamma}}{\Gamma \vdash f : \rho_1 \to \cdots \to \rho_m \to \alpha} \quad \boxed{\Gamma \vdash M_1 : \rho_1}}{\Gamma \vdash (f M_1) : \rho_2 \to \cdots \to \rho_m \to \alpha} \quad \ddots}{\vdots}}{\Gamma \vdash (f M_1 \ldots M_{m-2}) : \rho_{m-1} \to \rho_m \to \alpha} \quad \boxed{\Gamma \vdash M_{m-1} : \rho_{m-1}}}{\Gamma \vdash (f M_1 \ldots M_{m-1}) : \rho_m \to \alpha} \quad \boxed{\Gamma \vdash M_m : \rho_m}}{\Gamma \vdash (f M_1 \ldots M_m) : \alpha}$$

... there is a grammar rule:

$$(\alpha, \Gamma) \longmapsto \Bigg(\mathrm{f} \ \text{\textvisiblespace}\ (\rho_1, \Gamma)\ \text{\textvisiblespace}\ \ldots\ \text{\textvisiblespace}\ (\rho_m, \Gamma) \Bigg)$$

TODO

- SHOW correctness of those inference rules by composing them of $E^{\to}$, $I^{\to}$ and *axiom*.
- SHOW more examples of inference rules transformed into grammar rules.
- DESCRIBE general algorithm for this transformation.
- TALK ABOUT $\tau_1 \to \cdots \to \tau_n \to \alpha$
- TALK ABOUT $\beta\eta$-normal form which is generated by this method.

1.9 Inhabitation tree

Now we will introduce *Inhabitation tree*, structure slightly different from *Inhabitation machine*, which was introduced in [1] by Henk Barendregt. We can think about Inhabitation tree as about unfolded Inhabitation machine. The motivation for using Inhabitation trees is belief that it will help us reason about generation of λ-terms of a given type σ and with a given context Γ.

1.9.1 Definition of Inhabitation tree

Inhabitation tree is a *rooted tree*, possibly infinite. It has two types of nodes:

- Type nodes - containing type $\sigma \in \mathbb{T}$ - aka "OR-node" , Nonterminal-node.
- Symbol nodes - containing "λ-head" (nonempty finite sequence of variable names) or constant name. - aka "AND-node" , Terminal-node.

We construct Inhabitation tree for given type σ and context Γ.
We will define Inhabitation tree by describing its construction for a given (σ, Γ). Notice that it will closely follow the rules from 1.8.1:

- The root of Inhabitation tree for (σ, Γ) is *type node* with σ as type.
- All *type nodes* have as child nodes only *symbol nodes*.
- And all *symbol nodes* have as child nodes only *type nodes*.

Now we will resolve the child nodes of the root node.
There are two cases of σ (recall 1.3):

Atomic type $\sigma = \alpha$ where $\alpha \in A$.

Function type $\sigma = \tau_1 \to \cdots \to \tau_n \to \alpha$ where $n \geq 1, \alpha \in A$.

First case **Atomic type** — i.e., $\sigma = \alpha$ where $\alpha \in A$:
For every $(f, \rho_1 \to \cdots \to \rho_m \to \alpha) \in \Gamma$ where $\alpha \in A$ there is a child *symbol node* of the root containing constant name f. This symbol node containing f has m child subtrees corresponding to Inhabitation trees for $(\rho_1, \Gamma), \ldots, (\rho_n, \Gamma)$.

Compare this case with corresponding grammar rule:

$$(\alpha, \Gamma) \longmapsto \Big(\mathrm{f} \;\llcorner\!\lrcorner\; (\rho_1, \Gamma) \;\llcorner\!\lrcorner\; \ldots \;\llcorner\!\lrcorner\; (\rho_m, \Gamma) \Big)$$

Second case **Function type** — i.e., $\sigma = \tau_1 \to \cdots \to \tau_n \to \alpha$ where $n \geq 1, \alpha \in A$:
For every $i \in \{1, \ldots, n\}$ we create new *variable name* x_i which is not yet included in context Γ as variable or constant name.

There is one and only one child *symbol node* of the root containing "λ-head" $\lambda x_1 \ldots x_n$ which stands for sequence of variable names $(x_1, \ldots, x_n)$. This symbol node containing $\lambda x_1 \ldots x_n$ has one and only one child subtree corresponding to Inhabitation trees for $(\alpha, \Gamma \cup \{(x_1, \tau_1), \ldots, (x_n, \tau_n)\})$.

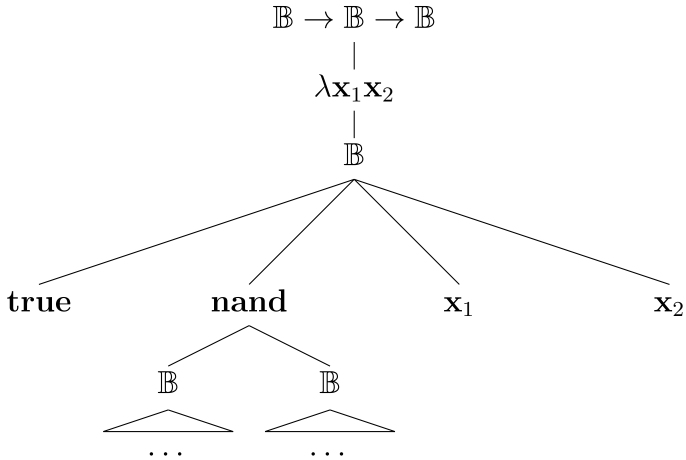
Compare this case with corresponding grammar rule:

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

Let us consider following (σ, Γ) as a simple example:

$$
\begin{aligned}
\sigma &= \mathbb{B} \to \mathbb{B} \to \mathbb{B} \\
\Gamma &= \{\ true : \mathbb{B} \\
&\quad\ ,\ nand : \mathbb{B} \to \mathbb{B} \to \mathbb{B}\ \}
\end{aligned}
$$

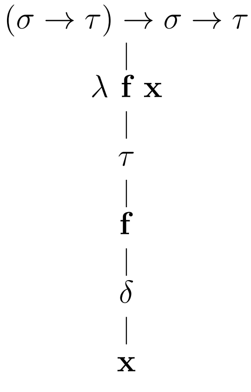
This particular (σ, Γ) results in the following tree:



Second example features our well known example:

$$
\begin{aligned}
\sigma &= (\sigma \to \tau) \to \sigma \to \tau \\
\Gamma &= \{\}
\end{aligned}
$$

Which results in following tree:



1.9.2 And-or tree and searching in Inhabitation tree

Let us consider following definition of *And-or tree*[5]:

And-or tree is a rooted tree where each node is labeled as either *and-node* or[6] *or-node*.

By *solving* And-or tree T we mean finding T' subtree of T such that it follows these conditions:

[5] **TODO**: Mention that on WIKI there is more general definition, but for our purposes is this one sufficient.

[6] xor

- The root of T' is the root of T.
- Each *and-node* in T' has all the child nodes as in T.
- Each *or-node* in T' has precisely one child node.[7]

Let us now consider following labeling of Inhabitation tree:

- **Type nodes** are labeled as **or-nodes**.
- **Symbol nodes** are labeled as **and-nodes**.

This labeling has following justification:
Selection of exactly one child node in *type node* corresponds to selection of exactly one grammar rule in order to rewrite nonterminal symbol.

Selection of all the child nodes in *symbol node* corresponds to rewriting all the nonterminal symbols in string that is being generated.

The motivation for defining *solving* of a And-or tree the way we did is that a found tree T' corresponds to generated λ-term. In order to understand this correspondence let's now talk about various tree representations of λ-terms.

1.9.3 Tree representations of λ-terms

From the definition of λ-term (1.2) we can straightforwardly derive the classical tree representation for λ-terms. Term M is translated into tree $T[M]$ by following rules:

- $x \in V \cup C$ translates into *leaf* x.
- (MN) translates into tree

$$\begin{array}{ccc} & @ & \\ \swarrow & & \searrow \\ T[M] & & T[N] \end{array}$$

- $\lambda x.M$ translates into tree

$$\begin{array}{c} \lambda x \\ | \\ T[M] \end{array}$$

We can enhance this representation by compressing consecutive lambda abstractions into one tree node like this:

- $\lambda x_1 \dots x_n.M$ translates into tree

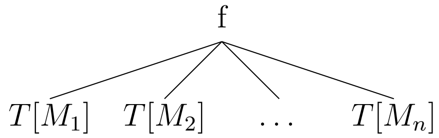
$$\begin{array}{c} \lambda x_1 \dots x_n \\ | \\ T[M] \end{array}$$

As this representation comes directly from definition it is evident that it covers all possible λ-terms.

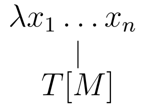
For representing expressions as trees it is however more common use a little different representation. It will also be the representation suitable for showing that *solving* Inhabitation tree generates wanted λ-term.

[7] **TODO**: MENTION why precisely one and not at least one ..or CHANGE the def.

- $x \in V \cup C$ translates into *leaf* x.
- $(fM_1M_2 \dots M_n)$ where $f \in V \cup C, n \geq 1$ translates into tree



- $\lambda x_1 \dots x_n.M$ translates into tree

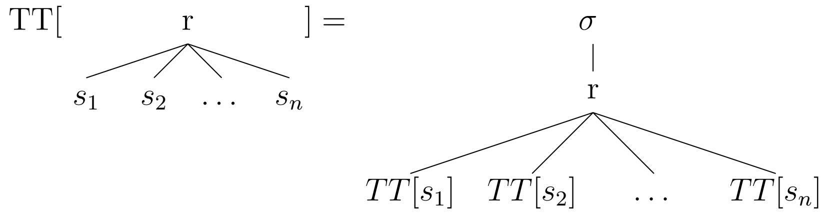


Notice that this representation does not cover all λ-terms, e.g. $(\lambda x.x)y$ is not expressible in it. But it does not bother us.

Let us now consider representation for *typed λ-terms*. Straightforward approach would be to add to each node a type entry which would be the type of the λ-term corresponding to subtree having this node as the root node.

Approach more suitable for our purpose is to add a special type node above each node. More specifically:

Let us consider tree t corresponding to a λ-term of a type σ with root r and subtrees $s_1, \dots, s_n$. Then corresponding tree $TT[t]$ for typed λ-term is obtained from the tree t as follows:



Now we can finally put the pieces together. Every solution to a Inhabitation tree has this just described tree form of a typed λ-term.

TODO

- EXAMPLES of tree representations of λ-terms
- TALK (more?) ABOUT "Barendregt-like" subsection 1.8.1
  Things about $\beta\eta$ normal form, etc.

1.9.4 Our approach to *solving* Inhabitation machine

A* algorithm

1.9.5 Inhabitation Machine

TODO

- DESCRIBE Inhabitation Machine...

1.10 Roadmap

1.11 Conversion to SKI combinators

2. Designed system

2.1 Top level view

2.1.1 Comments about main source files

Eva.hs

GP_Core.hs

2.2 Term generating

2.2.1 A* algorithm

2.3 Crossover

2.3.1 Finding same types

2.3.2 Two basic options

Resolve problems with free variables or avoid variables completely.

2.4 Mutation

2.4.1 Using term generation

3. Problems

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

3.1 Even Parity Problem

3.2 Big Context

3.3 Fly

3.4 Simple Symbolic Regression

3.5 Artificial Ant

3.6 Boolean Alternate

Conclusion

Bibliography

[1] Henk Barendregt, Wil Dekkers, Richard Statman, *Lambda Calculus With Types*. Cambridge University Press, 2010.
`http://www.cs.ru.nl/~henk/book.pdf`