Self-interpretation in Lambda Calculus

H. Geuvers

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: fall 2015

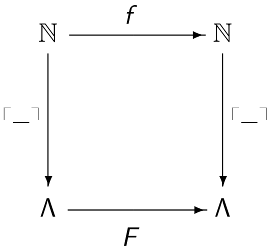
A data type D is a set with some operations (functions) on it.
An k-ary operation is a function $f : D^k \to D$.
Thereby a 0-ary operation $c : D^0 \to D$ is identified with a $c \in D$.
A datatype is determined by its operations on D:

$$\begin{array}{rcl} c_1, \ldots, c_{k_0} & : & D^0 \to D \;=\; D \\ f^1_1, \ldots, f^1_{k_1} & : & D^1 \to D \\ f^2_1, \ldots, f^2_{k_2} & : & D^2 \to D \\ & \ldots & \end{array}$$

$\texttt{Nat}$ has $\texttt{z} : \texttt{Nat}$, $\texttt{s} : \texttt{Nat} \to \texttt{Nat}$.
$\texttt{Tree}$ has $\texttt{l} : \texttt{Tree}, \texttt{p} : \texttt{Tree}^2 \to \texttt{Tree}$

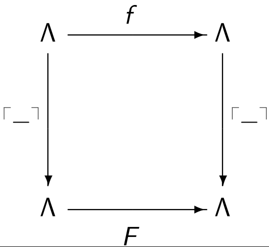
Defining functions on data types in the lambda-calculus

F λ-defines the function $f : \mathbb{N} \to \mathbb{N}$ if $\ulcorner f(n) \urcorner = F \ulcorner n \urcorner$ for all n:



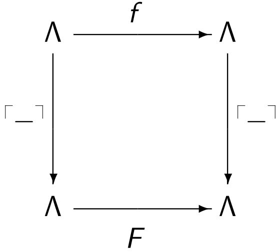
Can we enode the λ-calculus in itself?

F λ-defines the function $f : \Lambda \to \Lambda$ if $\ulcorner f(M) \urcorner = F \ulcorner M \urcorner$ for all $M \in \Lambda$.



Is $\ulcorner\ \urcorner$ just the identity?
Why is this useful?

Some functions $f : \Lambda \to \Lambda$ can only be defined on codes of terms, not on terms. So we will need to talk about codes.



EXAMPLE. There is no λ-term F such that

$$F(M\,N) = M \text{ for all } M, N$$

- There is a λ-term F such that $F\ulcorner M\,N\urcorner = \ulcorner M\urcorner$ for all M, N. (With a suitable encoding $\ulcorner - \urcorner$.)
- We also have an evaluator E:

$$E\ulcorner M\urcorner = M$$

Given $M_1, \ldots, M_k$, define

$$\langle M_1, \ldots, M_k \rangle \;:=\; \lambda z.z\, M_1 \ldots M_k$$

Define $\mathbf{U}_i^k$, with $1 \leq i \leq k$ by

$$\mathbf{U}_i^k \;:=\; \lambda x_1 \ldots x_k.x_i$$

Then

$$\boxed{\begin{array}{rcl} \mathbf{U}_i^k M_1 \ldots M_k & = & M_i \\ \langle M_1, \ldots, M_k \rangle \mathbf{U}_i^k & = & \mathbf{U}_i^k M_1 \ldots M_k \;=\; M_i \end{array}}$$

Note that $\mathbf{K} = \mathbf{U}_1^2$

Second encoding of data types (Böhm-Piperno-Guerini)

Consider the data type D with

$$c : D,\ f : D \to D,\ g : D^2 \to D$$

The Böhm-Piperno-Guerini coding (also denoted by $\ulcorner t \urcorner$) is

$$\begin{array}{rcl} \ulcorner c \urcorner & = & \lambda e.e\,\mathbf{U}_1^3\,e \\ \ulcorner f(t) \urcorner & = & \lambda e.e\,\mathbf{U}_2^3\,\ulcorner t \urcorner\,e \\ \ulcorner g(t_1,t_2) \urcorner & = & \lambda e.e\,\mathbf{U}_3^3\,\ulcorner t_1 \urcorner\,\ulcorner t_2 \urcorner\,e \end{array}$$

PROPOSITION. The constructors $(c,\ f,\ g)$ can be λ-defined: There are lambda terms F, G such that

$$\begin{array}{rcl} F \ulcorner t \urcorner & = & \ulcorner f(t) \urcorner \\ G \ulcorner t_1 \urcorner \ulcorner t_2 \urcorner & = & \ulcorner g(t_1,t_2) \urcorner \end{array}$$

PROOF. Take

$$\begin{array}{rcl} F & := & \lambda x\,e.e\,\mathbf{U}_2^3\,x\,e \\ G & := & \lambda x\,y\,e.e\,\mathbf{U}_3^3\,x\,y\,e. \end{array}$$

☻

THEOREM. Given $A_1, A_2, A_3 \in \Lambda$ there is an $H \in \Lambda$ such that

$$\begin{array}{rcl} H\ulcorner c\urcorner & = & A_1 H \\ H(\ulcorner f(t)\urcorner) & = & A_2 \ulcorner t\urcorner H \\ H(\ulcorner g(t_1, t_2)\urcorner) & = & A_3 \ulcorner t_1\urcorner\ulcorner t_2\urcorner H \end{array}$$

PROOF. Try $H = \langle\langle B_1, B_2, B_3\rangle\rangle$.

$$\begin{array}{rcll} H\ulcorner c\urcorner & = & \langle\langle B_1, B_2, B_3\rangle\rangle\ulcorner c\urcorner & \\ & = & \ulcorner c\urcorner\langle B_1, B_2, B_3\rangle & \\ & = & \langle B_1, B_2, B_3\rangle \mathbf{U}_1^3 \langle B_1, B_2, B_3\rangle & \\ & = & B_1\langle B_1, B_2, B_3\rangle & \\ & = & A_1\langle\langle B_1, B_2, B_3\rangle\rangle & \text{if } B_1 := \lambda z.A_1\langle z\rangle \\ & = & A_1 H & \\ H\ulcorner f(t)\urcorner & = & \langle B_1, B_2, B_3\rangle \mathbf{U}_2^3 \ulcorner t\urcorner\langle B_1, B_2, B_3\rangle & \\ & = & B_2\ulcorner t\urcorner\langle B_1, B_2, B_3\rangle & \\ & = & A_2\ulcorner t\urcorner H & \text{if } B_2 := \lambda x\, z.A_2 x\langle z\rangle \\ H\ulcorner g(t_1, t_2)\urcorner & = & A_3\ulcorner t_1\urcorner\ulcorner t_2\urcorner H & \text{if } B_3 := \lambda x\, y\, z.A_3 x y\langle z\rangle. \end{array}$$

☻

Data type for coding lambda terms

To encode λ-terms we consider the data type D with

$$\begin{aligned} \texttt{var} &: D \to D \\ \texttt{app} &: D \to D \to D \\ \texttt{abs} &: D \to D \end{aligned}$$

- This data types is a bit strange: there is no base case.
- It is a priori unclear how to encode the λ-terms in this data type. How to encode the variable binding? (Later slide.)

Like before, we can define the constructors `Var`, `App`, `Abs`:

$$\begin{aligned} \texttt{Var} &:= \lambda x\, e.e\, \mathbf{U}_1^3\, x\, e \\ \texttt{App} &:= \lambda x\, y\, e.e\, \mathbf{U}_2^3\, x\, y\, e \\ \texttt{Abs} &:= \lambda x\, e.e\, \mathbf{U}_3^3\, x\, e \end{aligned}$$

Like before, we have a recursion theorem:
THEOREM I. Given $A_1, A_2, A_3 \in \Lambda$ there is an $H \in \Lambda$ such that

$$\begin{array}{rcl} H(\mathtt{Var}\, x) & = & A_1\, x\, H \\ H(\mathtt{App}\, x\, y) & = & A_2\, x\, y\, H \\ H(\mathtt{Abs}\, x) & = & A_3\, x\, H \end{array}$$

PROOF. Take $H = \langle\langle B_1, B_2, B_3\rangle\rangle$ with

$$\begin{array}{rcl} B_1 & := & \lambda x\, z. A_1 x \langle z\rangle \\ B_2 & := & \lambda x\, y\, z. A_2 x y \langle z\rangle \\ B_3 & := & \lambda x\, z. A_3 x \langle z\rangle. \end{array}$$

Then the equations hold indeed. ☻.
(Exercise: Check this.)

Coding of lambda terms

Coding lambda terms $M \leadsto \ulcorner M \urcorner$

Definition (Mogensen) The coding of λ-terms inside the λ-calculus is defined as follows.

$$\begin{array}{rcl} \ulcorner x \urcorner & := & \texttt{Var}\, x \\ \ulcorner MN \urcorner & := & \texttt{App} \ulcorner M \urcorner \ulcorner N \urcorner \\ \ulcorner \lambda x.M \urcorner & := & \texttt{Abs}\, (\lambda x. \ulcorner M \urcorner) \end{array}$$

A variable x is encoded using x itself and abstraction "$\lambda x.$" is encoded using the same abstraction "$\lambda x.$".

Note: coding is not λ-definable: there is no term C such that $C\, M = \ulcorner M \urcorner$ for all M.
The reverse operation, evaluation, is λ-definable.

THEOREM. There exists a λ-term $\mathbf{E}$ (evaluator) such that for all $M \in \Lambda$

$$\mathbf{E}\ulcorner M \urcorner = M$$

PROOF. By recursion we can find an $\mathbf{E}$ such that

$$\begin{array}{rcl} \mathbf{E}(\texttt{Var}\ x) & = & x \\ \mathbf{E}(\texttt{App}\ x\ y) & = & \mathbf{E}\,x\,(\mathbf{E}\,y) \\ \mathbf{E}(\texttt{Abs}\ x) & = & \lambda z.\mathbf{E}\,(x\,z) \end{array}$$

Then

$$\begin{array}{rclclcl} \mathbf{E}(\ulcorner x \urcorner) & = & \mathbf{E}(\texttt{Var}\ x) & = & x & & \\ \mathbf{E}(\ulcorner MN \urcorner) & = & \mathbf{E}(\texttt{App}\ulcorner M \urcorner \ulcorner N \urcorner) & = & \mathbf{E}\ulcorner M \urcorner(\mathbf{E}\ulcorner N \urcorner) & = & MN \\ \mathbf{E}(\ulcorner \lambda x.M \urcorner) & = & \mathbf{E}(\texttt{Abs}(\lambda x.\ulcorner M \urcorner)) & = & \lambda x.\mathbf{E}\ulcorner M \urcorner & = & \lambda x.M. \end{array}$$

Filling in the details of $\mathbf{E}$ one has (writing $\mathbf{C} := \lambda x\,y\,z.x\,z\,y$)

$$\mathbf{E} = \langle\langle \mathbf{K}, \mathbf{S}, \mathbf{C} \rangle\rangle.$$

☻

Recursion for lambda terms using the encoding

We can state the recursion theorem for the encoded lambda terms slightly differently, as follows.

THEOREM II. Given $A_1, A_2, A_3 \in \Lambda$ there is an $H \in \Lambda$ such that

$$\begin{array}{rcl} H\,\ulcorner x\urcorner & = & A_1\,x\,H \\ H\,\ulcorner M\,N\urcorner & = & A_2\,\ulcorner M\urcorner\,\ulcorner N\urcorner\,H \\ H\,\ulcorner \lambda x.M\urcorner & = & A_3\,(\lambda x.\ulcorner M\urcorner)\,H \end{array}$$

PROOF. According to Theorem I, there is an H satisfying

$$\begin{array}{rcl} H(\mathtt{Var}\,x) & = & A_1\,x\,H \\ H(\mathtt{App}\,x\,y) & = & A_2\,x\,y\,H \\ H(\mathtt{Abs}\,x) & = & A_3\,x\,H \end{array}$$

These equations immediately imply the ones of the statement of Theorem II (check this!), so the same H suffices. ☻.

Application 1

If you see someone coming out of 'arrivals' in an airport, you cannot determine where he/she comes from.
Similarly, there is no F such that for all $X, Y \in \Lambda$

$$F(X\,Y) = X$$

(Given the outcome of applying a function on an argument, there is no way we can determine the function that produced this outcome.)

PROPOSITION. There exists an $F_i \in \Lambda$, $i \in \{1, 2\}$ such that

$$F_i \ulcorner X_1\, X_2 \urcorner = \ulcorner X_i \urcorner.$$

PROOF. We do this for $i = 1$. By the Recursion Theorem II, there exists F_1 s.t.

$$F_1(\ulcorner X_1\; X_2 \urcorner) = A_2 \ulcorner X_1 \urcorner \ulcorner X_2 \urcorner F_1 = \ulcorner X_1 \urcorner, \text{ taking } A_2 = \mathbf{U}_1^3.$$

This suffices. ☻

Second fixed point theorem*

LEMMA. There exists a term $\mathtt{Num} \in \Lambda$ such that for all $M \in \Lambda$

$$\mathtt{Num}\ulcorner M \urcorner =_\beta \ulcorner\ulcorner M \urcorner\urcorner$$

PROOF. Use recursion (Theorem I) for the lambda calculus data type with

$$\begin{array}{rcl} A_1\, x\, N & = & \mathtt{App}\,\ulcorner \mathtt{Var} \urcorner\,(\mathtt{Var}\, x) \\ A_2\, m\, n\, N & = & \mathtt{App}\,(\mathtt{App}\,\ulcorner \mathtt{App} \urcorner\,(N\, m))(N\, n) \\ A_3\, m\, N & = & \mathtt{App}\,\ulcorner \mathtt{Abs} \urcorner\,(\mathtt{Abs}\,(\lambda x.N(m\, x))) \end{array}$$

SECOND FIXED POINT THEOREM. For all F there is an X with

$$F\ulcorner X \urcorner =_\beta X$$

PROOF. Let $W := \lambda z.F(\mathtt{App}\; z(\mathtt{Num}\; z))$ and $X := W\ulcorner W \urcorner$. Then

$$\begin{array}{rl} X & = W\ulcorner W \urcorner \\ & = F(\mathtt{App}\ulcorner W \urcorner(\mathtt{Num}\ulcorner W \urcorner) = F(\mathtt{App}\ulcorner W \urcorner\ulcorner\ulcorner W \urcorner\urcorner) \\ & = F\ulcorner W \ulcorner W \urcorner\urcorner = F\ulcorner X \urcorner. \end{array}$$

☻

Application 2*

Self-modifying programs
For a given T (the program transformer) there exists a program P such that

$$\begin{aligned} P\,\mathbf{c}_k &= \mathbf{c}_{k+1} \quad \text{if } k \text{ is even,} \\ &= T\,\ulcorner P \urcorner\,\mathbf{c}_k \quad \text{otherwise.} \end{aligned}$$

- On even inputs, P performs a standard operation (adding 1)
- On odd inputs, the program P first modifies its own code using T.

To find P, apply the second fixed-point theorem to

$$F := \lambda p\,x.\texttt{if}\ (\text{Even}\,x)\ \texttt{then}\ (x+1)\ \texttt{else}(T\,p\,x)$$