5. Self-reflection

We present the following fact with a proof depending on another fact.

5.1. Fact. Let x, y be to distinct variables (e.g. x, x' or x'', x'''). Then

$$x \neq_\lambda y.$$

Proof. Use Fact 4.17. If $x =_l y$, then x has two normal forms: itself and y. ■

5.2. Application. $\mathsf{K} \neq_\lambda \mathsf{I}$.

Proof. Suppose $\mathsf{K} = \mathsf{I}$. Then

$$\begin{aligned} x &= \mathsf{K}xy \\ &= \mathsf{I}\mathsf{K}xy \\ &= \mathsf{K}\mathsf{I}xy \\ &= \mathsf{I}y \\ &= y, \end{aligned}$$

a contradiction. ■

5.3. Application. There is no term P_1 such that $P_1(xy) =_\lambda x$.

Proof. If P_1 would exist, then as $\mathsf{K}xy = x = \mathsf{I}x$ one has

$$\mathsf{K}x = P_1((\mathsf{K}x)y) = P_1(\mathsf{I}x) = \mathsf{I}.$$

Therefore we obtain the contradiction

$$x = \mathsf{K}xy = \mathsf{I}y = y. \blacksquare$$

In a sense this is a pity. The agents, that lambda terms are, cannot separate two of them that are together. When we go over to codes of lambda terms the situation changes. The situation is similar for proteins that cannot always cut into parts another protein, but are able to have this effect on the code of the proteins, the DNA.

Data types

Before we enter the topic of coding of lambda terms, it is good to have a look at some datatypes.

Context-free languages can be considered as algebraic data types. Consider for example

$$\boxed{\begin{array}{lcl} S & \to & 0 \\ S & \to & S^+ \end{array}}$$

This generates the language

$$\text{Nat} = \{0, 0^+, 0^{++}, 0^{+++}, \ldots\}$$

that is a good way to represent the natural numbers

$$0, 1, 2, 3, \ldots$$

Another example generates binary trees.

$$\begin{array}{lcl} S & \to & \spadesuit \\ S & \to & \bullet SS \end{array}$$

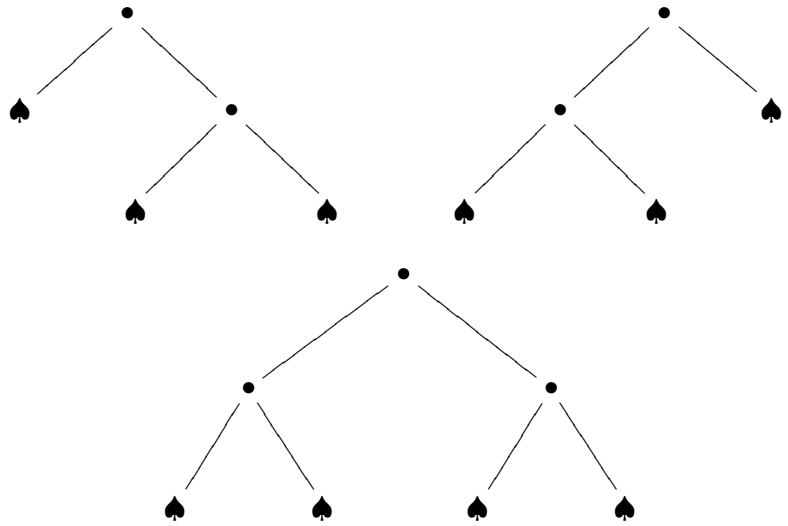
generating the language that we call *Tree*. It is not necessary to use parentheses. For example the words

$$\bullet\spadesuit\bullet\spadesuit\spadesuit$$
$$\bullet\bullet\spadesuit\spadesuit\spadesuit$$
$$\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit$$

are in Tree. With parentrheses and commas these expressions become more readable for humans, but these auxiliary signs are not necessary:

$$\bullet(\spadesuit, \bullet(\spadesuit, \spadesuit))$$
$$\bullet(\bullet(\spadesuit, \spadesuit), \spadesuit)$$
$$\bullet(\bullet(\spadesuit, \spadesuit), \bullet(\spadesuit, \spadesuit))$$

These expressions have as 'parse trees' respectively the following:



One way to represent as lambda terms such data types is as follows.

5.4. Definition. (Böhm and Berarducci)
(i) An element of Nat, like 0^{++} will be represented first like

$$s(sz)$$

and then like

$$\lambda sz.s(sz).$$

If n is in Nat we write $\ulcorner n \urcorner$ for this lambda term. So $\ulcorner 2 \urcorner \equiv \lambda sz.s(sz)$, $\ulcorner 3 \urcorner \equiv \lambda sz.s(s(sz))$. Note that $\ulcorner n \urcorner \equiv \lambda sz.s^n z \equiv \mathsf{C}_n$.

(ii) An element of Tree, like $\bullet\spadesuit\bullet\spadesuit\spadesuit$ will be represented first by

$$bs(bss)$$

and then by

$$\lambda bs.bs(bss).$$

This lambda term is denoted by $\ulcorner\bullet\spadesuit\bullet\spadesuit\spadesuit\urcorner$, in this case. In general a tree t in Tree will be represented as lambda term $\ulcorner t\urcorner$.

Now it becomes possible to compute with trees. For example making the mirror image is performed by the term

$$F_{\text{mirror}} \equiv \lambda tbs.t(\lambda pq.bqp)s.$$

The operation of enting one tree at the endpoints of another tree is performed by

$$F_{\text{enting}} \equiv \lambda t_1 t_2 bs.t_1 b(t_2 bs).$$

The attentive reader is advised to make exercises 5.4 and 5.5.

Tuples and projections

For the efficient coding of lambda terms as lambda terms a different representation of datatypes is needed. First we find a way to connect terms together in such a way, that the components can be retrieved easily.

5.5. Definition. Let $\vec{M} \equiv M_1, \ldots, M_n$ be a sequence of lambda terms. Define

$$\langle M_1, \ldots, M_n\rangle \equiv \lambda z.zM_1, \ldots, M_n.$$

Here the variable z should not be in any of the M's. Define

$$U_i^n \equiv \lambda x_1, \ldots, x_n.x_i.$$

5.6. Proposition. *For all natural numbers i, n with $1 \leq i \leq n$, one has*

$$\langle M_1, \ldots, M_n\rangle U_i^n = M_i.$$

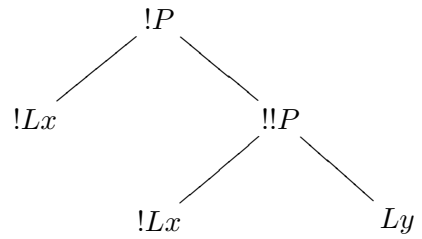
5.7. Corollary. *Define $P_i^n \equiv \lambda z.zU_i^n$. Then $P_i^n\langle M_1, \ldots, M_n\rangle = M_i$.*

Now we introduce a new kind of binary trees. At the endpoints there is not a symbol $\spadesuit$ but a variable x made into a leaf Lx. Moreover, any such tree may get ornamented with a !. Such binary trees we call labelled trees or simply ltrees.

5.8. Definition. The data type of ltrees is defined by the following context-free grammar. The start symbol is `ltree`.

`ltree`	$\rightarrow$	L `var`
`ltree`	$\rightarrow$	P `ltree ltree`
`ltree`	$\rightarrow$	! `ltree`
`var`	$\rightarrow$	x
`var`	$\rightarrow$	`var`$'$

A typical ltree is $!P!Lx!!P!LxLy$ or more readably $!P(!Lx, !!P(!Lx, Ly))$. It can be representated as a tree as follows.



We say that for ltree there are three *constructors*. A binary constructor P that puts two trees together, and two unary constructors L and !. L makes from a variable an ltree and ! makes from an ltree another one.

Now we are going to represent these expressions as lambda terms.

5.9. Definition. (Böhm, Piperno and Guerrini)
(i) We define three lambda terms $F_L, F_P, F_!$ to be used for the representation of ltree.

$$\begin{aligned} F_L &\equiv \lambda xe.eU_1^3xe; \\ F_P &\equiv \lambda xye.eU_2^3xye; \\ F_! &\equiv \lambda xe.eU_3^3xe. \end{aligned}$$

These definitions are a bit more easy to understand if written according to their intended use (do exercise 5.7).

$$\begin{aligned} F_Lx &= \lambda e.eU_1^3xe; \\ F_Pxy &= \lambda e.eU_2^3xye; \\ F_!x &= \lambda e.eU_3^3xe. \end{aligned}$$

(ii) For an element t of ltree we define the representing lambda term $\ulcorner t \urcorner$.

$$\begin{aligned} \ulcorner Lx \urcorner &= F_Lx; \\ \ulcorner Pt_1t_2 \urcorner &= F_P\ulcorner t_1 \urcorner\ulcorner t_2 \urcorner; \\ \ulcorner !t \urcorner &= F_!\ulcorner t \urcorner. \end{aligned}$$

Actually this is just a mnemonic. We want that the representations are normal forms, do not compute any longer.

$$\begin{aligned} \ulcorner Lx \urcorner &\equiv \lambda e.eU_1^3xe; \\ \ulcorner Pt_1t_2 \urcorner &\equiv \lambda e.eU_2^3\ulcorner t_1 \urcorner\ulcorner t_2 \urcorner e; \\ \ulcorner !t \urcorner &\equiv \lambda e.eU_3^3\ulcorner t \urcorner e. \end{aligned}$$

The representation of the data was chosen in such a way that computable function on them can be easily represented. The following result states that there exist functions on the represented labelled trees such that their action on a composed tree depend on the components and that function in a given way.

5.10. PROPOSITION. *Let A_1, A_2, A_3 be given lambda terms. Then there exists a lambda term H such that*[11].

$$\begin{aligned} H(F_L x) &= A_1 x H \\ H(F_P x y) &= A_2 x y H \\ H(F_! x) &= A_3 x H \end{aligned}$$

PROOF. We try $H \equiv \langle\langle B_1, B_2, B_3\rangle\rangle$ where the $\vec{B}$ are to be determined.

$$\begin{aligned} H(F_L x) &= \langle\langle B_1, B_2, B_3\rangle\rangle(F_L x) \\ &= F_L x\langle B_1, B_2, B_3\rangle \\ &= \langle B_1, B_2, B_3\rangle U_1^3 x\langle B_1, B_2, B_3\rangle \\ &= U_1^3 B_1, B_2, B_3 x\langle B_1, B_2, B_3\rangle \\ &= B_1 x\langle B_1, B_2, B_3\rangle \\ &= A_1 x H, \end{aligned}$$

provided that $B_1 \equiv \lambda x b.A_1 x\langle b\rangle$. Similarly

$$\begin{aligned} H(F_P x y) &= B_2 x y\langle B_1, B_2, B_3\rangle \\ &= A_2 x y H, \end{aligned}$$

provided that $B_2 \equiv \lambda x y b.A_2 x y\langle b\rangle$. Finally,

$$\begin{aligned} H(F_! x) &= B_3 x\langle B_1, B_2, B_3\rangle \\ &= A_3 x H, \end{aligned}$$

provided that $B_3 \equiv \lambda x b.A_3 x\langle b\rangle$. ■

Stil we have as goal to represent lambda terms as lambda terms in nf, such that decoding is possible by a fixed lambda term. Moreover, finding the code of the components of a term M should be possible from the code of M, again using a lambda term. To this end the (represented) constructors of ltree, $F_L, F_P, F_!$, will be used.

5.11. DEFINITION. (Mogensen) Define for a lambda term M its code $\ulcorner M\urcorner$ as follows.

$$\begin{aligned} \ulcorner x\urcorner &\equiv \lambda e.eU_1^3 xe &&= F_L x; \\ \ulcorner MN\urcorner &\equiv \lambda e.eU_2^3 \ulcorner M\urcorner\ulcorner N\urcorner e &&= F_P \ulcorner M\urcorner\ulcorner N\urcorner; \\ \ulcorner \lambda x.M\urcorner &\equiv \lambda e.eU_3^3(\lambda x.\ulcorner M\urcorner)e &&= F_! (\lambda x.\ulcorner M\urcorner). \end{aligned}$$

[11] A weaker requirement is the following, where H of a composed ltree depends on H of the components in a given way:

$$\begin{aligned} H(F_L x) &= A_1 x(Hx) \\ H(F_P x y) &= A_2 x y(Hx)(Hy) \\ H(F_! x) &= A_3 x(Hx) \end{aligned}$$

This is called primitive recursion, whereas the proposition provides (general) recursion.

The trick here is to code the lambda with lambda itself, one may speak of an inner model of the lambda calculus in itself. Putting the ideas of Mogensen [1992] and Böhm et al. [1994] together, as done by Berarducci and Böhm [1993], one obtains a very smooth way to create the mechanism of reflection the lambda calculus. The result was already proved in Kleene [1936][12].

5.12. THEOREM. *There is a lambda term* E *(evaluator or self-interpreter) such that*

$$\begin{aligned} \mathsf{E}\ulcorner x\urcorner &= x; \\ \mathsf{E}\ulcorner MN\urcorner &= \mathsf{E}\ulcorner M\urcorner(\mathsf{E}\ulcorner N\urcorner); \\ \mathsf{E}\ulcorner \lambda x.M\urcorner &= \lambda x.(\mathsf{E}\ulcorner M\urcorner). \end{aligned}$$

It follows that for all lambda terms M *one has*

$$\mathsf{E}\ulcorner M\urcorner = M.$$

PROOF. By Proposition 5.10 for arbitary $A_1, \ldots, A_3$ there exists an E such that

$$\begin{aligned} \mathsf{E}(F_L x) &= A_1 x \mathsf{E}; \\ \mathsf{E}(F_P mn) &= A_2 mn\mathsf{E}; \\ \mathsf{E}(F_! p) &= A_3 p\mathsf{E}. \end{aligned}$$

If we take $A_1 \equiv \mathsf{K}$, $A_2 \equiv \lambda abc.ca(cb)$ and $A_3 \equiv \lambda abc.b(ac)$, then this becomes

$$\begin{aligned} \mathsf{E}(F_L x) &= x; \\ \mathsf{E}(F_P mn) &= \mathsf{E}m(\mathsf{E}n); \\ \mathsf{E}(F_! p) &= \lambda x.(\mathsf{E}(px)). \end{aligned}$$

But then (do exercise 5.9)

$$\begin{aligned} \mathsf{E}(\ulcorner x\urcorner) &= x; \\ \mathsf{E}(\ulcorner MN\urcorner) &= \mathsf{E}\ulcorner M\urcorner(\mathsf{E}\ulcorner N\urcorner); \\ \mathsf{E}(\ulcorner \lambda x.M\urcorner) &= \lambda x.(\mathsf{E}(\ulcorner M\urcorner)). \end{aligned}$$

That E is a self-interpreter, i.e. $\mathsf{E}\ulcorner M\urcorner = M$, now follows by induction on M. ■

5.13. COROLLARY. *The term* $\langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$ *is a self-interpreter for the lambda calculus with the coding defined in Definition 5.11.*

PROOF. $\mathsf{E} \equiv \langle\langle B_1, B_2, B_3\rangle\rangle$ with the $\vec{B}$ coming from the $A_1 \equiv \mathsf{K}$, $A_2 \equiv \lambda abc.ca(cb)$ and $A_3 \equiv \lambda abc.b(ac)$. Looking at the proof of 5.10 one sees

$$\begin{aligned} B_1 &= \lambda xz.A_1 x\langle z\rangle \\ &= \lambda xz.x \\ &= \mathsf{K}; \end{aligned}$$

[12] But only valid for lambda terms M without free variables.

$$
\begin{array}{rcll}
B_2 & = & \lambda xyz.A_2xy\langle z\rangle & \\
 & = & \lambda xyz.\langle z\rangle x(\langle z\rangle y) & \\
 & = & \lambda xyz.xz(yz) & \\
 & = & \mathsf{S}; & \\
B_3 & = & \lambda xz.A_3x\langle z\rangle & \\
 & = & \lambda xz.(\lambda abc.b(ac))x\langle z\rangle & \\
 & = & \lambda xz.(\lambda c.xcz) & \\
 & \equiv & \lambda xzc.xcz & \\
 & \equiv & \lambda xyz.xzy & \\
 & \equiv & \mathsf{C}, & \text{see exercise 4.6.}
\end{array}
$$

Hence $\mathsf{E} = \langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$. ■

This term

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

is truly a tribute to

Kleene, Stephen Cole
(1909-1994)

(using the family-name-first convention familiar from scholastic institutions) who invented in 1936 the first self-interpreter for the lambda calculus[13].

The idea of a language that can talk about itself has been heavily used with higher programming languages. The way to translate ('compile') these into machine languages is optimized by writing the compiler in the language itself (and run it the first time by an older *ad hoc* compiler). This possibility of efficiently executed higher programming languages was first put into doubt, but was realized by mentioned reflection since the early 1950-s and other optimalizations. The box of Pandora of the world of IT was opened.

The fact that such extremely simple (compared to a protein like titin with slightly less than 27000 aminoacids) self interpreter is possible gives hope to understand the full mechanism of cell biology and evolution. In Buss and Fontana [1994] evolution is modelled using lambda terms.

Exercises

5.1. Show that

$$
\begin{array}{rcl}
\mathsf{K} & \neq_\lambda & \mathsf{S}; \\
\mathsf{I} & \neq_\lambda & \mathsf{S}.
\end{array}
$$

.

[13] Kleene's construction was much more involved. In order to deal with the 'binding effect' of λx lambda terms where first translated into **CL**before the final code was obtained. This causes some technicalities that make the original E more complex.

5.2. Show that there is no term P_2 such that $P_2(xy) =_\lambda y$.

5.3. Construct all elements of Tree with exactly four ♠s in them.

5.4. Show that

$$\begin{aligned} F_{\text{mirror}}\ulcorner\bullet\spadesuit\bullet\spadesuit\spadesuit\urcorner &= \ulcorner\bullet\bullet\spadesuit\spadesuit\spadesuit\urcorner; \\ F_{\text{mirror}}\ulcorner\bullet\bullet\spadesuit\spadesuit\spadesuit\urcorner &= \ulcorner\bullet\spadesuit\bullet\spadesuit\spadesuit\urcorner; \\ F_{\text{mirror}}\ulcorner\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit\urcorner &= \ulcorner\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit\urcorner. \end{aligned}$$

5.5. Compute $F_{\text{enting}}\ulcorner\bullet\spadesuit\bullet\spadesuit\spadesuit\urcorner\ulcorner\bullet\bullet\spadesuit\spadesuit\spadesuit\urcorner$.

5.6. Define terms P_i^n such that for $1 \leq i \leq n$ one has

$$P_i^n\langle M_1, \ldots, M_n\rangle = M_i.$$

5.7. Show that the second set of three equations in definition 5.9 follows from the first set.

5.8. Show that given terms A_1, A_2, A_3 there exists a term H such that the scheme of primitive recurion, see footnote 5.10 is valid.

5.9. Show the last three equations in the proof of Theorem 5.12.

5.10. Construct lambda terms P_1 and P_2 such that for all terms M, N

$$P_1\ulcorner MN\urcorner = M \;\&\; P_2\ulcorner MN\urcorner = N.$$

References

Alberts, B. et al. [1997]. *The Cell*, Garland.

Barendregt, H. P. [1984]. *The Lambda Calculus, its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics 103, revised edition, North-Holland Publishing Co., Amsterdam.

Barendregt, H. P. [1997]. The impact of the lambda calculus in logic and computer science, *Bull. Symbolic Logic* **3**(2), pp. 181–215.

Berarducci, Alessandro and Corrado Böhm [1993]. A self-interpreter of lambda calculus having a normal form, *Computer science logic (San Miniato, 1992)*, Lecture Notes in Comput. Sci. 702, Springer, Berlin, pp. 85–99.

Blackmore, S. [2004]. *Consciousness, an Introduction*, Oxford University Press, Oxford.

Böhm, Corrado, Adolfo Piperno and Stefano Guerrini [1994]. λ-definition of function(al)s by normal forms, *Programming languages and systems—ESOP '94 (Edinburgh, 1994)*, Lecture Notes in Comput. Sci. 788, Springer, Berlin, pp. 135–149.

Buss, L.W. and W. Fontana [1994]. 'the arrival of the fittest': Toward a theory of biological organization, *Bulletin of Mathematical Biology* **56**(1), pp. 1–64.

Chalmers, D. [1996]. *The Conscious Mind, Towards a Fundamental Theory*, Oxford University Press, Oxford.

Chomsky, N. [1956]. Three models of the description of language, *IRE Transactions on Information Theory* **2**(3), pp. 113–124.

Church, A. [1932]. A set of postulates for the foundation of logic, *Annals of Mathematics, second series* **33**, pp. 346–366.

Church, A. [1936]. An unsolvable problem of elementary number theory, *American Journal of Mathematics* **58**, pp. 345–363.

Curry, H. B. [1930]. Grundlagen der kombinatorischen Logic,, *American Journal of Mathematics* **52**, pp. 509–536, 789–834.

Dennet, D. [1993]. *Consciousness Explained*, Penguin Books.

Goldstein, J. [1983]. *The Experience of Insight*, Shambhala.

Hofstadter, D. [1979]. *Gödel Escher Bach, An Eternal Golden Braid*, Harvester Press.

Howe, D. [1992]. Reflecting the semantics of reflected proof, *Proof Theory, ed. P. Aczel*, Cambridge University Press, pp. 229–250.

Kleene, S. C. [1936]. Lambda-definability and recursiveness, *Duke Mathematical Journal* **2**, pp. 340–353.

Kozen, Dexter C. [1997]. *Automata and computability*, Undergraduate Texts in Computer Science, Springer-Verlag, New York.

Menninger, K., M. Mayman and P. Pruyser [1963]. *The Vital Balance. The Life Process in Mental Health and Illness*, Viking.

Mogensen, Torben Æ. [1992]. Efficient self-interpretation in lambda calculus, *J. Funct. Programming* **2**(3), pp. 345–363.

Peitsch, M.C., D.R. Stampf, T.N.C. Wells and J.L. Sussman [1995]. The swiss-3dimage collection and pdb-browser on the world-wide web, *Trends in Biochemical Sciences* **20**, pp. 82–84. URL: `<www.expasy.org>`.

Schönfinkel, M. [1924]. Über die Bausteine der mathematischen Logik, *Mathematische Annalen* **92**, pp. 305–316.

Smullyan, R. [1992]. *Gödel's Incompleteness Theorems*, Oxford University Press.

Stapp, H. [1996]. The hard problem: A quantum approach, *Journal of Consciousness Studies* **3**(3), pp. 194–210.

Tarski, A. [1933/1995]. *Introduction to Logic*, Dover.

Turing, A.M. [1936]. On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society, Series 2* **42**, pp. 230–265.

Yates, M. [1998]. What computers can't do, *Plus* **5**.
URL: `<plus.maths.org/issue5/index.html>`.