Lecture 1: Combinatory Logic and Lambda Calculus

H. Geuvers

Radboud University
Nijmegen, NL

21st Estonian Winter School in Computer Science
Winter 2016

Outline

Combinatory Logic

Lambda Calculus

Combinators

$$\Sigma_{\mathbf{CL}} = \{\mathbf{I}, \mathbf{K}, \mathbf{S}, x, {}',), (, =\}$$

We introduce several simple grammars over $\Sigma_{\mathbf{CL}}$.

- $\boxed{\texttt{constant} \ := \ \mathbf{I} \mid \mathbf{K} \mid \mathbf{S}}$
- $\boxed{\texttt{variable} \ := \ x \mid \texttt{variable}'}$
- $\boxed{\texttt{term} \ := \ \texttt{constant} \mid \texttt{variable} \mid (\texttt{term}\ \texttt{term})}$
- $\boxed{\texttt{formula} \ := \ \texttt{term} = \texttt{term}}$

Intuition:
in $(F\,A)$ the term F stands for a function and A for an argument

Combinatory Logic

Axioms

$$\begin{array}{rcll} \mathbf{I}\,P &=& P & (\mathbf{I}) \\ \mathbf{K}\,P\,Q &=& P & (\mathbf{K}) \\ \mathbf{S}\,P\,Q\,R &=& P\,R\,(Q\,R) & (\mathbf{S}) \end{array}$$

Deduction rules

$$\begin{array}{rcl} & & P = P \\ P = Q & \Rightarrow & Q = P \\ P = Q, Q = R & \Rightarrow & P = R \\ P = Q & \Rightarrow & P\,R = Q\,R \\ P = Q & \Rightarrow & R\,P = R\,Q \end{array}$$

Here P, Q, R denote arbitrary terms
$\mathbf{I}\,P$ stands for $(\mathbf{I}\,P)$, $\mathbf{K}\,P\,Q$ for $((\mathbf{K}\,P)\,Q)$ and $\mathbf{S}\,P\,Q\,R$ for $(((\mathbf{S}\,P)\,Q)\,R)$
In general $P\,Q_1 \ldots Q_n \;\equiv\; (..((P\,Q_1)\,Q_2)\ldots Q_n)$ (association to the left)

Some conventions of Combinatory Logic

Consider the term $M\,P\,Q$.

- $M\,P\,Q$ denotes $(M\,P)\,Q$ and not! $M\,(P\,Q)$!!
- First apply M to P and then the result is applied to Q.
- You may view $M\,P\,Q$ as the function M given two arguments, first P and then Q.
- So an alternative writing for $M\,P\,Q$ would be $M(P, Q)$, but we will not write that!
- $M\,P\,Q$ can receive more arguments, e.g. $P\,M$, which is easy with the **CL** notation: $M\,P\,Q\,(P\,M)$.
- We write $M =_{\mathbf{CL}} P$ or just $M = P$ to denote that this equation is derivable from the axioms of Combinatory Logic using the derivation rules.
- We write $M \equiv P$ to denote that M and P are exactly the same terms.

Some magic with combinators

PROPOSITION.

- Let $\mathbf{D} \equiv \mathbf{S}\,\mathbf{I}\,\mathbf{I}$. Then (doubling)

$$\mathbf{D}\,x =_{\mathbf{CL}} x\,x.$$

- Let $\mathbf{B} \equiv \mathbf{S}\,(\mathbf{K}\,\mathbf{S})\,\mathbf{K}$. Then (composition)

$$\mathbf{B}\,f\,g\,x =_{\mathbf{CL}} f\,(g\,x).$$

- Let $\mathbf{L} \equiv \mathbf{D}\,(\mathbf{B}\,\mathbf{D}\,\mathbf{D})$. Then (self-doubling, life!)

$$\mathbf{L} =_{\mathbf{CL}} \mathbf{L}\,\mathbf{L}.$$

Proof I

Remember the Axioms

$$
\boxed{
\begin{array}{rcll}
\mathbf{I}\,P & = & P & \qquad\qquad (\mathbf{I}) \\
\mathbf{K}\,P\,Q & = & P & \qquad\qquad (\mathbf{K}) \\
\mathbf{S}\,P\,Q\,R & = & P\,R\,(Q\,R) & \qquad\qquad (\mathbf{S})
\end{array}
}
$$

Let $\mathbf{D} \equiv \mathbf{S}\,\mathbf{I}\,\mathbf{I}$. Then (doubling)

$$\mathbf{D}\,x =_{\mathbf{CL}} x\,x.$$

PROOF.

$$
\begin{array}{rcl}
\mathbf{D}\,x & \equiv & \mathbf{S}\,\mathbf{I}\,\mathbf{I}\,x \\
& = & \mathbf{I}\,x\,(\mathbf{I}\,x) \\
& = & x\,x.
\end{array}
$$

Proof II

Remember the Axioms

$$\boxed{\begin{array}{rcll} \mathbf{I}\,P & = & P & \qquad (\mathbf{I}) \\ \mathbf{K}\,P\,Q & = & P & \qquad (\mathbf{K}) \\ \mathbf{S}\,P\,Q\,R & = & P\,R\,(Q\,R) & \qquad (\mathbf{S}) \end{array}}$$

Let $\mathbf{B} \equiv \mathbf{S}\,(\mathbf{K}\,\mathbf{S})\mathbf{K}$. Then (composition)

$$\mathbf{B}\,f\,g\,x =_{\mathbf{CL}} f\,(g\,x).$$

Proof.

$$\begin{array}{rcl} \mathbf{B}\,f\,g\,x & \equiv & \mathbf{S}\,(\mathbf{K}\,\mathbf{S})\mathbf{K}\,f\,g\,x \\ & = & \mathbf{K}\,\mathbf{S}\,f\,(\mathbf{K}\,f)\,g\,x \\ & = & \mathbf{S}\,(\mathbf{K}\,f)\,g\,x \\ & = & \mathbf{K}\,f\,x\,(g\,x) \\ & = & f\,(g\,x). \end{array}$$

Proof III

Remember the Axioms

$$\begin{array}{|rcll|}\hline \mathbf{I}\,P & = & P & (\mathbf{I}) \\ \mathbf{K}\,P\,Q & = & P & (\mathbf{K}) \\ \mathbf{S}\,P\,Q\,R & = & P\,R\,(Q\,R) & (\mathbf{S}) \\ \hline \end{array}$$

Let $\mathbf{L} \equiv \mathbf{D}\,(\mathbf{B}\,\mathbf{D}\,\mathbf{D})$. Then (self-doubling)

$$\mathbf{L} =_{\mathbf{CL}} \mathbf{L}\,\mathbf{L}.$$

PROOF.

$$\begin{array}{rcl} \mathbf{L} & \equiv & \mathbf{D}\,(\mathbf{B}\,\mathbf{D}\,\mathbf{D}) \\ & = & \mathbf{B}\,\mathbf{D}\,\mathbf{D}\,(\mathbf{B}\,\mathbf{D}\,\mathbf{D}) \\ & = & \mathbf{D}\,(\mathbf{D}\,(\mathbf{B}\,\mathbf{D}\,\mathbf{D})) \\ & \equiv & \mathbf{D}\,\mathbf{L} \\ & = & \mathbf{L}\,\mathbf{L}. \end{array}$$

Substitution

Let M, L be terms and let x be a variable. The result of substitution of L for x in M, notation $M[x := L]$ is defined by recursion on M.

case for M	definition of $M[x := L]$
x	L
y	y, provided $x \not\equiv y$
$\mathbf{C}$	$\mathbf{C}$(for $\mathbf{C} \in \{\mathbf{I}, \mathbf{K}, \mathbf{S}\}$)
$P\,Q$	$(P[x := L])\,(Q[x := L])$

Proposition

If $M =_{\mathbf{CL}} N$, then $M[x := Q] =_{\mathbf{CL}} N[x := Q]$.

Examples

$$x\mathbf{I}[x := \mathbf{S}] \equiv \mathbf{S}\,\mathbf{I}.$$
$$\mathbf{K}\,\mathbf{I}\,y\,x[x := \mathbf{S}] \equiv \mathbf{K}\,\mathbf{I}\,y\mathbf{S}.$$

First insight **Combinatory Completeness**

PROPOSITION. For every term P and variable x, there is a term F (where x does not occur in F) such that

$$F\,R =_{\mathbf{CL}} P[x := R] \qquad \text{for every } R.$$

We denote this term F constructed in the proof as $[x]P$.

PROOF. Induction on the structure of P.

Case 1. $P \equiv x$. Take $[x]x \equiv \mathbf{I}$. Then

$$([x]x)R \equiv \mathbf{I}\,R =_{\mathbf{CL}} R =_{\mathbf{CL}} x[x := R].$$

Case 2. $x \notin P$. Take $[x]P \equiv \mathbf{K}\,P$. Then indeed

$$([x]P)\,R =_{\mathbf{CL}} \mathbf{K}\,P\,R =_{\mathbf{CL}} P =_{\mathbf{CL}} P[x := R].$$

Case 3. $P \equiv U\,V$. Take $[x](U\,V) \equiv \mathbf{S}\,([x]U)\,([x]V)$. Then indeed

$$([x](U\,V))\,R \equiv \mathbf{S}\,([x]U)\,([x]V)R =_{\mathbf{CL}} (([x]U)\,R)\,(([x]V)\,R) =_{\mathbf{CL}}$$

$$(U[x := R])\,(V[x := R]) =_{\mathbf{CL}} (U\,V)[x := R]. \blacksquare$$

Algorithms

The previous proof gives the following algorithm

P	$[x]P$
x	$\mathbf{I}$
P with $x \notin P$	$\mathbf{K}\,P$
$U\,V$	$\mathbf{S}\,([x]U)\,([x]V)$

There are different possible algoritms. This is quite an efficient one.

Second insight **Fixed Points**

PROPOSITION. Every combinator has a fixed point: For every term P there exists a term X such that

$$P\,X =_{\mathbf{CL}} X.$$

PROOF. Given P, define

$$\begin{array}{rcl} W & := & [x]P\,(x\,x) \\ X & := & W\,W. \end{array}$$

Then X is a so called fixed point of P.

$$X \equiv ([x]P\,(x\,x))\,W =_{\mathbf{CL}} P\,(W\,W) \equiv P\,X.$$

Hence

$$P\,X =_{\mathbf{CL}} X. \quad \square$$

L is a fixed point of **D** if one has $\mathbf{L} = \mathbf{D}\,\mathbf{L} = \mathbf{L}\,\mathbf{L}$

Intended meaning of a λ-term

The meaning of

$$\lambda x.x^2$$

is the function

$$x \longmapsto x^2$$

that assigns to x the value x^2 (x times x)
So according to this intended meaning we have

$$(\lambda x.x^2)(6) = 6^2 = 36.$$

The parentheses around the 6 are usually not written:

$$(\lambda x.x^2)6 = 36$$

Principal axiom is the β-eqality:

$$(\lambda x.M)N =_\beta M[x := N]$$

Language

Alphabet: $\Sigma = \{x, ', (,), ., \lambda, =\}$
Language: the set of lambda terms, Λ:

$$
\begin{array}{rcl}
\texttt{variable} & := & \texttt{x} \mid \texttt{variable}' \\
\texttt{term} & := & \texttt{variable} \mid (\texttt{term}\ \texttt{term}) \mid (\lambda\,\texttt{variable}\ .\ \texttt{term}) \\
\texttt{formula} & := & \texttt{term} = \texttt{term}
\end{array}
$$

Theory (we often write just $=$ for $=_\beta$)

Axioms	$(\lambda x.\, M)N$	$=_\beta$	$M[x := N]$
	M	$=_\beta$	M
Rules	$M =_\beta N$	$\Rightarrow$	$N =_\beta M$
	$M =_\beta N, N =_\beta L$	$\Rightarrow$	$M =_\beta L$
	$M =_\beta N$	$\Rightarrow$	$ML =_\beta NL$
	$M =_\beta N$	$\Rightarrow$	$LM =_\beta LN$
	$M =_\beta N$	$\Rightarrow$	$\lambda x.\, M =_\beta \lambda x.\, N$

Substitution

M	$M[x := N]$
x	N
y	y
PQ	$(P[x := N])(Q[x := N])$
$\lambda x.\, P$	$\lambda x.\, P$
$\lambda y.\, P$	$\lambda y.\, (P[x := N])$

where $y \not\equiv x$

Application associates to the left

$$P\, Q_1 \ldots Q_n \equiv (\ldots((P\, Q_1)\, Q_2) \ldots Q_n).$$

Abstraction associates to the right

$$\lambda x_1 \ldots x_n.M \equiv (\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)))).$$

Outer parentheses are often omitted. For example

$$(\lambda x.x)y \equiv ((\lambda x.x)y)$$

Bound and free variables

$\lambda x.x$ and $\lambda y.y$ acting on M both give M

Renaming bound variables

- In the term $\lambda x.M$, the 'λx' binds the x in M.
- Variables can occur free or bound.
- We don't want to distinguish between terms that only differ in their bound variables
- We write $M \equiv_\alpha N$ (or just $M \equiv N$) if N arises from M by renaming bound variables

Examples

- $\lambda x.x \equiv_\alpha \lambda y.y$
- $\lambda x\, y.x \equiv_\alpha \lambda y\, x.y$
- $\lambda x.(\lambda x.x)\, x \equiv_\alpha \lambda y.(\lambda x.x)\, y$
- $(\lambda x.(\lambda y.x\, y))\, x \equiv_\alpha (\lambda z.(\lambda y.z\, y))\, x$

Substution revisited

- $P[x := N]$ is only allowed if no free variable in N becomes bound after substitution.
- Otherwise: rename bound variables first.

$$(\lambda x.\lambda y.x\,y)\,(y\,y) =_\beta (\lambda y.x\,y)[x := y\,y]$$

$$(\;\equiv\; ??\;\; \lambda y.y\,y\,y\;\; \text{NO!!}\;\;) \equiv (\lambda z.x\,z)[x := y\,y] \equiv \lambda z.y\,y\,z$$

$$\begin{array}{lll} \mathbf{K}\,y\,z & \equiv & (\lambda x.(\lambda y.\,x))\,y\,z \\ & =_\beta ?? & (\lambda y.\,y)\,z \\ & =_\beta & z?? \end{array}$$

better:

$$\begin{array}{lll} \mathbf{K}\,y\,z & \equiv & (\lambda x.(\lambda y'.\,x))\,y\,z \\ & =_\beta & (\lambda y'.\,y)\,z \\ & =_\beta & y \text{ as it should.} \end{array}$$

Lambda Calculus subsumes Combinatory Logic

$$\begin{array}{rcllrcl} \mathbf{I} & \equiv & \lambda x.x & \Rightarrow & \mathbf{I}\, M & =_\beta & M \\ \mathbf{K} & \equiv & \lambda x\, y.x & \Rightarrow & \mathbf{K}\, M\, P & =_\beta & M \\ \mathbf{S} & \equiv & \lambda x\, y\, z.x\, z\, (y\, z) & \Rightarrow & \mathbf{S}\, M\, P\, Q & =_\beta & M\, Q\, (P\, Q) \end{array}$$

So we can define $(-)_\lambda : \mathbf{CL} \to \Lambda$ by

M	$(M)_\lambda$
$\mathbf{I}$	$\lambda x.x$
$\mathbf{K}$	$\lambda x\, y.x$
$\mathbf{S}$	$\lambda x\, y\, z.x\, z\, (y\, z)$
$P\, Q$	$(P)_\lambda\, (Q)_\lambda$

Satisfying

$$M =_{\mathbf{CL}} N \Rightarrow (M)_\lambda =_\beta (N)_\lambda$$

But not the other way around:
$\mathbf{S}\,\mathbf{K}\,\mathbf{I} \neq_{\mathbf{CL}} \mathbf{I}$, but in Λ we have $(\mathbf{S}\,\mathbf{K}\,\mathbf{I})_\lambda =_\beta (\mathbf{I})_\lambda$.

Also(!): Combinatory Logic subsumes Lambda Calculus

DEFINITION We define the embedding $(-)_{\mathbf{CL}} : \Lambda \to \mathbf{CL}$ by induction on terms as follows. (Where $[x]N$ is the abstraction defined for **CL**.)

M	$(M)_{\mathbf{CL}}$
x	x
$\lambda x.P$	$[x]\,(P)_{\mathbf{CL}}$
$P\,Q$	$(P)_{\mathbf{CL}}\,(Q)_{\mathbf{CL}}$

EXAMPLE

$$
\begin{array}{lllllll}
(\lambda x\,y.y)_{\mathbf{CL}} & = & [x]([y]y]) & = & \mathbf{K}\,\mathbf{I} & & \\
(\lambda x\,y.x)_{\mathbf{CL}} & = & [x]([y]x]) & = & [x](\mathbf{K}\,x) & = & \mathbf{S}\,(\mathbf{K}\,\mathbf{K})\,\mathbf{I}
\end{array}
$$

We have $M =_\beta N \iff (M)_{\mathbf{CL}} =_{\mathbf{CL}} (N)_{\mathbf{CL}}$

Reduction

The equations can be ordered into computation or reduction rules
One-step reduction $\rightarrow$; more-step reduction $\twoheadrightarrow$ (0, 1 or more steps).

Axiom	$(\lambda x.M)\,N \rightarrow M[x := N]$		
Rules for $\rightarrow$	$M \rightarrow N$	$\Rightarrow$	$M\,Z \rightarrow N\,Z$
	$M \rightarrow N$	$\Rightarrow$	$Z\,M \rightarrow Z\,N$
	$M \rightarrow N$	$\Rightarrow$	$\lambda x.M \rightarrow \lambda x.N$
Rules for $\twoheadrightarrow$	$M \twoheadrightarrow M$		
	$M \rightarrow N$	$\Rightarrow$	$M \twoheadrightarrow N$
	$M \twoheadrightarrow N \wedge N \twoheadrightarrow L$	$\Rightarrow$	$M \twoheadrightarrow L$

Examples:

$$\begin{array}{rcl} \mathbf{I}\,x & \rightarrow & x. \\ \mathbf{I}\,\mathbf{I}\,x & \rightarrow & \mathbf{I}\,x \\ & \rightarrow & x. \\ \mathbf{I}\,\mathbf{I}\,x & \twoheadrightarrow & x. \end{array}$$

Reduction Graph

Given $M \in \Lambda$, the graph of M, $\mathcal{G}(M)$, is

$$\{N \mid M \twoheadrightarrow N\}$$

with $\to$ as the edges and the 'reducts' of M as the vertices

For example let $P \equiv \lambda x.\mathbf{I}\,\mathbf{I}\,x\,x$ and $M \equiv P\,P$.
Then

$$\mathcal{G}(M) = \begin{tikzcd} P\,P \arrow[rr] & & \mathbf{I}\,\mathbf{I}\,P\,P \arrow[dl] \\ & \mathbf{I}\,P\,P \arrow[ul] & \end{tikzcd}$$

Fixed point theorem

THEOREM. For all $F \in \Lambda$ there is an $M \in \Lambda$ such that

$$F\,M =_\beta M$$

PROOF. Define $\mathbf{W} \equiv \lambda x.F\,(x\,x)$ and $M \equiv \mathbf{W}\,\mathbf{W}$. Then

$$\begin{array}{rcl} M & \equiv & \mathbf{W}\,\mathbf{W} \\ & \equiv & (\lambda x.F\,(x\,x))\,\mathbf{W} \\ & =_\beta & F\,(\mathbf{W}\,\mathbf{W}) \\ & \equiv & F\,M. \quad \Box \end{array}$$

COROLLARY. For any 'context' $C[\vec{x}, m]$ there exists a M such that

$$M\,\vec{P} =_\beta C[\vec{P}, M] \text{ for all terms } \vec{P}$$

PROOF. M can be taken the fixed point of $\lambda m\,\vec{x}.C[\vec{x}, m]$.
Then $M\,\vec{P} =_\beta (\lambda m\,\vec{x}.C[\vec{x}, m])\,M\,\vec{P} =_\beta C[\vec{P}, M]$. $\Box$

Using the Fixed Point Theorem

THEOREM. There is a Fixed Point Combinator $\mathbf{Y}$, that produces a fixed point for every term:

$$\mathbf{Y}\,F =_\beta F\,(\mathbf{Y}\,F) \quad \text{for all } F \in \Lambda.$$

PROOF. We have seen that, defining $\mathbf{W} \equiv \lambda x.F\,(x\,x)$, we get $M \equiv \mathbf{W}\,\mathbf{W}$ as a fixed point of F. So the following term is a fixed point combinator:

$$\mathbf{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)). \quad \square$$

Examples: We can construct terms $\mathbf{L}$, $\mathbf{O}$, $\mathbf{P}$ such that

$$\begin{array}{rcll} \mathbf{L} & =_\beta & \mathbf{L}\,\mathbf{L} & \text{take } \mathbf{L} \equiv \mathbf{Y}\,\mathbf{D}; \\ \mathbf{O}\,x & =_\beta & \mathbf{O} & \text{take } \mathbf{O} \equiv \mathbf{Y}\,\mathbf{K}; \\ \mathbf{P} & =_\beta & \mathbf{P}\,x. & \end{array}$$

Natural numbers and arithmetic in λ-calculus

The natural numbers are given by $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$
NOTATION For terms $F, A \in \Lambda$ and $n \in \mathbb{N}$, define $F^n\, A$ as follows:

$$\begin{array}{rcl} F^0\, A & := & A, \\ F^{n+1}\, A & := & F(F^n\, A) \end{array}$$

Thus $F^2\, A = F(F\, A)$ and $F^3\, A = F(F(F\, A))$.
DEFINITION (i) The Church numerals are $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \ldots$, with

$$\mathbf{c}_n := \lambda f\, x.f^n\, x.$$

(ii) A function $f : \mathbb{N} \to \mathbb{N}$ is called λ-definable if there is a term $F \in \Lambda$ such that for all $n \in \mathbb{N}$ one has

$$F\, \mathbf{c}_n =_\beta \mathbf{c}_{f(n)}.$$

Some representable functions

DEFINE

$$\begin{array}{rcl} A_+ & := & \lambda n\, m.\lambda f\, x.n\, f\, (m\, f\, x) \\ A_* & := & \lambda n\, m.\lambda f\, x.n\, (m\, f)\, x \\ A_{\text{exp}} & := & \lambda n\, m.\lambda f\, x.m\, n\, f\, x \end{array}$$

These functions λ-define addition, multiplication, and exponentiation. This means that we claim that the following holds:

$$\begin{array}{rcl} A_+\, \mathbf{c}_n\, \mathbf{c}_m & =_\beta & \mathbf{c}_{n+m} \\ A_*\, \mathbf{c}_n\, \mathbf{c}_m & =_\beta & \mathbf{c}_{n*m} \\ A_{\text{exp}}\, \mathbf{c}_n\, \mathbf{c}_m & =_\beta & \mathbf{c}_{m^n} \end{array}$$

We verify this only for A_+:

$$A_+\mathbf{c}_n\mathbf{c}_m =_\beta \lambda fx.\mathbf{c}_n f(\mathbf{c}_m fx) =_\beta \lambda fx.f^n(f^m x) \equiv \lambda fx.f^{n+m}x \equiv \mathbf{c}_{n+m}.$$

COROLLARY The function $f : \mathbb{N} \to \mathbb{N}$, with $f(n) = (n+2)^3$, is λ-definable

Two examples of data types: natural numbers and trees

Natural numbers:

`Nat := zero | suc Nat`

Binary Trees:

`Tree := leaf | join Tree Tree`

Equivalently, as a context-free grammar

$$\begin{array}{rcl} \texttt{Nat} & \to & \texttt{z} \mid (\texttt{s Nat}) \\ \texttt{Tree} & \to & \texttt{l} \mid (\texttt{j Tree Tree}) \end{array}$$

For Nat, we know what belongs to it:

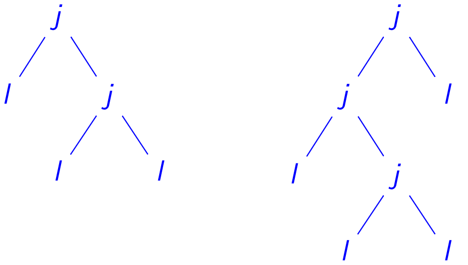
$$\texttt{Nat} = \{\texttt{z}, (\texttt{sz}), (\texttt{s(sz)}), (\texttt{s(s(s z))}), \ldots\} = \{\texttt{s}^n\texttt{z} \mid n \in \mathbb{N}\}$$

Binary Trees

`Tree  :=  l | (j Tree Tree)`

Examples of elements of `Tree`

`(jl(jll))` and `(j(jl(jll))l)`



Translating data into lambda terms (Böhm-Berarducci)

For $\mathtt{Nat}$: $\quad \mathtt{t} \mapsto \ulcorner \mathtt{t} \urcorner := \lambda s\, z.t$

For example

$$\ulcorner (\mathtt{s}(\mathtt{s}(\mathtt{sz}))) \urcorner = \lambda s\, z.(s(s(s\, z))) = \mathbf{c}_3$$

So for $\mathtt{Nat}$, the encoding gives us simply the Church numerals:

$$\ulcorner n \urcorner = \mathbf{c}_n.$$

For $\mathtt{Tree}$: $\quad \mathtt{t} \mapsto \ulcorner \mathtt{t} \urcorner := \lambda j\, l.t$

For example

$$\ulcorner (\mathtt{jl}(\mathtt{jll})) \urcorner = \lambda j\, \ell.(j\, \ell\, (j\, \ell\, \ell))$$

Basically $\ulcorner \mathtt{t} \urcorner$ represents $\mathtt{t}$ iff $\ulcorner \mathtt{t} \urcorner\, j\, \ell =_\beta t$ (where $\mathtt{l}$ is replaced by ℓ and $\mathtt{j}$ by j).

Operating on data after representing them

For Nat we can operate on the codes to represent functions:

$$\begin{aligned} A_+ \ulcorner \mathtt{n} \urcorner \ulcorner \mathtt{m} \urcorner &=_\beta \ulcorner \mathtt{n} + \mathtt{m} \urcorner \\ A_* \ulcorner \mathtt{n} \urcorner \ulcorner \mathtt{m} \urcorner &=_\beta \ulcorner \mathtt{n} \times \mathtt{m} \urcorner \end{aligned}$$

Definition. The λ-term F λ-defines the function $f : \mathbb{N} \to \mathbb{N}$ if

$$F \ulcorner \mathtt{n} \urcorner =_\beta \ulcorner \mathtt{f(n)} \urcorner \qquad \text{for all } n \in \mathbb{N}$$

Or: this *diagram commutes*:

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathtt{f}"] \arrow[d, "\ulcorner \_ \urcorner"'] & \mathbb{N} \arrow[d, "\ulcorner \_ \urcorner"] \\ \Lambda \arrow[r, "F"] & \Lambda \end{tikzcd}$$

For example addition:

$$\begin{tikzcd} \mathbb{N} \times \mathbb{N} \arrow[r, "+"] \arrow[d, "{\ulcorner \_ \urcorner, \ulcorner \_ \urcorner}"'] & \mathbb{N} \arrow[d, "\ulcorner \_ \urcorner"] \\ \Lambda \times \Lambda \arrow[r, "F"] & \Lambda \end{tikzcd}$$

Church-Turing Thesis

CHURCH-TURING THESIS
All functions $\mathtt{f} : \mathbb{N} \to \mathbb{N}$ that are computable are also λ-definable.

NB. This covers both "human computable" and "machine computable".

NB. This is not a theorem that can be proven. But it could be disproved!

Then Church went on constructing a function that is not λ-defiable hence by his thesis, not (human/machine) computable.
Turing did the same for his computational model, "Turing machines".

Functions on trees

Define on Trees the operation of mirroring:

$$\begin{array}{rcl} \texttt{Mirror (l)} & = & \texttt{l} \\ \texttt{Mirror (j } \texttt{t}_1\ \texttt{t}_2\texttt{)} & = & \texttt{j (Mirror } \texttt{t}_2\texttt{) (Mirror } \texttt{t}_1\texttt{)} \end{array}$$

We will construct a λ-term A_M such that

$$A_M \ulcorner \texttt{t} \urcorner = \ulcorner \texttt{Mirror(t)} \urcorner$$

Representing the basic operations “leaf” and “join”

LEMMA. The λ-terms

$$\begin{array}{rcl} L & := & \lambda j\,\ell.\ell \\ J & := & \lambda t_1\,t_2.\lambda j\,\ell.j(t_1\,j\,\ell)\,(t_2\,j\,\ell) \end{array}$$

define “leaf” and “join” on `Tree`. For L that's immediate.
For J, that means:

$$J\,\ulcorner \mathtt{t}_1 \urcorner\,\ulcorner \mathtt{t}_2 \urcorner = \ulcorner \mathtt{j}(\mathtt{t}_1, \mathtt{t}_2) \urcorner \qquad (1)$$

for all $\mathtt{t}_1$, $\mathtt{t}_2$: `Tree`.

PROOF.

$$\begin{array}{rcl} J\ulcorner \mathtt{t}_1 \urcorner\ulcorner \mathtt{t}_2 \urcorner & = & (\lambda t_1\,t_2.\lambda j\,\ell.j(t_1\,j\,\ell)\,(t_2\,j\,\ell))\,\ulcorner \mathtt{t}_1 \urcorner\,\ulcorner \mathtt{t}_2 \urcorner \\ & = & \lambda j\,\ell.j\,(\ulcorner \mathtt{t}_1 \urcorner\,j\,\ell)\,(\ulcorner \mathtt{t}_2 \urcorner\,j\,\ell) \\ & = & \lambda j\,\ell.j\,t_1\,t_2 \\ & = & \ulcorner \mathtt{j}\,\mathtt{t}_1\,\mathtt{t}_2 \urcorner. \end{array}$$

∎

Representing functions on $\mathtt{Tree}$ in Λ

Suppose function $\mathtt{f}$ on $\mathtt{Tree}$ is defined with the following recursion scheme (where $\mathtt{a}$ and $\mathtt{h}$ are given).

$$\begin{array}{rcl} \mathtt{f}\,\mathtt{leaf} & := & \mathtt{a} \\ \mathtt{f}\,(\mathtt{join}\,\mathtt{t}_1\,\mathtt{t}_2) & := & \mathtt{h}\,(\mathtt{f}\,\mathtt{t}_1)\,(\mathtt{f}\,\mathtt{t}_2) \end{array}$$

Lemma If A defines $\mathtt{a}$ and H defines $\mathtt{h}$ in Λ, then F defines $\mathtt{f}$, with:

$$F := \lambda t.t\,H\,A$$

Proof. We show that $\ulcorner\mathtt{t}\urcorner\,H\,A = \ulcorner\mathtt{f}\mathtt{t}\urcorner$ for all $\ulcorner\mathtt{t}\urcorner$. The case for "leaf" is immediate. For "join":

$$\begin{array}{rcl} \ulcorner\mathtt{j}\,\mathtt{t}_1\,\mathtt{t}_2\urcorner\,H\,A & = & J\,\ulcorner\mathtt{t}_1\urcorner\,\ulcorner\mathtt{t}_2\urcorner\,H\,A \\ & = & (\lambda j\,\ell.j\,(\ulcorner\mathtt{t}_1\urcorner\,j\,\ell)\,(\ulcorner\mathtt{t}_2\urcorner\,j\,\ell))\,H\,A \\ & = & H\,(\ulcorner\mathtt{t}_1\urcorner\,H\,A)\,(\ulcorner\mathtt{t}_2\urcorner\,H\,A) \\ & = & H\,\ulcorner\mathtt{f}\,\mathtt{t}_1\urcorner\,\ulcorner\mathtt{f}\,\mathtt{t}_2\urcorner \text{(by Induction Hypothesis)} \\ & = & \ulcorner\mathtt{f}\,(\mathtt{j}\,\mathtt{t}_1\,\mathtt{t}_2)\urcorner \end{array}$$

Representing mirroring in Λ

The function `Mirror` is also defined by a recursion scheme over `Tree`:

$$\begin{array}{rcl} \texttt{Mirror leaf} & = & \texttt{leaf} \\ \texttt{Mirror (join } \texttt{t}_1\ \texttt{t}_2\texttt{)} & = & \texttt{join (Mirror } \texttt{t}_2\texttt{) (Mirror } \texttt{t}_1\texttt{)} \end{array}$$

So the "helping functions" are L (for `leaf`) and $\lambda a\, b. J\, b\, a$ (for $\mathtt{h}\,\mathtt{t}_1\,\mathtt{t}_2 = \mathtt{join}\,\mathtt{t}_2\,\mathtt{t}_1$).

Conclusion: `Mirror` is defined in Λ by

$$A_M = \lambda t. t\, L\, (\lambda a\, b. J\, b\, a).$$

Booleans

The type of booleans, `Bool`, contains just two constants, 'true' and 'false': `Bool := true | false`

These are represented in λ-calculus in the standard way:

$\mathtt{true} \mapsto \ulcorner \mathtt{t} \urcorner := \lambda \mathtt{t}\,\mathtt{f}.\mathtt{t}$

$\mathtt{false} \mapsto \ulcorner \mathtt{f} \urcorner := \lambda \mathtt{t}\,\mathtt{f}.\mathtt{f}$

These well-known terms ($\mathbf{K}$ and $\mathbf{K}_*$) are thus also called $\mathbf{T}$ and $\mathbf{F}$.

Some more λ-definable functions:

$$\begin{array}{rcl} \text{Neg} & := & \lambda b.b\,\mathbf{F}\,\mathbf{T} \\ \text{Zero} & := & \lambda n.n\,(\lambda x.\mathbf{F})\,\mathbf{T} \\ \text{ITE} & := & \lambda b\,x\,y.b\,x\,y \end{array}$$

- Neg defines negation on the booleans
- Zero defines the test-for-zero function from `Nat` to `Bool`
- ITE defines the if-then-else function on $\mathtt{Bool}\times\mathtt{Nat}\times\mathtt{Nat}$.
  We write if b then M else N for $\text{ITE}\,b\,M\,N$.

Using the fixed point combinator to λ-define functions

On `Nat`, we can also define the predecessor (which is remarkably tricky!) $p : \mathbb{N} \to \mathbb{N}$ satisfying $p(0) = 0$ and $p(n+1) = n$.
So, we have a λ-term Pred satisfying

$$\begin{aligned} \mathrm{Pred}\ulcorner 0 \urcorner &= \ulcorner 0 \urcorner \\ \mathrm{Pred}\ulcorner n+1 \urcorner &= \ulcorner n \urcorner \end{aligned}$$

Can we λ-define the faculty function $! : \mathbb{N} \to \mathbb{N}$??

$$n! = n * (n-1) * (n-2) * \ldots 2 * 1.$$

We are looking for a term Fac satisfying

$$\mathrm{Fac}\, n =_\beta \text{if } (\mathrm{Zero}\, n) \text{ then } \ulcorner 1 \urcorner \text{ else } (A_*\, n\, (\mathrm{Fac}(n-1)))$$

This we can solve by taking

$$\mathrm{Fac} := \mathbf{Y}(\lambda f\, n.\text{if } (\mathrm{Zero}\, n) \text{ then } \ulcorner 1 \urcorner \text{ else } (A_*\, n\, (f(\mathrm{Pred}\, n))))$$