University of Cambridge
Faculty of Mathematics

# Combinatorial Logic, or is it Really the Lambda Calculus?

Josie Smith
Prof Martin Hyland

## The Lambda Calculus

The $\lambda$ *calculus* is a formal language consisting of a set of variables $x, y, z$, etc.

Lambda terms: $M, N ::= x|(MN)|(\lambda x.M)$

$x$ is a *free* variable in terms such as $xy$ and $\lambda y.xy$, but is *bound* in terms such as $\lambda x.xy$ and $\lambda xy.xy$.

**$\alpha$-Equivalence:**
Substituting bound variables for other variables (similar to dummy variables). eg. $\lambda x.xy = \lambda z.zy$

**$\beta$-Equivalence:**
Evaluating a $\lambda$ term, eg. $(\lambda x.xy)z = zy$

**$\eta$-Equivalence:**
If 2 terms $\beta$-reduce to the same thing, they are $\eta$-equivalent. eg. $\lambda x.yx = y$

Rewriting of equals for equals in the lambda calculus is a (universal) form of computation. The $\lambda$ calculus is the basis for functional programming languages, and is a logic of equality between terms with terms closed under application and function abstraction.

## Combinatory Logic

An *applicative structure* $(A, \bullet)$ is just a set $A$, with an application $\bullet$. For this to be a *combinatory logic*, there must exist an $S$ and $K$ in $A$ such that:

$$\mathbf{Sabc} = \mathbf{ac(bc)}$$
$$\mathbf{Kab} = \mathbf{a}$$

## Aim

$L$ is an interpretation of the $\lambda$ calculus, where $L(n)$ is the set of terms with $n$ free variables. $C$, a combinatory algebra, is an interpretation of combinatory logic. $L(n-1)$ and $L(n)$ are isomorphic, via application on $x$ and $\lambda$-abstraction (taking $a \mapsto \lambda x.ax$).

$$L(0) \to L(1) \to L(2) \to L(3) \to ...$$
$$a \mapsto a \bullet x \quad \lambda x.bx \leftarrow\!\shortmid b$$

*Question:* For a given application $\bullet$, is there the structure of an interpretation of the $\lambda$-calculus on the algebraic theory $\{C(n)\}_{n\geq 0}$?
*Answer:* Iff $a \mapsto a \bullet z$ is an isomorphism.

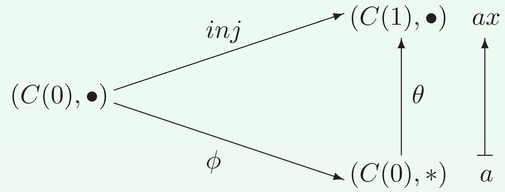
Hence for $C$ to be a model of L, we need isomorphims between

$$C(0) \to C(1) \to C(2) \to ...$$

My aim is to find equations between combinators which make this true.

## Finding Equations

Peter Freyd's idea: If $C(0)$ and $C(1)$ are isomorphic, there must be an applicative structure $(A, *)$ on $C(0)$ which corresponds to that of $C(1)$.




The arrows must indicate maps between combinatory algebras, so must behave nicely. Hence $\theta(a * b) = \theta(a) \bullet \theta(b)$, where $\theta$ is application on x, so $(a * b)x = ax(bx)$ (associating to the left). We then notice that $*$ is just $S$ from earlier, so we write $a * b = Sab$. This now means that $(C(0), S)$ is a combinatory algebra.

Now, for the diagram to be commutative, we want $\phi(a)x = a$. Notice that $\phi(a) = Ka$. For $K$ to be a nice mapping, $K(ab) = K(a) * K(b) = S(Ka)(Kb)$. Thus our first equation, which means that $K$ is a map of combinatory algebras, is

$$\mathbf{K(ab)} = \mathbf{S(Ka)(Kb)}$$

But as $(C(0), S)$ is a combinatory algebra just as $(C(0), \bullet)$ is, it must also have a "$K$" and "$S$", and these must be the images of the $K$ and $S$ in $(C(0), \bullet)$. Hence they are $KK$ and $KS$. Checking that these act (under operation by $S$, naturally!) in the correct way gives us 2 more equation:

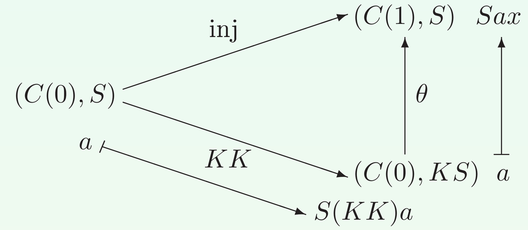
$$\mathbf{S(S(KK)a)b} = \mathbf{a} \qquad \text{correctness of } KK$$
$$\mathbf{S(S(S(KS)a)b)c} = \mathbf{S(Sac)(Sbc)} \qquad \text{correctness of } KS$$

Finally, for $\theta$ to be an isomorphism, we need to find a map that is its inverse. For $a \in C(0)$, the map $(C(1), \bullet) \to (C(0), S)$ is just $a \mapsto Ka$, as $inj$ is the constant injection. But for the new free variable $x$, we notice the $Ix = x$ (where $I = \lambda x.x = SKK$), so we map $x \mapsto I$. These define the map uniquely, so the composite of this inverse with $\theta$ must be the identity. Hence $a \mapsto ax \mapsto S(Ka)I$, so our fourth equation is

$$\mathbf{S(Ka)I} = \mathbf{a}$$

## Generalising

As $(C(0), S)$ is also a combinatory algebra with $KK$ and $KS$, we can draw a similar diagram, and repeat the process described above.




So $a \mapsto S(KK)a$, as the application we are now working under is S. It is important to remember that $K$ and $S$ are not only a function and an application, but also elements in $C(0)$.

Before, we had 4 equations, plus the definitions of $K$ and $S$. Similarly by imposing the "niceness" of the maps in the new diagram, we can get 4 more equations, which are not included as they are long complicated, and not particularly enlightening.

We can also map the new $K$ and $S$ to ones in $(C(0), KS)$, and get even more equations! This could go on indefinitely. Here are recursive relations to calculate the new $K$ and $S$, and 4 equations for each level. Lower case $k$ and $s$ are the "actual" terms in $C(0)$, whereas upper case $K$ and $S$ are the function and application respectively, written acting on a and b.

| | | |
|---|---|---|
| **Level 1** $(C(0), S_0 = \bullet)$ | $k_1 = K$ $K_1 = Ka$ | $s_1 = S$ $S_1 = Sab$ |
| **Level n** $(C(n), S_{n-1})$ | $k_n = S_{n-2}k_{n-1}k_{n-1}$ $K_n = S_{n-1}k_n a$ | $s_n = S_{n-2}k_{n-1}s_{n-1}$ $S_n = S_{n-1}(S_{n-1}s_n a)b$ |

Equations:
$S_n(K_{n+1}a)x = a$
$K_n(S_{n-1}ab) = S_n(K_n a)(K_n b)$
$S_n(S_{n+1}ab)x = S_n(S_n ax)(S_n bx)$
$S_n(K_n a)I = a$

## What Next?

Hence it is these 4 equations that are necessary (and sufficient) for $C$ to be a model of the $\lambda$ calculus. To get these same equations with free variables instead of $a, b$ and $c$ in $C(0)$, just repeat the above process, but starting in $C(3)$. Now you can substitute any free variable for any $\lambda$ term.
Comparing our equations with Curry's:

**Our Equations**

$K(xy) = S(Kx)(Ky)$
$S(S(KK)x)y = x$
$S(S(S(KS)x)y)z = S(Sxz)(Syz)$
$S(Kx)I = x$

**Curry's Equations**

$S(S(KS)(S(KK)(S(KS)K)))(KK) = S(KK)$
$S(KS)(S(KK)) = S(KK)(S(S(KS)K)(K(SKK)))$
$S(K(S(KS)))(S(KS)(S(KS)))$
$= S(S(KS)(S(KK)(S(KS)(S(K(S(KS)))S))))(KS)$
$S(S(KS)K)(K(SKK)) = SKK$

These are actually the same equations; by finding isomorphic equations in $C(0)$ (and simplifying using our 4th equation) we deduce Curry's equations from ours. Similarly, $\lambda$-abstracting Curry's equations and $\beta\eta$-reducing gives our equations.

It is an interesting question to understand what happens if you have some equations and not others. What happens if you have all equations save the 4th is worth thinking about: the 4th is a kind of uniqueness typical of universal properties in category theory and that does not seem computationally essential.

It is these relationships between the combinators that make a combinatory algebra into a $\lambda(\beta\eta)$ calculus. However, when trying to find similar equations to make a combinatory algebra into a $\lambda(\beta)$ calculus without the $\eta$-equivalence rule, there are a few more. Instead of isomorphisms we have retracts and these, unlike the isomorphisms, are not uniquely determined by the structure given, so the situation looks more subtle.