Extensional equalities in combinatory logic and lambda calculus

Preliminary report

Rémi Germe, supervised by Samuel Mimram[1]

January 2025

Introduction

This introduction aims to provide the reader with a general idea of what to expect in this report, and also to develop some intuitions on the presented results - which are very formal.

Lambda-calculus and combinatory logic are two very simple formalisms of computations. They are defined inductively, which provides *syntactical* (or *intensional*) equalities : if two terms (which are intuitively functions) have the exact same definition, they are equal. However, it is possible that two different functions may always yield the same results. For example, think of two different sort algorithms operating on arrays.

If two functions intuitively do the same job, we say they are *extensionaly* equal. In fact, it is possible to derive *extensional* equalities for both lambda calculus and combinatory logic from their respective *intensional* equalities. This is straightforward for lambda calculus by considering natural operations on terms called reductions.

However for combinatory logic, things are trickier. The goal of this research project was to investigate ways to derive extensional equality for combinatory logic. In fact it is possible to deduce extensional equality from a finite number of some strange-looking axioms at first sight. My job was to understand where these axioms come from, and to find a similar set of axioms for a variation of combinatory logic (see Section 3).

Finally, lambda calculus and combinatory logic are somehow equally expressive, although going from one to the other isn't trivial. I started by investigating this equivalence when using a more traditional definition of combinatory logic than used in [1] (see Section 2).

1. State of the art

This section provides with definitions of both worlds studied (λ-calculus and combinatory logic), and presents the difference between two notions of equality : *intensional* and *extensional* equalities. This section heavily relies on [1] and [2] and even sometimes use examples presented in these books. Unfortunately this section is very formal.

1.1. Lambda-calculus

1.1.1. Definitions

Definition 2.1. (λ-terms)
Given a set of *variables* V, λ-terms are defined as follows :

- **variable** : $x \in V$ is a λ-term
- **application** : if M and N are λ-terms, then MN is a λ-term
- **abstraction** : if x is a variable and M is a λ-term, then $\lambda x.M$ is a λ-term

[1]Professor at École polytechnique, https://www.lix.polytechnique.fr/Labo/Samuel.Mimram/

Intuitively, abstractions correspond to functions ($\lambda x.M$ is the function that associates M to x) and applications correspond to function evaluations (MN is passing N as argument to M).

Remark 2.2. (Association on the left convention) By convention, when there are multiple applications in a row, parentheses may be omitted and the rule of association on the left is used. Here's an example with four terms $M, N, P, Q : MNPQ \equiv ((MN)P)Q$.

Remark 2.3. (Functions of multiple variables) Also by convention, functions of multiple variables are a notation for nested functions and are thus allowed. For example, a function of three variables is a shorthand such as : $\lambda xyz.M \equiv \lambda x.(\lambda y.(\lambda z.M))$.

Example 2.4. Given $V = \{x, y, z\}$:
- $\lambda x.x$, $\lambda x.y$, $\lambda xz.zx$, $\lambda x.((\lambda x.x)x)$, z, xyz are valid λ-terms - note that variables are not necessarily bounded and that the scope of a specific variable is important
- $\lambda t.x$ is not a valid λ-term because $t \notin V$

Definition 2.5. (Scope of variables)
All variables appearing inside a λ-term M are either *free* if they are not introduced by an abstraction, or *bound* otherwise. We won't give a formal definition of free and bound variables as it is unnecessary technical and intuitively uninteresting.

The set of free variables of M is denoted by $\mathrm{FV}(M)$. Note that alternatively we can construct inductively this set :

– $\mathrm{FV}(x) = \{x\}$ for all variable x

– $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$ for all terms M, N

– $\mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) \setminus \{x\}$ for all variable x and term M

Definition 2.6. (Substitution)
For λ-terms M, N and a variable x, $[M/x]N$ denotes the term obtained after replacing all free occurences of x in N by M. The substitution is defined inductively :

– $[M/x]x \equiv M$

– $[M/x]y \equiv y$ for all variables $y \neq x$

– $[M/x](NP) \equiv ([M/x]N)([M/x]P)$

– $[M/x](\lambda x.N) \equiv \lambda x.N$

– $[M/x](\lambda y.N) \equiv \lambda y.[M/x]N$ if $y \notin \mathrm{FV}(M)$

– $[M/x](\lambda y.N) \equiv \lambda z.[M/x][y/z]N$ if $y \in \mathrm{FV}(M)$ with z a variable such as $z \notin \mathrm{FV}(MN)$

1.1.2. Reductions : α, β, η

These reductions are introduced to express very natural ideas. For example, renaming a bounded variable do not modify what a function does. We want to have $\lambda x.x \simeq \lambda z.z$ for some equivalence relation $\simeq$, as both of these λ-terms represent the identity.

Definition 2.7. (α-reduction)
To define the equivalence relation we're searching for, we want to be able to replace every instance of $\lambda x.M$ by $\lambda y.[y/x]M$, which is called an *α-conversion*. If two terms M and N *α-converts* to each other - that is we can transform M to N after a finite number of α-conversion, we write $M \underset{\alpha}{=} N$.

Another very natural idea is to actually compute functions inside λ-terms. For example, we'd like to have $(\lambda x.x)(yz) \to yz$.

Definition 2.8. (β-reduction)
β-reduction corresponds to the evaluation, we can use the following pattern to β-reduce terms :

$$(\lambda x.U)M \rhd_\beta [M/x]U$$

If we can transform a term M into a term N by a finite number of β-reductions or reversed β -reductions, that is $M \rhd_\beta M' \ {}_\beta\lhd M'' \rhd_\beta ... \ {}_\beta\lhd N$, we write $M \underset{\beta}{=} N$.

Definition 2.9. (η-reduction)
η-reduction is a bit less intuitive than β-reduction but is required for further results. For a term N and a variable x, we can η-reduce terms using the following pattern :

$$\lambda x.Nx \rhd_\eta N \text{ if } x \notin \mathrm{FV}(N)$$

Similarly to β-reductions, we derive the equality relation '$\cdot \underset{\eta}{=} \cdot$'.

Theorem 2.10. ($\beta\eta$ and normal forms)
Using both β and η-reductions leads to an equality relation '$\cdot \underset{\beta\eta}{=} \cdot$'. Some terms admit a normal form, which is a reduction which cannot be reduced anymore. In most cases, this intuitively corresponds to having fully computed the term.

Remark 2.11. (Discussion on normal forms) All terms do not accept a normal form. For example consider $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$, we can β-reduce Ω infinitely : $\Omega \rhd_\beta \Omega \rhd_\beta ...$ and this the only way to β-reduce this term. It is even possible to have a term which keeps growing when we β-reduce it, let $L \equiv (\lambda x.xxy)(\lambda x.xxy)$, so $L \rhd_\beta Ly \rhd_\beta Lyy \rhd_\beta ...$ We won't mind those cases from now on but reduction isn't trivial at all.

1.1.3. Extensional equality

Definition 2.12. (Extensional equality)
Two terms are *extensionally equal* if they produce the same result when the the arguments are provided to these terms. Formally, extensional equality can be constructed from the syntactic equality (called *intensional*) by adding the following rule ζ :

$$(\zeta)\ \frac{Mx = Nx}{M = N} \text{ if } x \notin \mathrm{FV}(MN)$$

The rule reads as "if $Mx = Nx$ for x such that $x \notin \mathrm{FV}(MN)$, deduce that $M = N$". The '=' sign here stands for extensional equality.

Remark 2.13. (Intensionality vs extensionality) This concept of extensionality can be found in a lot of mathematical spaces. For example in analysis, extensional equality is the standard way to go when talking of equality : given two real functions f and g, if for all $x \in \mathbb{R}$ we have $f(x) = g(x)$, then we say $f = g$. However, in computer-science-related fields, it is more convenient to use intensional equality. For example, merge sort and insertion sort both have the same effects but they differ in complexity ($\mathcal{O}(n \log n)$ vs $\mathcal{O}(n^2)$), so they are somehow extensionally equal but intensional equality is more relevant here.

Theorem 2.14. ($\alpha\beta\eta$-reduction is extensional)
$\underset{\zeta}{=}$ and $\underset{\alpha\beta\eta}{=}$ determine the same equality relation. That is, applying α, β and η reductions allow to determine if two terms are extensionally equal.

Using α and $\beta\eta$-reduction to normal form enables to check for extensional equality in practice. These rules are pretty simple to implement in Ocaml for example.

1.2. Combinatory logic

1.2.1. Definitions

Combinatory logic is an alternative to λ-calculus developed by Curry [3], and works without variables. They express things differently but they are somehow as expressive as λ-calculus.

Definition 2.15. (Combinatory logic (CL))
Combinators are defined inductively using the following constructors :

- **basic combinator** : **S**, **K**, **I** are combinators.
- **application** : if U and V are combinators, then MN is a combinator

The convention of association to the left is also used for CL-terms.

Definition 2.16. (Weak reduction)
Similarly to β-reduction introduced for λ-calculus, *weak reduction* is the way to compute CL expressions. We only have three basic combinators, which weakly reduce as follows :

$-$ $\mathbf{I}U \quad \triangleright_w U$ identity

$-$ $\mathbf{K}UV \quad \triangleright_w U$ first projection

$-$ $\mathbf{S}UVZ \triangleright_w UZ(VZ)$ distribution of a variable

Similarly to λ-calculus, if a CL-term U can be transformed into a term V in a finite number of steps using weak reduction operations or reversed weak reduction operations, we write $U \underset{w}{=} V$.

Example 2.17. (Expressiveness of CL) Although there are only three basic combinators, CL is very expressive. For example, it is possible to define a function of two arguments which swap them. In λ-calculus, that could simply be $\lambda xy.yx$, whereas in CL $\mathbf{S}(\mathbf{K}(\mathbf{SI}))(\mathbf{S}(\mathbf{KK})\mathbf{I})$ would work. Indeed :

$$\begin{aligned}
\mathbf{S}\underbrace{(\mathbf{K}(\mathbf{SI}))}_{U}\underbrace{(\mathbf{S}(\mathbf{KK})\mathbf{I})}_{V}\underbrace{x}_{Z}y &\triangleright_w \mathbf{K}\underbrace{(\mathbf{SI})}_{U}\underbrace{x}_{V}\left(\mathbf{S}\underbrace{(\mathbf{KK})}_{U'}\underbrace{\mathbf{I}}_{V'}\underbrace{x}_{Z'}\right)y \\
&\triangleright_w \mathbf{SI}\left(\mathbf{K}\underbrace{\mathbf{K}}_{U}\underbrace{x}_{V}\left(\mathbf{I}\underbrace{x}_{U'}\right)\right)y \\
&\triangleright_w \underbrace{\mathbf{S}}_{U}\underbrace{\mathbf{I}(\mathbf{K}x)}_{V}\underbrace{y}_{Z} \\
&\triangleright_w \underbrace{\mathbf{I}y}_{U}\left(\mathbf{K}\underbrace{x}_{U'}\underbrace{y}_{V'}\right) \\
&\triangleright_w yx \quad \text{we successfully swapped the arguments !}
\end{aligned}$$

Although it is possible to find CL-terms for functions we want, it is extremely convenient to develop an abstraction algorithm to automatize the procedure.

Definition 2.18. (CL abstractions)
In order to define abstractions, a set of variables V is introduced like in λ-calculus. CL-terms are extended to allow variables. Then, for a variable x and a CL-term U, $[x].U$ is a notation denoting the result of the following abstraction procedure :

- $[x].x \equiv \mathbf{I}$
- $[x].M \equiv \mathbf{K}U$ if $x \notin \mathrm{FV}(U)$
- $[x].UV \equiv \mathbf{S}([x].U)([x].V)$ if $x \in \mathrm{FV}(UV)$

As done in λ-calculus, abstractions of multiple variables $[x, y, z].U$ are shortcuts for nested abstractions and are thus allowed.

Remark 2.19. (Definition given in [1]) The definition in [1] includes another case with a higher priority than the one using $\mathbf{S}$: $[x].Ux \equiv U$ if $x \notin \mathrm{FV}(U)$. This case corresponds to η-reduction and leads to way shorter abstraction terms. However, the definition given above is more traditional, and slightly modifies proofs done in [1].

Example 2.20. In fact, the previous example CL-term can be obtained using this procedure.

$$\begin{aligned}
[x, y].yx &\equiv [x].([y].yx) \\
&\equiv [x].(\mathbf{S}([y].y)([y].x)) \\
&\equiv [x].(\mathbf{SI}(\mathbf{K}x)) \\
&\equiv \mathbf{S}([x].\mathbf{SI})([x].\mathbf{K}x) \\
&\equiv \mathbf{S}(\mathbf{K}(\mathbf{SI}))(\mathbf{S}(\mathbf{KK})\mathbf{I})
\end{aligned}$$

1.2.2. Extensional equality

In this section, the '=' sign will denote extensional equality. Similarly to λ-calculus, extensional equality can be constructed by adding the following rule for all CL-terms U and V :

$$(\zeta)\ \frac{Ux = Vx}{U = V} \text{ if } x \notin \mathrm{FV}(UV)$$

However weak reduction does not suffice to derive (ζ). That is because of a fundamental property which stands in λ-calculus but doesn't in combinatory logic :

$$(\xi)\ \frac{U = V}{[x].U = [x].V}$$

Indeed, in λ-calculus, this is obviously true because if $M = N$, then $\lambda x.M = \lambda x.N$. However, abstraction is not a native object of combinatory logic, it is a construction rather than a constructor.

For example, let's consider x and $\mathbf{I}x$.
- $x \underset{w}{=} \mathbf{I}x$ as $\mathbf{I}x \rhd_w x$
- on one hand $[x].x \equiv \mathbf{I}$
- on the other hand $[x].\mathbf{I}x \equiv \mathbf{S}(\mathbf{KI})\mathbf{I}$
- and $\mathbf{I} \underset{w}{\neq} \mathbf{S}(\mathbf{KI})\mathbf{I}$ *i.e.* $[x].x \underset{w}{\neq} [x].\mathbf{I}x$

There are multiple options to obtain extensionality :
1. start by considering $\underset{w}{=}$ instead of $\equiv$
2. - add (ζ) rule in the definition of the extensional equality (thus getting $\underset{\zeta}{=}$)
   - or add both (ξ) and (η) rules (thus getting $\underset{\xi\eta}{=}$)
   - or add a finite set of well-chosen closed axioms (thus getting $\underset{\text{ext}}{=}$)

Adding rules (ζ) or (ξ) and (η) mean an infinite number of axioms for the extensional equality, as these rules are valid for all CL-terms U and V. Conversely, Curry found that only a small set of closed axioms (*i.e.* they do not contain variables but only basic combinators) would suffice.

Definition 2.21. (Extensionality axioms for CL)

***E-ax* 1:** $\mathsf{S}\,(\mathsf{S}\,(\mathsf{KS})\,(\mathsf{S}(\mathsf{KK})(\mathsf{S}(\mathsf{KS})\mathsf{K})))\,(\mathsf{KK}) = \mathsf{S}(\mathsf{KK})$;

***E-ax* 2:** $\mathsf{S}\,(\mathsf{S}(\mathsf{KS})\mathsf{K})\,(\mathsf{KI}) = \mathsf{I}$;

***E-ax* 3:** $\mathsf{S}(\mathsf{KI}) = \mathsf{I}$;

***E-ax* 4:** $\mathsf{S}(\mathsf{KS})(\mathsf{S}(\mathsf{KK})) = \mathsf{K}$;

***E-ax* 5:** $\mathsf{S}\,(\mathsf{K}(\mathsf{S}(\mathsf{KS})))\,(\mathsf{S}(\mathsf{KS})(\mathsf{S}(\mathsf{KS}))) = \mathsf{S}\,(\mathsf{S}\,(\mathsf{KS})\,(\mathsf{S}(\mathsf{KK})(\mathsf{S}(\mathsf{KS})(\mathsf{S}(\mathsf{K}(\mathsf{S}(\mathsf{KS})))\mathsf{S}))))\,(\mathsf{KS})$.

Figure 1: Screenshot from [1]

Because I considered a definition of abstractions not using η-reduction, we need to add an extra axiom (see Section 2) :

(ax-η) $\quad \mathbf{I} \underset{\text{ext}}{=} \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})\mathbf{I}))(\mathbf{KI})$

These axioms define an equality relation $\underset{\text{ext}}{=}$.

Examples of computations of these axioms will be presented in Section 2 and Section 3. The idea is to choose axioms which allow us to derive (ξ) and (η).

Theorem 2.22. (Extensionality 1)
$\underset{\zeta}{=}$ and $\underset{\xi\eta}{=}$ determine the same equality relation (true in general).

Theorem 2.23. (Extensionality 2)
$\underset{\xi\eta}{=}$ and $\underset{\text{ext}}{=}$ determine the same equality relation (because we chose the axioms to get this precise result).

Thus, $\underset{\text{ext}}{=}$ and $\underset{\zeta}{=}$ determine the same extensional equality relation.

2. Check the equivalence between λ-calculus and CL

As mentioned in the definition of CL-abstractions, the main reference [1] used a different definition including η-reduction. The definition I used do not include it, so I checked the whole proof of equivalence between λ-calculus and CL from the very start to familiarize with objects presented earlier.

Turns out almost all arguments remain valid - or lowered results still strong enough might be obtained with very little work. Maybe todo for the final report : put in an appendix section the whole review of the proof / the details which need adjustments. There is however one thing we need to make sure : our extensional equality in CL must implement η-reduction.

That is, we need $[x].Ux \underset{\text{ext}}{=} U$ for all CL-term U for which $x \notin \mathrm{FV}(U)$, *i.e.* we need $[x].Ux \equiv \mathbf{S}(\mathbf{K}U)\mathbf{I}$. However, we only want a finite number of axioms. A solution I came up with consists in closing the term using an abstraction : we ask for $[U, x].Ux \underset{\text{ext}}{=} [U].U$.

$$\begin{aligned}
[U, x].Ux &\equiv [U].(\mathbf{S}(\mathbf{K}U)\mathbf{I}) \\
&\equiv \mathbf{S}([U].\mathbf{S}(\mathbf{K}U))(\mathbf{KI}) \\
&\equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})([U].\mathbf{K}U))(\mathbf{KI}) \\
&\equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})\mathbf{I}))(\mathbf{KI}) \text{ on one hand}
\end{aligned}$$

$$[U].U \equiv \mathbf{I} \text{ on the other hand}$$

So we add the following axiom of extensionality in the theory :

(ax-η) $\mathbf{I} =_{\text{ext}} \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})\mathbf{I}))(\mathbf{KI})$

For example, we have $\mathbf{I}x =_{\text{ext}} x$ (because $\mathbf{I}x =_{w} x$), so we need to have $[x].\mathbf{I}x =_{\text{ext}} [x].x$. With a definition of abstractions including η-reduction, this would be obviously true. Yet, our definition doesn't use η-reduction and we need to make sure the relation is verified.

$$\begin{aligned}[x].\mathbf{I}x &\equiv \mathbf{S}(\mathbf{KI})\mathbf{I} \quad \text{(would be } \mathbf{I} \text{ directly if } \eta\text{-reduction in definition)}\\ [x].x &\equiv \mathbf{I}\end{aligned}$$

So we need to derive $\mathbf{S}(\mathbf{KI})\mathbf{I} =_{\text{ext}} \mathbf{I}$. To do that, let's apply $\mathbf{I}$ to (ax-η) :

$$\begin{aligned}\mathbf{II} &=_{\text{ext}} \mathbf{I} \text{ by weak reduction on one hand}\\ &=_{\text{ext}} ([U].(\mathbf{S}(\mathbf{K}U)\mathbf{I}))\mathbf{I} \text{ by (ax-}\eta\text{) on the other hand}\\ &=_{\text{ext}} \mathbf{S}(\mathbf{KI})\mathbf{I} \text{ by evaluation theorem}\end{aligned}$$

Thus, the (ax-η) can successfully be used to replace η-reduction.

3. Extensionality axioms for linear combinatory logic

Then, the goal was to use the same technique to determine axioms in another situation of combinatory logic.

In this section, a restriction of λ-calculus and combinatory logic is considered, leading to different basic combinators for combinatory logic. We will see a method to retrieve some extensionality axioms.

Definition 4.1. (Linear λ-calculus)
Linear λ-calculus is a restriction of λ-calculus in which each bound variable is used exactly once. For all abstractions of the form $\lambda x.M$, x is used exactly once as a free variable in M.

Let's give a slightly different definition of combinatory logic in this linear context.

Definition 4.2. (Linear CL)
Instead of using $\mathbf{S}$, $\mathbf{K}$, $\mathbf{I}$ as basic combinators, we will now use $\mathbf{B}$, $\mathbf{C}$, $\mathbf{I}$ such as :
- $\mathbf{B}UVZ \rhd_w U(VZ)$ (passing argument to the second function)
- $\mathbf{C}UVZ \rhd_w UZV$ (passing argument to the first function)
- $\mathbf{I}U \rhd_w U$ (still the identity)

Definition 4.3. (Abstraction for linear CL)
The definition of abstraction for linear CL needs to be adapted too :
- $[x].x = \mathbf{I}$
- $[x].UV = \mathbf{C}([x].U)V$ if $x \in \text{FV}(U)$ - so $x \notin \text{FV}(V)$
- $[x].UV = \mathbf{B}U([x].V)$ if $x \in \text{FV}(V)$ - so $x \notin \text{FV}(U)$

This definition is well-defined because of the linear context.

Intuitively, each of the extensionality axioms are related to a combinator $\mathbf{B}$, $\mathbf{C}$ or $\mathbf{I}$ or η-reduction. We need to make sure that each of these elements will pass through (ξ).

Example 4.4. (Finding extensionality axiom associated with I)
We have $\mathbf{I}U =_{\text{ext}} U$ so we need to have $[x].\mathbf{I}U =_{\text{ext}} [x].U$.

$$
\begin{aligned}
[x].\mathbf{I}U &\equiv \mathbf{BI}([x].U) \text{ by def of abstractions} \\
&\underset{w}{=} ([v].\mathbf{BI}v)([x].U) \text{ by evaluation theorem} \\
&\underset{\text{ext}}{=} ([v].v)([x].U) \text{ this is what we want : this step is the axiom} \\
&\underset{w}{=} [x].U \text{ by evaluation theorem}
\end{aligned}
$$

So the axiom we need is :

$$[v].\mathbf{BI}v \underset{\text{ext}}{=} [v].v$$

or by replacing each term by its expanded form

$$\mathbf{B}(\mathbf{BI})\mathbf{I} \underset{\text{ext}}{=} \mathbf{I}$$

Results

It is possible to determine 8 axioms in total :

- 1 for **I**
- 3 for **B** : depending in which of 3 arguments of **B** the variable of the abstraction is located. In the example above, the variable x of the abstraction $[x].\mathbf{I}U$ is necessarily located in U, but with **B** we are considering $[x].\mathbf{B}UVZ$ and x could be in U (1 axiom), V (1 axiom) or Z (1 axiom)
- 3 for **C**
- 1 for (η)

I actually computed those axioms and listed them there : https://blog.remigerme.xyz/cs/cl-linear-lambda-calculus. I did not fully write them here because they look terrible and don't give insight at all, the only interesting thing is the process used to obtain them.

4. Future work

For the second trimester, two options were considered :

- a practical one, which is the formalization of the content of this preliminary report in Agda
- a theoretical one, consisting in exploring categorical combinators [4].

After thinking about it, I think I'll work on the first topic and try to formalize the results and prove them in Agda.

References

[1] J. R. Hindley and J. P. Seldin, *Lambda-Calculus and Combinators: An Introduction*, 2nd ed. Cambridge University Press, 2008.

[2] H. P. Barendregt, *The lambda calculus - its syntax and semantics*, vol. 103. in Studies in logic and the foundations of mathematics, vol. 103. North-Holland, 1985.

[3] H. B. Curry, "Grundlagen der Kombinatorischen Logik," *American Journal of Mathematics*, vol. 52, no. 3, pp. 509–536, 1930, Accessed: Jan. 05, 2025. [Online]. Available: http://www.jstor.org/stable/2370619

[4] P.-L. Curien, "Categorical combinators," *Information and Control*, vol. 69, no. 1, pp. 188–254, 1986, doi: https://doi.org/10.1016/S0019-9958(86)80047-X.